14th Symposium on Languages, Applications and Technologies

SLATE 2025, June 26–27, 2025, Faro, Portugal

Edited by

Jorge Baptista

José Barateiro

Editors

Jorge Baptista
University of Algarve, Faro, Portugal
INESC-ID Lisbon, Portugal
jbaptis@ualg.pt

José Barateiro
University of Algarve, Faro, Portugal
NOVA LINCS, Faro, Portugal
jebarateiro@ualg.pt

ACM Classification 2012
Theory of computation → Formal languages and automata theory; Computing methodologies → Natural language processing; Information systems → World Wide Web

ISBN 978-3-95977-387-4

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-387-4.

Publication date
August, 2025

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/OASIcs.SLATE.2025.0

OASIcs – OpenAccess Series in Informatics

OASIcs is a series of high-quality conference proceedings across all fields in informatics. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

Papers

14th Symposium on Languages, Applications and Technologies (SLATE 2025).
Editors: Jorge Baptista and José Barateiro
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

We are pleased to present the proceedings of the 14th Symposium on Languages, Applications and Technologies (SLATE 2025), held at the University of Algarve, Portugal, on June 26 and 27, 2025. This marks the first time SLATE is hosted in the southern region of Portugal, in the University of Algarve, reinforcing the event's commitment to academic decentralisation and broader geographical inclusion.

SLATE continues to be a transdisciplinary forum that brings together researchers and practitioners working in programming languages, natural language processing, and technology-enhanced learning. Contributions span theoretical frameworks and practical implementations, reflecting the diversity and vitality of current research at the intersection of language and technology.

This year's edition featured three distinguished keynote speakers. Pablo Gamallo (CiTIUS / Universidade de Santiago de Compostela) addressed multilingualism and language modelling in his talk Large Language Models for the Galician-Portuguese Diasystem. Pedro Rangel Henriques (Algoritmi / Universidade do Minho) explored formal knowledge representations in An Ontology for the Language Domain. Paulo Quaresma (VISTA Lab / Universidade de Évora) discussed Agentic AI: State of the Art, Challenges and Limitations, offering a critical perspective on autonomous AI systems and their implications.

All submitted papers underwent a double-blind peer review process. We extend our thanks to the Programme Committee and external reviewers for their careful assessments. The selected papers reflect high scientific quality and innovation, and we hope they serve as a valuable contribution to ongoing research in these domains.

We would like to express our gratitude to all authors, keynote speakers, reviewers, and participants. Special thanks are due to the organising committee, institutional partners, and local volunteers for their unwavering dedication and support.

We trust these proceedings will provide a stimulating reading and inspire further research collaborations across communities.

The SLATE 2025 Chairs
Jorge Baptista and José Barateiro (Universidade do Algarve)

14th Symposium on Languages, Applications and Technologies (SLATE 2025).
Editors: Jorge Baptista and José Barateiro
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Salvador Abreu (3, 8)
NOVA-LINCS, University of Évora, Portugal

Jorge Manuel Afonso Alves (7)
UNIAG, Instituto Politécnico de Bragança, Portugal

Jean-François Baffier (8)
IIJ Research, Tokyo, Japan

Jorge Baptista (1, 6, 9)
University of Algarve, Campus de Gambelas, Faro, Portugal;
INESC-ID Lisboa, Portugal

Tiago Baptista (11)
ALGORITMI Research Centre / LASI, Dept. of Informatics, University of Minho, Braga, Portugal

Fernando Batista (12)
Instituto Universitário de Lisboa (ISCTE-IUL), Portugal;
INESC-ID Lisboa, Portugal

Mauro Cardoso (12)
Instituto Universitário de Lisboa (ISCTE-IUL), Portugal;
INESC-ID Lisboa, Portugal

Khalil Chrit (8)
NOVA LINCS, University of Évora, Portugal

Francisco Coelho (3)
NOVA-LINCS, University of Évora, Portugal

Alvaro Costa Neto (10, 13)
ALGORITMI Research Centre/LASI – DI, University of Minho, Braga, Portugal;
Research Centre in Digitalization and Intelligent Robotics (CeDRI), Laboratório Associado para a Sustentabilidade e Tecnologia em Regiões de Montanha (SusTEC), Instituto Politécnico de Bragança, Portugal;
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, Barretos, Brazil

Miguel Da Corte (1, 6)
University of Algarve, Campus de Gambelas, Faro, Portugal;
INESC-ID Lisboa, Portugal

Bruno Dinis (3)
CIMA, University of Évora, Portugal

Davi Silva Eleuterio (7)
UNIAG, Instituto Politécnico de Bragança, Portugal

André Freitas (11)
ALGORITMI Research Centre / LASI, Dept. of Informatics, University of Minho, Braga, Portugal

Julio Beites Gonçalves (5)
ALGORITMI Research Centre/LASI – DI, University of Minho, Braga, Portugal

Pedro Rangel Henriques (4, 5, 10, 11, 13)
ALGORITMI Research Centre / LASI, Dep. of Informatics, University of Minho, Braga, Portugal

João Loureiro (13)
ALGORITMI Research Centre/LASI – DI, University of Minho, Braga, Portugal

Kevin Martins (2)
ALGORITMI Research Centre, University of Minho, Braga, Portugal

Paulo Matos (7)
CeDRI, Instituto Politécnico de Bragança, Portugal

Rui Mendes (2)
LASI Research Centre, University of Minho, Braga, Portugal

Izabela Müller (9)
University of Algarve, Faro, Portugal;
INESC-ID Lisboa, Portugal

Pedro Filipe Oliveira (7)
CeDRI, Instituto Politécnico de Bragança, Portugal

Vasco Manuel Oliveira (4)
Checkmarx, Braga, Portugal; ALGORITMI Research Centre/LASI, Dep. of Informatics, University of Minho, Braga, Portugal

Pedro Patinho (8)
NOVA LINCS, University of Évora, Portugal

Maria João Varanda Pereira (5, 13)
Research Centre in Digitalization and Intelligent Robotics (CeDRI), Laboratório Associado para a Sustentabilidade e Tecnologia em Regiões de Montanha (SusTEC), Instituto Politécnico de Bragança, Portugal

14th Symposium on Languages, Applications and Technologies (SLATE 2025).
Editors: Jorge Baptista and José Barateiro
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Sónia Reis (9)
University of Algarve, Faro, Portugal;
INESC-ID Lisboa, Portugal

Eugénio Ribeiro (12)
Instituto Universitário de Lisboa (ISCTE-IUL), Portugal;
INESC-ID Lisboa, Portugal

João Santos (10)
ALGORITMI Research Centre/LASI – DI, University of Minho, Braga, Portugal

Dietmar Seipel (3)
Universität Würzburg, Germany

Alberto Simões (4)
Checkmarx, Braga, Portugal;
2Ai – School of Technology, IPCA, Barcelos, Portugal;
LASI – Associate Laboratory of Intelligent Systems, Guimarães, Portugal

Committees

Organization Committee

Jorge Baptista
University of Algarve, Portugal

José Barateiro
University of Algarve, Portugal

Simão Melo de Sousa
University of Algarve, Portugal

Alberto Simões
2Ai, School of Technology, IPCA, Portugal

Scientific Committee

Alberto Simões
2Ai, School of Technology, IPCA, Barcelos, Portugal

Alexander Paar
TWT GmbH Science & Innovation, Germany

Ana Alves
University of Coimbra, Portugal

Andreas Rauber
TT Wien, Austria

António Teixeira
University of Aveiro, Portugal

António Cruz
Instituto Politécnico de Viana do Castelo, Portugal

António Menezes Leitão
Universidade Técnica de Lisboa, Portugal

Barrett Bryant
University of North Texas, USA

Bostjan Slivnik
University of Ljubljana, Slovenia

Brett Drury
Liverpool Hope University, UK

Daniela da Cruz
DataDog, Portugal

Diana Santos
Universidade de Oslo, Norway

Dietmar Seipel
University of Würzburg, Germany

Eugénio Ribeiro
INESC-ID Lisboa & ISCTE-IUL, Portugal

Fernando Baptista
INESC-ID & ISCTE-IUL, Portugal

Filipe Portela
UM & ESMAD, Portugal

Geylani Kardas
Ege University, Turkey

Giovani Librelotto
Universidade Federal de Santa Maria, Brazil

Hugo Gonçalo Oliveira
Universidade de Coimbra, Portugal

Irene Rodrigues
Universidade de Évora, Portugal

Ivan Lukovic
University of Novi Sad, Serbia

Jakub Swacha
University of Szczecin, Poland

Jan Janousek
Czech Technical University, Czech Republic

Jaroslav Poruban
Technical University of Kosice, Slovakia

João Saraiva
Universidade do Minho, Portugal

João Paulo Fernandes
Universidade de Coimbra, Portugal

Jorge Baptista
U. Algarve & L2F-Spoken Language Lab, INESC ID Lisboa, Portugal

José Barateiro
Universidade do Algarve, Portugal

José Borbinha
INESC-ID, Portugal

José Carlos Paiva
Universidade do Porto, Portugal

14th Symposium on Languages, Applications and Technologies (SLATE 2025).
Editors: Jorge Baptista and José Barateiro
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

José João Almeida
Universidade do Minho, Braga, Portugal

José Luís Sierra
Universidad Complutense de Madrid, Spain

José Paulo Leal
Faculdade de Ciências, Universidade do Porto, Portugal

Josep Silva
Universidad Politécnica de Valencia, Spain

Luísa Coheur
U.Lisboa/IST & INESC-ID Lisboa, Portugal

Marcos Garcia Gonzalez
Universidade de Santiago de Compostela, Spain

Maria João Varanda
Instituto Politécnico de Bragança, Portugal

Mario Beron
Universidad Nacional de San Luis, Argentina

Mário Pinto
Politécnico do Porto, Portugal

Marjan Mernik
Univerza v Mariboru, Slovenia

Pablo Gamallo
Universidade de Santiago de Compostela, Spain

Paulo Quaresma
Universidade de Évora, Portugal

Pedro Henriques
Universidade do Minho, Portugal

Raquel Amaro
Universidade Nova de Lisboa, FCSH, Portugal

Ricardo Rodrigues
Universidade de Coimbra, Portugal

Ricardo Queirós
IPP/ESMAD, Portugal

Ricardo Rocha
Universidade do Porto, Portugal

Ruben Solera Ureña
INESC-ID Lisboa, Portugal

Salvador Abreu
Universidade de Évora, Portugal

Sérgio Paulo
INESC-ID Lisboa, Portugal

Simão Melo de Sousa
Universidade da Beira Interior, Portugal

Susana Correia
Universidade Nova de Lisboa, FCSH, Portugal

From Prediction to Precision: Leveraging LLMs for Equitable and Data-Driven Writing Placement in Developmental Education

Miguel Da Corte
University of Algarve, Campus de Gambelas, Faro, Portugal
INESC-ID Lisboa, Portugal

Jorge Baptista
University of Algarve, Campus de Gambelas, Faro, Portugal
INESC-ID Lisboa, Portugal

Abstract

Accurate text classification and placement remain challenges in U.S. higher education, with traditional automated systems like ACCUPLACER functioning as "black-box" models with limited assessment transparency. This study evaluates Large Language Models (LLMs) as complementary placement tools by comparing their classification performance against a human-rated gold standard and ACCUPLACER. A 450-essay corpus was classified using Claude, Gemini, GPT-3.5-turbo, and GPT-4o across four prompting strategies: Zero-shot, Few-shot, Enhanced, and Enhanced+ (definitions with examples). Two classification approaches were tested: (i) a 1-step, 3 class classification task, distinguishing DevEd Level 1, DevEd Level 2, and College-level texts in one single run; and (ii) a 2-step classification task, first separating College vs. Non-College texts before further classifying Non-College texts into DevEd sublevels. The results show that structured prompt refinement improves the precision of LLMs' classification, with Claude Enhanced + achieving 62.22% precision (1 step) and Gemini Enhanced + reaching 69.33% (2 step), both surpassing ACCUPLACER (58.22%). Gemini and Claude also demonstrated strong correlation with human ratings, with Claude achieving the highest Pearson scores ($\rho = 0.75$; 1-step, $\rho = 0.73$; 2-step) vs. ACCUPLACER ($\rho = 0.67$). While LLMs show promise for DevEd placement, their precision remains a work in progress, highlighting the need for further refinement and safeguards to ensure ethical and equitable placement.

2012 ACM Subject Classification Social and professional topics → Adult education; Social and professional topics → Student assessment; Computing methodologies → Information extraction; Computing methodologies → Lexical semantics; Computing methodologies → Language resources; Computing methodologies → Natural language generation

Keywords and phrases Large Language Models (LLMs), Developmental Education (DevEd), writing assessment, text classification, English writing proficiency

Digital Object Identifier 10.4230/OASIcs.SLATE.2025.1

Supplementary Material *Software*: `https://gitlab.hlt.inesc-id.pt/u000803/deved`

Funding This work was supported by Portuguese national funds through FCT (Reference: UIDB/50021/2020, DOI: 10.54499/UIDB/50021/2020) and by the European Commission (Project: iRead4Skills, Grant number: 1010094837, Topic: HORIZON-CL2-2022-TRANSFORMATIONS-01-07, DOI: 10.3030/101094837).

1 Introduction and Objectives

Higher education institutions play a critical role in developing students' academic skills and ensuring equitable access to educational opportunities, particularly for those whose writing and reading abilities do not yet meet college-level standards. In the United States of America, this support is provided through a foundational literacy program known as Developmental

Education (DevEd). Placement in DevEd remains a persistent challenge, with traditional tools like ACCUPLACER[1] operating as "black box" automated systems. These assessments rely on machine-scored essays, emphasizing surface-level features (e.g., *word count*, *syntactic complexity*, *cohesion*) while offering limited insight into students' true writing proficiency. As a result, misplacement is common and could hinder student success [12] and progression in an academic program. In this paper, we focus on the writing skills of college-intending students at the onset of their academic career.

To address these limitations, institutions are exploring more transparent and precise assessment and course placement methods. LLMs have emerged as promising tools for placement and literacy instruction due to their writing assistance capabilities and potential for automated feedback [1, 33]. However, their reliability in assessing writing proficiency and alignment with institutional standards requires further exploration [3].

This study examines the role of LLMs in writing assessment, placement, and feature generalization within a DevEd setting. Building on previous work [10, 14], it expands an annotated corpus of 300 essays classified as Non-college level (DevEd Level 1 and DevEd Level 2) by both ACCUPLACER and human raters, adding 150 College-Level essays that were classified through the same procedures. The inclusion of College-Level texts enables a more comprehensive evaluation of LLM classification across these three proficiency levels. All writing samples were produced in English, which is the main language of instruction at the institution under study and the primary language assessed by ACCUPLACER.

In this study, we aim to:

(i) evaluate LLMs for DevEd placement by comparing their classifications to ACCUPLACER and a human-rated gold standard;

(ii) analyze the impact of a multi-step classification strategy on LLM precision across different prompting framework; and

(iii) determine whether prompt refinement mitigates classification bias, improving consistency and equity in placement outcomes.

To achieve these objectives, we propose the following research questions:

(i) How do LLM classifications compare to ACCUPLACER and human ratings in predicting placement outcomes?

(ii) What is the impact of a multi-step classification strategy on LLM precision across different prompting frameworks?

(iii) Can prompt refinement mitigate bias in LLM classification, enhancing placement accuracy and consistency across proficiency levels?

This research contributes to data-driven, equitable, and transparent DevEd placement by integrating LLMs with human-annotated corpora. By addressing systemic barriers in classification, placement, and pedagogy that hinder student success, this study explores alternative methods to refine entrance assessment tools and support linguistically underprepared students in DevEd and beyond [15]. The findings may inform more effective learning strategies, ultimately improving outcomes for students navigating writing-intensive coursework.

Our paper is structured as follows: Section 2 reviews LLMs in writing assessment and classification. Section 3 details the corpus, experimental setup, and LLMs used. Section 4 presents and discusses findings. Section 5 summarizes key contributions and future work.

[1] `https://www.accuplacer.org/` (Last accessed: 28th July 2025; all URLs in this paper were checked on this date.)

2 Related Work

Research on the use of LLMs in writing assessment, text classification, and automated feedback has gained increasing attention from higher education institutions and their policymakers [34, 41]. Several studies have evaluated LLMs, in particular to investigate their classification precision, interpretability, and feedback generation [18, 26, 42, 48] and have positioned them as potential alternatives to automated placement systems comparable to ACCUPLACER.

Despite this growing interest, we found that ACCUPLACER and similar standardized, automated systems remain widely used, yet they rely on "black-box" algorithms that prioritize surface-level linguistic features (e.g., *word count*, *syntactic complexity*) over deeper assessments of writing proficiency [16, 23] to better understand if students can communicate effectively. Some studies estimate that 30–50% of students are misplaced due to these limitations [30], raising concerns about the validity of such assessments [35]. Moreover, the proprietary nature of systems like ACCUPLACER restrict reproducibility, limit assessment transparency, and raise ethical research concerns [43].

While LLMs are also proprietary and subject to frequent updates, their integration in structured, prompt-based research provides a greater degree of process transparency and interpretability. Unlike ACCUPLACER, LLM-based classification allows researchers to design and control input conditions, evaluate model-generated justifications, and iteratively refine prompts to align with pedagogical goals. Nevertheless, we recognize that true reproducibility remains a challenge, particularly as models evolve over time. Although this caveat raises valid concerns, the promise of LLMs in advancing transparent placement practices remains a compelling one, continuing to gain traction in the field. Recent work highlights how carefully constructed LLM applications can improve placement accuracy while promoting fairness in high-stakes decisions [42, 47], which is one of the central aims of our ongoing study.

LLMs have shown promise in classification tasks across diverse text domains [42], demonstrating cost-effective assessment capabilities [38, 46]. For instance, GPT-4 and Gemini-Pro have excelled in sentiment classification for complex texts, surpassing traditional models with GPT-4 achieving the highest accuracy (0.76) [47]. GPT-4 has also outperformed Gemini in grammar correction, while Gemini excelled in sentence completion and logical reasoning [3]. In zero-shot text classification across 11 datasets, GPT-4 significantly outperformed GPT-3.5 [17]. Multi-prompting and iterative refinement strategies have been identified as effective in improving classification consistency by enabling LLMs to better internalize linguistic criteria [36], specifically, human-defined standards [29, 48].

Beyond classification, LLMs are being explored for Automated Essay Scoring (AES), where studies highlight their accuracy, consistency, and generalizability [28, 40]. When provided with clear rubrics and text examples, GPT-4 has demonstrated strong performance in AES tasks [48]. A study assessing 119 placement essays with four LLMs (Claude 2, GPT-3.5, GPT-4, and PaLM 2[2]) found that GPT-4 exhibited the highest intrarater reliability, though its performance fluctuated slightly over time [34]. Another study found GPT-4o to be the most consistent AES model (ICC above 0.7), while older GPT versions and Gemini 1.5 Flash displayed lower agreement (below 0.5) [41]. While LLMs perform well in rubric-based assessments, we noticed they struggle with discourse-level writing evaluation, particularly in areas requiring structural feedback, where human expertise is essential [34, 45].

Although the literature reveals promising results, we found that some challenges remain in writing classification due to LLMs prompt sensitivity and response variability. The literature posits that even slight variations in prompt phrasing can significantly impact classification

[2] https://ai.google.dev/palm_docs/palm

accuracy and consistency [6]. To address this, multi-step classification strategies have been proposed, where models first distinguish broad proficiency levels before refining subcategories [7, 22]. Bias in scoring also remains a concern. In an evaluation of 20 essays, for example, the GPT-4o1 model achieved the highest correlation with human ratings (Spearman's = .74) but, like GPT-4, exhibited a tendency to overrate texts [42].

LLM reliability varies with task complexity and domain. A study classifying 1,693 assessment questions based on Bloom's taxonomy [5] found traditional Machine Learning (ML) models outperforming Generative Artificial Intelligence (GenAI) models, with GPT-3.5-turbo achieving 0.62 precision versus 0.87 for traditional ML techniques, while Gemini Pro underperformed (0.43) [20]. Additionally, we learned that in [27] that Gemini Pro tends to struggle with fine-grained text comprehension tasks compared to GPT-4. Despite these limitations, LLMs have demonstrated potential in educational applications, particularly in tutoring. In an evaluation of 58,459 tutoring interactions, LearnLM[3] was preferred by expert raters over GPT-4o, Claude 3.5, and Gemini 1.5 Pro, highlighting its capacity for AI-driven learning support [45].

The studies we have here presented align with our objectives and research questions, emphasizing the complexities of assessing student writing consistently and accurately, and highlighting opportunities to improve traditional automated placement systems (e.g., ACCUPLACER) by leveraging LLMs and AI technologies. As these models evolve, improving classification precision, mitigating bias, and improving feedback quality will be paramount to ensure more reliable and equitable high-stakes assessments, particularly in DevEd [25, 28].

3 Methodology

To evaluate the precision of LLMs and their alignment with human raters for improving writing placement in DevEd, we first present a description of the corpus in Section 3.1, followed by a detailed explanation of the experimental setup in Section 3.2, along with an overview of the LLMs used and the classification experiments devised for this study.

3.1 Corpus

The corpus consists of essays written in English by college-intending students during the 2023–2024 academic year as part of Tulsa Community College's[4] standardized placement process. All essays were produced in a proctored environment without access to writing aids and were prompted by one of eleven possible reflective topics (e.g., *differences among people*, *unlimited change*) with minimal instructions.

A stratified random sampling method was employed to draw 450 essays from a larger pool of 1,000 placement essays. This sampling strategy minimized selection bias, complied with institutional data access protocols, and resulted in a corpus evenly distributed across three levels: DevEd Level 1, DevEd Level 2, and College-Level (150 texts each). The dataset combines an existing 300-essay developmental education corpus with 150 newly added College-Level texts, allowing for a more comprehensive evaluation of writing proficiency. Although demographic data was available for 67% of participants, it was excluded from the present analysis and will be examined in future research.

[3] https://ai.google.dev/gemini-api/docs/learnlm
[4] https://www.tulsacc.edu

For text-level classification, we reference an adapted version of the institution's official course descriptions:

- **DevEd Level 1**: Texts with frequent grammar, spelling, and punctuation issues, and lacking cohesion;
- **DevEd Level 2**: Texts with some structural improvements still needing targeted support; and
- **College Level**: Texts demonstrating academic-level writing with minimal errors.

All essays were also independently evaluated by two trained raters. Both raters are native English speakers who hold at least a bachelor's degree and have over five years of experience in higher education, particularly in DevEd curriculum and student writing assessment at U.S. community colleges. They were recruited through an open call for volunteers to participate in the writing assessment task. Prior to scoring, raters completed calibration sessions using classification guidelines developed by the authors of this paper [9] to ensure consistency in applying the three-level placement scale. Discrepancies were resolved through adjudication and score averaging, a widely accepted practice in classification tasks. Although only two raters were employed due to the intensive nature of the task and institutional resource constraints, their qualifications and training contributed to strong agreement. Interrater reliability [19] analysis yielded substantial agreement (K-alpha = 0.66), demonstrating a high level of consistency and reinforcing the dataset's reliability.

Table 1 presents key corpus statistics, including token distribution and classification assignments by ACCUPLACER and human raters.

Table 1 Corpus statistics and classification by level (by ACCUPLACER and human raters.

Levels	Tokens/text	Accuplacer	Human Raters
DevEd Level 1	≈ 190	150	118
DevEd Level 2	≈ 321	150	238
College Level	≈ 481	150	94
Total	**≈ 148,800**	**450**	**450**

It is pertinent to note that access to writing assessment corpora from standardized entrance exams is governed by strict protocols to protect at-risk, educationally disadvantaged student populations, ensure data privacy, and comply with institutional and ethical guidelines, which inherently limits their availability for research and broader analysis. This study adhered to these ethical considerations and followed the Institution's Review Board (IRB) protocols, under approval # 22-05.

3.2 Experimental Setup

We investigated the feasibility of using LLMs as complementary tools for placement decisions and writing instruction in DevEd. Specifically, we evaluated their classification performance against the gold standard corpus classification described in Section 3.1. Furthermore, the results of the LLM classification were then compared with the institution's existing placement system, ACCUPLACER, to assess its precision against alternative methods.

To evaluate classification performance, two classification experiments were envisaged:

- (i) a **1-step classification**, where students' texts were categorized into three classes: College-level, DevEd Level 1, and DevEd Level 2; and
- (ii) a **2-step classification**, where texts were first classified into College and DevEd categories and then further divided into DevEd Levels 1 and 2.

The transition from a **1-step** to a **2-step** classification focuses on potentially reducing initial complexity by first making a broad binary distinction (College vs. DevEd), ensuring that texts with clear college-level characteristics are correctly identified before refining classifications within the DevEd category. Since the differences between DevEd Levels 1 and 2 are often more subtle, this sequential approach allows for a more targeted classification between proficiency levels [18].

LLMs Selection

Four LLMs were tested and evaluated using their default hyperparameters. This study adopts the commonly accepted definition of large language models as pre-trained, transformer-based architectures [38]. Model selection was based on their wide availability via API access and their representation of varying tiers of generative performance, reasoning ability, and cost-efficiency – factors frequently highlighted in the literature as critical for educational applications and scalable implementation [18, 26, 47].

Model selection was also guided by practical considerations, including task complexity, processing speed, and data security – criteria commonly emphasized in applied LLM research and deployment frameworks. While the selected models align with current literature and offer broad applicability, this does not preclude the inclusion of additional or emerging models (e.g., DeepSeek) in future iterations of this research. The four LLMs included in this study are listed below in alphabetical order:

- **Claude**[5] [2], developed by Anthropic, is known to provide a more accessible API and structured responses, making it easier to evaluate for classification consistency [6]. It has also been reported to have better consistency in logical reasoning [4], which may enhance its performance in tasks requiring nuanced proficiency distinctions (e.g., DevEd Level 1 vs. Level 2). Claude 3.5 Sonnet (October 2024 version) was the model used.
- **Gemini**[6] [37], unlike the text-based LLMs GPT-3.5-turbo and GPT-4o, is designed for cross-modal reasoning and trained on diverse web sources, including educational content [44], which may enhance its ability to recognize academic proficiency markers and improve efficiency in handling texts with high linguistic variability [24, 27]. For this study, Gemini 2.0 Flash was used.
- **OpenAI's ChatGPT**[7]: (i) **GPT-3.5-turbo**, while computationally efficient [32], it is leveraged to assess any improvements in classification consistency with the more advanced, latest cost-efficient model [39], (ii) **GPT-4o**, in the task of DevEd placement context. The advanced model is explored for its strengths in text comprehension and structured response generation, which could not only help students identify writing weaknesses but also enhance the provision of targeted, context-aware feedback [27]. The experiments use the GPT models via the official OpenAI API.

LLMs Classification Experiments

To systematically examine how prompt structuring influences classification precision, we incorporated a four-stage prompting approach into the two experiments described in Section 3.2: (i) 1-step classification and (ii) 2-step classification.

[5] `https://docs.anthropic.com/en/home`
[6] `https://deepmind.google/technologies/gemini/`
[7] `https://chat.openai.com`

Prompt construction was guided by existing literature on LLM-based classification and rubric-based writing assessment [36, 48], ensuring that the design was not ad hoc. Prompts were iteratively refined to maximize linguistic clarity, align with defined classification objectives, and maintain compatibility across model architectures. These human-authored prompts were designed to capture patterns that distinguish proficient from non-proficient writing [31]. The complete series of prompts was archived for public access via GitHub[8] [11].

To evaluate whether increasing prompt specificity improved classification outcomes, we introduced a four-stage prompting sequence. Precision, the primary evaluation metric in this study, assessed the reliability of a model's positive predictions aligning with human-assigned classifications [21]. This focus is particularly relevant in DevEd contexts, where misclassification, whether through overplacement or underplacement, can significantly affect students' academic trajectories. The four prompting stages were as follows:

- (i) **Zero-shot**, in which the models classified texts following a prompt with a laconic approach. No sample texts were provided at this stage in the classification prompt. Due to the availability of comparable data, in terms of model precision, results from this approach establish the *baseline* for evaluation with the subsequent prompting strategies.
- (ii) **Few-shot**, in which the models received the same prompt but with basic definitions of each proficiency level, adapted from the Institution's official course descriptions and curriculum. No sample texts were provided at this stage either.
- (iii) **Enhanced**, in which the classification prompt included detailed descriptions of proficiency levels and key linguistic markers that encompassed grammatical accuracy, syntactic complexity, lexical richness, and discourse cohesion. No sample texts were provided at this stage.
- (iv) **Enhanced+**, this approach extends the **Enhanced** classification prompting by incorporating three manually assessed and classified sample texts (using ACCUPLACER's linguistic descriptors), each corresponding to the three proficiency levels already mentioned (DevEd Level 1, DevEd Level 2, and College-level). These samples were randomly selected from outside the corpus but drawn from the same database and time epoch.

The four-stage experimental framework aligned with the research questions in Section 1, allowing for a comparative analysis of how LLMs classify writing proficiency under varying levels of linguistic guidance. This approach evaluated classification performance with and without explicit linguistic criteria and concrete linguistic evidence (text samples). Across all prompting strategies, LLMs were also instructed to provide classification rationales, which, while more robust than ACCUPLACER, remain less comprehensive than human evaluations – an aspect to be explored in a subsequent study. A sample of this feedback is included in Appendix A.

4 Experimental Results

This section presents the results of the two classification experiments[9] described in Section 3.2: (i) 1-step classification and (ii) 2-step classification. In Sections 4.1 and 4.2, we analyze LLMs' performance across the four-stage prompting approaches, evaluating their effectiveness in distinguishing proficiency levels.

[8] `https://gitlab.hlt.inesc-id.pt/u000803/deved`
[9] All experiments were conducted over eight days, from February 18 to February 25, 2025.

4.1 1-step Classification: 3 classes

The 1-step classification experiment required LLMs to simultaneously differentiate among three proficiency levels – College-Level, DevEd Level 1, and DevEd Level 2 – without an intermediate binary decision. This method directly assessed the models' ability to distinguish college-ready writing from varying degrees of developmental writing in a single step.

Table 2 presents each LLMs' precision in classifying texts across these three levels under four prompting strategies. The results include precision figures for each level compared to the gold standard, as well as the overall model performance. A "+" next to the overall precision score indicates improvement relative to the Zero-shot approach (baseline), while a "−" denotes a decline. Results are analyzed in terms of overall precision first before comparing them to ACCUPLACER. Additionally, a confusion matrix of the best-performing model is included to provide a more detailed breakdown of classification outcomes relative to the gold standard.

Table 2 LLM and ACCUPLACER classification precision vs. gold standard, per level and overall.

LLM / Experiment	Level 1	Level 2	College	Overall Precision
Claude Zero-shot	44.44%	60.00%	72.09%	52.44%
Claude Few-shot	53.65%	61.73%	86.67%	59.1% (+6.66%)
Claude Enhanced	50.00%	66.83%	85.71%	60.89% (+8.45%)
Claude Enhanced+	50.89%	73.65%	73.08%	**62.22** % (**+9.78**%)
Gemini Zero-shot	41.16%	70.00%	64.38%	51.33%
Gemini Few-shot	42.86%	60.12%	81.08%	52.44% (+1.11%)
Gemini Enhanced	29.29%	64.04%	84.62%	42.89% (-8.44%)
Gemini Enhanced+	52.53%	63.76%	78.26%	59.56% (+8.23%)
GPT-3.5-turbo Zero-shot	28.78%	20.69%	81.82%	29.56%
GPT-3.5-turbo Few-shot	38.75%	65.04%	68.42%	48.44% (+18.88%)
GPT-3.5-turbo Enhanced	55.33%	59.26%	40.54%	53.33% (+23.77%)
GPT-3.5-turbo Enhanced+	43.85%	57.09%	43.75%	51.11% (+21.55%)
GPT-4o Zero-shot	36.59%	49.14%	88.24%	41.78%
GPT-4o Few-shot	43.95%	65.38%	78.26%	54.89% (+13.11%)
GPT-4o Enhanced	41.54%	69.53%	74.19%	54.00% (+12.22%)
GPT-4o Enhanced+	49.09%	70.76%	79.66%	61.33% (+19.55%)
Accuplacer	51.33%	68.67%	54.67%	58.22%

In the **Zero-shot** experiment, Claude achieved the highest classification precision (52.44%), followed closely by Gemini (51.33%), outperforming all other LLMs. The remaining models ranked in descending order of precision as GPT-4o > GPT-3.5-turbo.

In the **Few-shot** experiment, when given basic proficiency definitions, Claude remained the best-performing model (59.1%), showing noticeable gains (approximately +6.66%) from the Zero-shot run. The ranking of models in this experiment was: GPT-4o > Gemini > GPT-3.5-turbo.

In the **Enhanced** experiment, when refined linguistic descriptors were introduced, Claude maintained its leading position (60.89%). However, not much improvement (+1.79%) was observed from the previous setup (Claude Few-shot). The remaining models ranked as follows: GPT-4o > GPT-3.5-turbo > Gemini.

Lastly, in the **Enhanced+** experiment, Claude continued to perform best (62.22%), with a 10.22% improvement over its Zero-shot outcome, but only +1.33% improvement over the previous setup. The ranking, this time, was GPT-4o > Gemini > GPT-3.5-turbo. Notably, the highest gains in precision from the Zero-shot to the enhanced+ experiments were seen in GPT-3.5-turbo (21.55%) and GPT-4o (19.55%).

When compared to ACCUPLACER, three models, Claude (4%), GPT-4o (3.11%), and Gemini (1.34%), achieved slightly higher overall precision in the Enhanced+ experiments. Notably, Claude outperformed ACCUPLACER also across the Few-shot and Enhanced prompting strategies. While LLMs show potential for structured DevEd placement tasks compared to ACCUPLACER, they still present limitations, as evidenced by their low precision scores.

To complement this analysis, Figure 1 provides a visual representation of the classification precision scores distribution of LLMs and ACCUPLACER in the 1-step classification experiment.

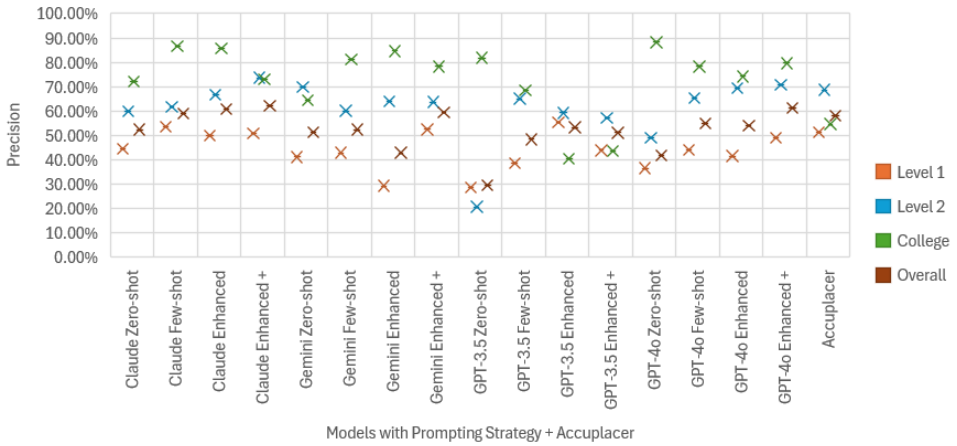


Figure 1 LLM and ACCUPLACER classification precision distribution per level and overall.

To better understand Claude's overall performance, given its highest classification precision in all prompting strategies, Table 3 displays the confusion matrix with the model's distribution of actual versus predicted classifications in its best performance, the enhanced+ experiment:

Table 3 Confusion matrix of predicted (LLM) vs. actual classifications (gold standard) of Claude in the Enhanced+ experiment.

↓ Actual / Predicted →	Level 1	Level 2	College	Total
Level 1	**114**	4	0	118
Level 2	108	**109**	21	238
College	2	35	**57**	94
Total	224	148	78	450

The confusion matrix analysis revealed that Claude excels in identifying Level 1 texts, correctly classifying nearly all (114/118) with only four misclassified as Level 2, indicating a strong grasp of beginning-level writing. However, its performance dropped significantly for Level 2, correctly identifying less than half of these texts. A strong tendency to underclassify was observed, as 108 Level 2 texts are misclassified as Level 1, and 21 as College-Level. Claude demonstrated good discrimination of advanced writing, correctly classifying 57 out of 94 College-Level texts, with most misclassifications occurring just one level below. Extreme errors were rare, with only two College-Level texts misclassified as Level 1. Overall, the model underclassified far more often than it overclassified (145 vs. 25 instances), suggesting it applied a stricter threshold for advancement compared to the human gold standard.

Based on the classification results obtained, further statistical validation is necessary. Pearson correlation coefficients (ρ) are computed next to assess the reliability and alignment of LLM-generated classifications with the gold standard.

Correlation of LLMs Classifications with Gold Standard

Pearson correlation coefficients (ρ) measured the degree of linear association between the classification levels assigned by LLMs and the gold standard in the 1-step experiment. A higher ρ value indicates a stronger correlation, meaning the LLMs' predictions align more closely with human-assigned levels.

Pearson correlation (ρ) scores were interpreted using SPSS guidelines[10], based on Cohen (1988) [8], which define correlation strength as follows:

$0.1 < |r| < 0.3$ indicates *small/weak* correlation (W);
$0.3 < |r| < 0.5$ corresponds to *medium/moderate* correlation (M); and
$0.5 < |r| \ldots$ denotes *large/strong* correlation (S).

The computed Pearson correlation coefficients (ρ) for each system are presented in Table 4.

Table 4 Pearson scores of LLMs and ACCUPLACER vs. gold standard. Correlation interpretation: substantial (S); moderate (M).

LLM/System	Zero-shot	Few-shot	Enhanced	Enhanced+
Claude	0.67 (S)	0.66 (S)	0.67 (S)	**0.75 (S)**
Gemini	0.60 (S)	0.62 (S)	0.62 (S)	0.63 (S)
GPT-3.5-turbo	0.40 (M)	0.54 (S)	0.51 (S)	0.39 (M)
GPT-4o	0.63 (S)	0.60 (S)	0.60 (S)	0.69 (S)
Accuplacer	**0.67 (S)**			

The results indicated that most LLMs exhibited a strong correlation with the gold standard, with Claude Enhanced+ achieving the highest alignment ($\rho = 0.75$), surpassing ACCUPLACER ($\rho = 0.67$). GPT-4o also outperformed ACCUPLACER but just by a smaller margin (0.02%) in the Enhanced+ run as well. In the Zero-shot and Enhanced prompting strategies, Claude matched ACCUPLACER's Pearson score, while the remaining models and configurations consistently underperformed.

Notably, GPT-3.5-turbo Zero-shot showed one of the weakest correlation scores ($\rho = 0.40$), but its performance improved significantly in the Few-shot ($\rho = 0.54$) and Enhanced ($\rho = 0.51$) settings before declining again in Enhanced+ ($\rho = 0.39$). This fluctuation suggests that models such as GPT-3.5-turbo may be more sensitive to prompt variations than others.

The strong correlation observed with Claude Enhanced+, particularly in such structured classification settings, highlights the potential of certain LLMs to approximate human-level proficiency in classification, though inconsistencies across models require further investigation.

4.2 2-step Classification: 3 classes

The 2-step classification experiment is followed in an attempt to address the risk of models disproportionately assigning texts to a particular category. Dividing a classification problem into two steps has shown to improve the overall precision of certain LLMs [29, 48]. To this extent, all 450 texts were initially classified as either College-Level or DevEd Level. Then, the DevEd-level texts, specifically, underwent a second classification, further distinguishing between DevEd Level 1 and DevEd Level 2. While the prompting methodology remained consistent (compared to the 1-step), level definitions and provided examples were slightly adjusted to align with the objectives of this classification experiment.

[10] `https://libguides.library.kent.edu/SPSS/PearsonCorr`

Table 5 presents the final precision scores[11] for each LLM (per level) across all four prompting strategies. A "+" next to the overall precision score indicates improvements relative to the Zero-shot approach (baseline), while a "−" represents the opposite. [13]

Table 5 LLM and ACCUPLACER classification precision vs. gold standard, per level and overall.

LLM / Experiment	Level 1	Level 2	College	Overall Precision
Claude Zero-shot	46.19%	74.07%	63.21%	56.89%
Claude Few-shot	51.72%	76.67%	57.48%	60.00% (+3.11%)
Claude Enhanced	52.88%	77.60%	58.97%	61.33% (+4.44%)
Claude Enhanced+	57.95%	77.36%	62.61%	66.00 % (+9.11%)
Gemini Zero-shot	48.83%	60.37%	82.61%	56.07%
Gemini Few-shot	57.79%	63.33%	76.92%	62.22% (+6.15%)
Gemini Enhanced	54.49%	61.92%	73.91%	59.78% (+3.71%)
Gemini Enhanced+	**66.67%**	**69.50%**	**73.85%**	**69.33% (+13.26%)**
GPT-3.5-turbo Zero-shot	45.29%	58.97%	43.62%	47.11%
GPT-3.5-turbo Few-shot	48.80%	64.52%	54.22%	50.89% (+3.78%)
GPT-3.5-turbo Enhanced	37.20%	65.00%	61.40%	46.44% (-0.67%)
GPT-3.5-turbo Enhanced+	46.24%	62.76%	47.06%	53.56% (+6.44%)
GPT-4o Zero-shot	40.81%	59.26%	72.09%	49.33%
GPT-4o Few-shot	34.24%	48.54%	82.35%	39.33% (-10.00%)
GPT-4o Enhanced	35.09%	51.38%	89.47%	41.33% (-8.00%)
GPT-4o Enhanced+	48.91%	73.83%	73.61%	61.11% (+11.78%)
Accuplacer	51.33%	68.67%	54.67%	58.22%

In the **Zero-shot** experiment, Claude demonstrated the highest classification precision (56.89%), closely followed by Gemini with only a 0.82% difference. The ranking of the remaining models in decreasing precision is GPT-4o > GPT-3.5-turbo.

In the **Few-shot** experiment, Gemini outperformed Claude by 2.22%. Both models showed modest improvements over their Zero-shot runs, with Claude achieving a 3.11% increase and Gemini, with almost twice the precision gain, achieving a 6.15%. The ranking of the remaining models was reversed from the Zero-shot experiment, as GPT-3.5-turbo outperformed GPT-4o. Notably, GPT-4o exhibited a significant 10% decrease in precision, indicating that added proficiency definitions did not necessarily enhance its performance.

In the **Enhanced** experiment, Claude achieved the highest precision score (61.33%), outperforming all other models. However, its improvement over the Few-shot setup was minimal (+1.33%), despite the inclusion of more precise linguistic descriptors and features in the prompt. Gemini ranked second, showing a 3.71% gain from its baseline but a 2.44% drop from the Few-shot setup, suggesting inconsistent benefits from increased prompt specificity. The remaining models ranked as follows: GPT-3.5-turbo > GPT-4o.

In the **Enhanced+** experiment, Gemini demonstrated the most significant gains, achieving an overall precision of 69.33%. Claude followed closely with a precision score of 66%, while GPT-4o ranked third with a precision score of 61.11%. GPT-3.5-turbo had the lowest precision in this setup, scoring 53.56%. All four models demonstrated performance gains from each of their baselines (Zero-shot) and the Enhanced run. However, Claude was the model to consistently improve its precision across all prompting strategies, suggesting that incorporating text samples alongside linguistic guidance enhances classification precision.

[11] This study is part of an ongoing research project. While the textual portion of the corpus cannot be released at this stage, a dataset containing the full classification results from the 2-step experiment – where overall precision scores exceeded those of the 1-step approach – has been made available for evaluation at `https://gitlab.hlt.inesc-id.pt/u000803/deved/` [13]. Full corpus access will be available upon completion of the study.

When compared to ACCUPLACER, Gemini and Claude demonstrated the highest overall precision in the Enhanced+ experiments, with notable gains of 11.11% and 7.78%, respectively. Both models also outperformed ACCUPLACER in the Few-shot and Enhanced strategies. GPT-4o showed a smaller improvement of 2.89% in the Enhanced+ strategy, while GPT-3.5-turbo consistently underperformed throughout. Despite their enhanced potential for structured DevEd placement, LLMs still exhibit some limitations, as reflected in their precision scores.

To complement this analysis, Figure 2 provides a visual representation of the classification precision scores distribution of LLMs and ACCUPLACER in the 2-step classification approach.

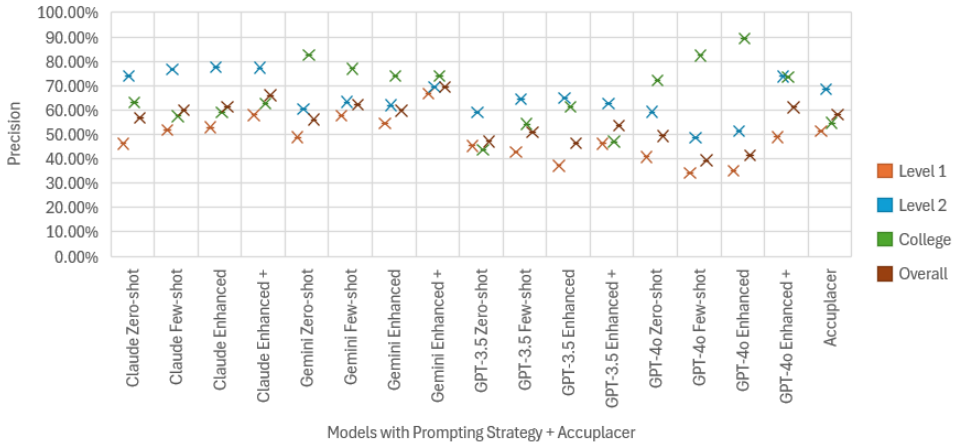


Figure 2 LLM and ACCUPLACER classification precision distribution, per level and overall.

Table 6 shows a confusion matrix detailing the model's distribution of actual versus predicted classifications in its best performance, the Enhaced+ experiment:

Table 6 Confusion matrix of predicted (LLM) vs. actual classifications (gold standard) in the Gemini Enhanced+ experiment.

↓ Actual / Predicted →	Level 1	Level 2	College	Total
Level 1	**84**	33	1*	118
Level 2	42	**180**	16	238
College	0	46	**48**	94
Total	126	259	65	450

Almost all errors (137 out of 138 misclassifications) occurred between adjacent levels, suggesting the model understands the ordinal relationship between levels. When Gemini made errors on Level 1 texts, it nearly always overclassified texts by just one level. The single outlier*, a text classified by Gemini as College-level but by human raters as DevEd Level 1, was closely examined. This text contained a single string of tokens, yet it was the shortest in the dataset.

Text 13: "Our ability to change ourselves is limitless."

Notably, Gemini distinguished it as a "statement" rather than a "text," which is the terminology used in all other justifications. The model's justification *verbatim* was:

> This is a concise, grammatically correct *statement* expressing a complex idea. It demonstrates strong control of language and does not contain any foundational errors. The sentence structure is simple but effective, conveying a clear and impactful message.

Gemini, additionally, showed some bias toward classifying texts as Level 2, which is also the level involved in most errors, both as the actual and predicted class. Overall, Gemini is more likely to underclassify than overclassify Level 2 texts, suggesting Level 2 is the most difficult to distinguish. On the contrary, the model never misclassified College texts as Level 1, showing excellent discrimination between these distant categories. As exhibited in Table 5, in the Enhanced+ run, Gemini maintains similar precision across all levels (≈67-74%), indicating balanced performance rather than excelling at one level at the expense of others.

Correlation of LLMs Classifications with Gold Standard

Similarly to the 1-step experiment, Pearson correlation coefficients (ρ) were also computed. Results are presented in Table 7.

Table 7 Pearson scores of LLMs and ACCUPLACER vs. gold standard. Correlation interpretation: substantial (S); moderate (M).

LLM/System	Zero-shot	Few-shot	Enhanced	Enhanced+
Claude	0.68 (S)	0.69 (S)	0.71 (S)	**0.73 (S)**
Gemini	0.64 (S)	0.62 (S)	0.58 (S)	0.69 (S)
GPT-3.5-turbo	0.59 (M)	0.54 (S)	0.49 (M)	0.45 (M)
GPT-4o	0.61 (S)	0.49 (M)	0.54 (S)	0.70 (S)
Accuplacer	**0.67 (S)**			

Compared to the 1-step classification, Pearson correlation calculations revealed that most LLMs exhibit a strong alignment with the gold standard. Claude Enhanced+ achieved the highest correlation ($\rho = 0.73$), followed closely by Claude Enhanced ($\rho = 0.71$) and GPT-4o ($\rho = 0.70$), outperforming ACCUPLACER ($\rho = 0.67$). There was a minimal difference (0.02% points) in Pearson scores observed between the 1-step and 2-step classifications for Claude Enhanced+, hinting at the model's stability in its classification performance.

The strong correlation observed in Claude and GPT-4o reinforces the idea that LLMs, especially in a 2-step classification, can closely align with human-assigned proficiency levels.

4.3 Precision Comparison: 1-step vs. 2-step classification experiments

Having conducted both 1-step and 2-step classification experiments using four prompting strategies, we highlight in Table 8 the top 3 major precision gains (green) and losses (red), per level and overall, to assess how LLMs respond to structured classification strategies. The figures represent the difference in precision between the 2-step and 1-step experiments, as overall precision scores were higher in the 2-step. This comparison directly supports the analysis of research questions 2 and 3 in Section 1, evaluating the impact of prompt refinement on classification precision and bias mitigation.

Most models improved in overall precision with the 2-step classification, particularly GPT-3.5-turbo Zero-shot (+17.55%), Gemini Enhanced (+16.89%), and Gemini Few-shot (+9.78%). Although not among the top three, Gemini Enhanced+ performed similarly to Gemini Few-shot, with only a 0.01% difference. Conversely, GPT-4o struggled to maintain positive precision gains across three out of the four prompting strategies, suggesting that a 2-step classification strategy may not be as effective for this particular model.

We observed significant gains in Level 1 in two of the four prompting strategies, particularly for Gemini (Few-shot: +14.93%, Enhanced: +25.20%) and GPT-3.5-turbo Zero-shot (+16.51%). Particularly for GPT-3.5-turbo Zero-shot, it also saw the sharpest trade-off,

Table 8 Precision gains and losses between 1 and 2-step experiments across prompting strategies.

LLM / Experiment	Level 1	Level 2	College	Overall
Claude Zero-shot	+1.75%	+14.07%	-8.88%	+4.45%
Claude Few-shot	-1.93%	+14.94%	-29.19%	+0.90%
Claude Enhanced	+2.88%	+10.77%	-26.74%	+0.44%
Claude Enhanced+	+7.06%	+3.71%	-10.47%	+3.78%
Gemini Zero-shot	+7.67%	-9.63%	+18.23%	+4.74%
Gemini Few-shot	+14.93%	+3.21%	-4.16%	+9.78%
Gemini Enhanced	+25.20%	-2.12%	-10.71%	+16.89%
Gemini Enhanced+	+14.14%	+5.74%	-4.41%	+9.77%
GPT-3.5-turbo Zero-shot	+16.51%	+38.28%	-38.20%	+17.55%
GPT-3.5-turbo Few-shot	+4.05%	-0.52%	-14.20%	+2.45%
GPT-3.5-turbo Enhanced	-18.13%	+5.74%	+20.86%	-6.89%
GPT-3.5-turbo Enhanced+	+2.39%	+5.67%	+3.31%	+2.45%
GPT-4o Zero-shot	+4.22%	+10.12%	-16.15%	+7.55%
GPT-4o Few-shot	-9.71%	-16.84%	+4.09%	-15.56%
GPT-4o Enhanced	-6.45%	-18.15%	+15.28%	-12.67%
GPT-4o Enhanced+	-0.18%	+3.07%	-6.05%	-0.22%

improving +38.28% in Level 2 but dropping nearly the same amount (-38.20%) in College-Level classification. Similarly, for Level 2, Claude Zero and Few-shot showed notable gains (+14.07% and +14.94%, respectively) but substantial losses at the College-Level (-29.19% and -26.74%) in the Few-shot and Enhanced runs.

Overall, most results suggest that breaking down classification into distinct steps improved precision. Among all models, Gemini demonstrated the most consistent performance across levels and prompting strategies. The trade-off between gains in lower proficiency levels and losses in College-Level precision suggests that some models, particularly Claude and GPT-3.5-turbo, may be more in-tuned with DevEd writing patterns, prioritizing features typical of lower-level texts while misclassifying more advanced writing as DevEd.

5 Conclusions and Future Work

We evaluated four LLMs as data-driven alternatives for equitable DevEd placement, offering (potential) competitive, cost-effective suggestions for higher education institutions traditionally reliant on "black-box" systems like ACCUPLACER. Our findings indicated that both classification structure and prompt refinement played a critical role in enhancing LLM precision when distinguishing between DevEd and College-level texts.

In terms of classification structure, across the 1 and 2-steps classification experiments, models exhibited higher precision in Level 2 and College-Level classifications but struggled with Level 1, likely due to the inconsistent structures and frequent grammatical errors in Level 1 texts. Compared to the 1-step experiment, the 2-step classification saw Gemini Enhanced+ achieved the highest precision (66.67%) in Level 1, reinforcing the notion that breaking classification into multiple steps enhances precision [29, 48], and the importance of incorporating clear linguistic criteria with sample texts in the classification prompts [36]. We noticed, too, that Claude and Gemini consistently underclassified Level 2 texts, suggesting they apply a stricter advancement threshold compared to human raters. This likely occurs because the models are more conservative in assigning texts to higher proficiency levels, meaning they are more prone to classifying Level 2 texts as Level 1 rather than correctly identifying them.

Claude and Gemini demonstrated a strong correlation with human ratings, with Claude achieving the highest Pearson scores ($\rho = 0.75$; 1-step, $\rho = 0.73$; 2-step), surpassing ACCUPLACER ($\rho = 0.67$). These findings confirm that LLMs have the potential approximate human-level proficiency classification, yet their precision remains a work in progress, particularly for high-stakes placement decisions. Given the social and economic impact of misplacement, safeguards are necessary to ensure an ethical and fair assessment of students' skills.

To refine the role of LLMs in placement decisions, our future research will: (i) explore the expansion of the dataset to increase generalizability; (ii) further refine prompting strategies to mitigate classification inconsistencies; (iii) test newer, more sophisticated LLMs for improved performance; (iv) investigate LLM-generated feedback to evaluate its effectiveness in justifying classification decisions and enhancing instructional usability in teaching practices. By addressing these areas, future studies will continue to contribute to more transparent, data-driven placement methodologies that balance automation with fairness and reliability.

References

1 Tufan Adiguzel, Mehmet Haldun Kaya, and Faith Kürşat Cansu. Revolutionizing education with AI: Exploring the transformative potential of ChatGPT. *Contemporary Educational Technology*, 15(3):ep429, 2023. doi:10.30935/cedtech/13152.

2 AI Anthropic. The Claude 3 model family: Opus, Sonnet, Haiku. *Claude-3 Model Card*, 1:1, 2024.

3 Nathan Atox and Mason Clark. Evaluating Large Language Models through the lens of linguistic proficiency and world knowledge: A comparative study. *Authorea Preprints*, 2024.

4 Jaehyeon Bae, Seoryeong Kwon, and Seunghwan Myeong. Enhancing software code vulnerability detection using gpt-4o and claude-3.5 sonnet: A study on prompt engineering techniques. *Electronics*, 13(13), 2024. doi:10.3390/electronics13132657.

5 Benjamin Samuel Bloom. *Taxonomy of educational objectives: Handbook II*. David McKay, 1956.

6 Loredana Caruccio, Stefano Cirillo, Giuseppe Polese, Giandomenico Solimando, Shanmugam Sundaramurthy, and Genoveffa Tortora. Claude 2.0 Large Language Model: Tackling a real-world classification problem with a new iterative prompt engineering approach. *Intelligent Systems with Applications*, 21:200336, 2024. doi:10.1016/J.ISWA.2024.200336.

7 Xiang Chen, Ningyu Zhang, Xin Xie, Shumin Deng, Yunzhi Yao, Chuanqi Tan, Fei Huang, Luo Si, and Huajun Chen. KnowPrompt: Knowledge-aware prompt-tuning with synergistic optimization for relation extraction. In *Proceedings of the ACM Web Conference 2022*, WWW '22, pages 2778–2788, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3485447.3511998.

8 Jacob Cohen. *Statistical power analysis*. Hillsdale, NJ: Erlbaum, 1988.

9 Miguel Da Corte and Jorge Baptista. Enhancing writing proficiency classification in Developmental Education: the quest for accuracy. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 6134–6143, 2024. URL: https://aclanthology.org/2024.lrec-main.542.

10 Miguel Da Corte and Jorge Baptista. Leveraging NLP and machine learning for English (l1) writing assessment in Developmental Education. In *Proceedings of the 16th International Conference on Computer Supported Education (CSEDU 2024), 2-4 May, 2024, Angers, France*, volume 2, pages 128–140, 2024. doi:10.5220/0012740500003693.

11 Miguel Da Corte and Jorge Baptista. LLM-based writing classification prompts for 1-step and 2-step classification experiments. Technical report, University of Algarve & INESC-ID Lisboa, 2025.

12 Miguel Da Corte and Jorge Baptista. Refining English writing proficiency assessment and placement in Developmental Education using NLP tools and Machine Learning. In *Proceedings of the 17th International Conference on Computer Supported Education - Volume 2: CSEDU*, pages 288–303. INSTICC, SciTePress, 2025. doi:10.5220/0013351500003932.

13 Miguel Da Corte and Jorge Baptista. Results for LLM-based 2-step classification. Technical report, University of Algarve & INESC-ID Lisboa, 2025.

14 Miguel Da Corte and Jorge Baptista. Toward consistency in writing proficiency assessment: Mitigating classification variability in Developmental Education. In *Proceedings of the 17th International Conference on Computer Supported Education - Volume 2: CSEDU*, pages 139–150. INSTICC, SciTePress, 2025. doi:10.5220/0013353900003932.

15 Jane Denison-Furness, Stacey Lee Donohue, Annemarie Hamlin, and Tony Russell. Welcome/Not Welcome: From Discouragement to Empowerment in the Writing Placement Process at Central Oregon Community College. In Jassica Nastal, Mya Poe, and Christie Toth, editors, *Writing Placement in Two-Year Colleges: The Pursuit of Equity in Postsecondary Education*, pages 107–127. The WAC Clearinghouse/University Press of Colorado, 2022. doi:10.37514/PRA-B.2022.1565.2.04.

16 Nikki Edgecombe and Michael Weiss. Promoting equity in Developmental Education reform: A conversation with Nikki Edgecombe and Michael Weiss. *Center for the Analysis of Postsecondary Readiness*, page 1, 2024.

17 Jessica López Espejel, El Hassane Ettifouri, Mahaman Sanoussi Yahaya Alassan, El Mehdi Chouham, and Walid Dahhane. GPT-3.5, GPT-4, or BARD? evaluating LLMs reasoning ability in zero-shot setting and performance boosting through prompts. *Natural Language Processing Journal*, 5:100032, 2023. doi:10.1016/J.NLP.2023.100032.

18 John Fields, Kevin Chovanec, and Praveen Madiraju. A survey of text classification with transformers: How wide? how large? how long? how accurate? how expensive? how safe? *IEEE Access*, 12:6518–6531, 2024. doi:10.1109/ACCESS.2024.3349952.

19 Deen Freelon. Recal OIR: ordinal, interval, and ratio intercoder reliability as a web service. *International Journal of Internet Science*, 8(1):10–16, 2013.

20 Mohammed Osman Gani, Ramesh Kumar Ayyasamy, Saadat M Alhashmi, Khondaker Sajid Alam, Anbuselvan Sangodiah, Khondaker Khaleduzzman, and Chinnasamy Ponnusamy. Towards enhanced assessment question classification: a study using Machine Learning, deep learning, and Generative AI. *Connection Science*, 37(1):2445249, 2025. doi:10.1080/09540091.2024.2445249.

21 Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756*, 2020. arXiv:2008.05756.

22 Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. Tabllm: Few-shot classification of tabular data with Large Language Models. In *International Conference on Artificial Intelligence and Statistics*, pages 5549–5581. PMLR, 2023. URL: https://proceedings.mlr.press/v206/hegselmann23a.html.

23 Sarah Hughes and Ruth Li. Affordances and limitations of the ACCUPLACER automated writing placement tool. *Assessing Writing*, 41:72–75, 2019.

24 Muhammad Imran and Norah Almusharraf. Google Gemini as a next generation AI educational tool: a review of emerging educational technology. *Smart Learning Environments*, 11(1):22, 2024. doi:10.1186/S40561-024-00310-Z.

25 Elizabeth Kopko, Jessica Brathwaite, and Julia Raufman. The next phase of placement reform: Moving toward equity-centered practice. research brief. *Center for the Analysis of Postsecondary Readiness*, 2022. URL: https://files.eric.ed.gov/fulltext/ED626145.pdf.

26 Milan Kostic, Hans Friedrich Witschel, Knut Hinkelmann, and Maja Spahic-Bogdanovic. LLMs in automated essay evaluation: A case study. In *Proceedings of the AAAI Symposium Series*, volume 3, pages 143–147, 2024. doi:10.1609/AAAISS.V3I1.31193.

27 Gyeong-Geon Lee, Ehsan Latif, Lehong Shi, and Xiaoming Zhai. Gemini Pro defeated by GPT-4v: Evidence from education. *arXiv preprint arXiv:2401.08660*, 2023.

28 Stephanie Link and Svetlana Koltovskaia. *Automated Scoring of Writing*, pages 333–345. Springer International Publishing, Cham, 2023. doi:10.1007/978-3-031-36033-6_21.

29 Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in Natural Language Processing. *ACM Computing Surveys*, 55(9):1–35, 2023. doi:10.1145/3560815.

30 Ross Markle. Redesigning course placement in service of guided pathways. *Educational Considerations*, 50(2):8, 2025.

31 Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. Recent advances in Natural Language processing via Large pre-trained Language Models: A survey. *ACM Computing Surveys*, 56(2):1–40, 2023. doi:10.1145/3605943.

32 Yida Mu, Ben P. Wu, William Thorne, Ambrose Robinson, Nikolaos Aletras, Carolina Scarton, Kalina Bontcheva, and Xingyi Song. Navigating prompt complexity for zero-shot classification: A study of large language models in computational social science. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 12074–12086, Torino, Italia, May 2024. ELRA and ICCL. URL: https://aclanthology.org/2024.lrec-main.1055/.

33 Sasha Nikolic, Carolyn Sandison, Rezwanul Haque, Scott Daniel, Sarah Grundy, Marina Belkina, Sarah Lyden, Ghulam M Hassan, and Peter Neal. ChatGPT, Copilot, Gemini, SciSpace and Wolfram versus higher education assessments: an updated multi-institutional study of the academic integrity impacts of Generative Artificial Intelligence (GenAI) on assessment, teaching and learning in engineering. *Australasian Journal of Engineering Education*, 29(2):126–153, 2024.

34 Austin Pack, Alex Barrett, and Juan Escalante. Large language models and automated essay scoring of english language learner writing: Insights into validity and reliability. *Computers and Education: Artificial Intelligence*, 6:100234, 2024. doi:10.1016/J.CAEAI.2024.100234.

35 Dolores Perin, Julia Raufman, and Hoori Santikian Kalamkarian. Developmental reading and English assessment in a researcher-practitioner partnership. Technical report, CCRC, Teachers College, Columbia University, 2015.

36 Dylan Phelps, Thomas Pickard, Maggie Mi, Edward Gow-Smith, and Aline Villavicencio. Sign of the times: Evaluating the use of large language models for idiomaticity detection. In Archna Bhatia, Gosse Bouma, A. Seza Doğruöz, Kilian Evang, Marcos Garcia, Voula Giouli, Lifeng Han, Joakim Nivre, and Alexandre Rademaker, editors, *Proceedings of the Joint Workshop on Multiword Expressions and Universal Dependencies (MWE-UD) @ LREC-COLING 2024*, pages 178–187, Torino, Italia, May 2024. ELRA and ICCL. URL: https://aclanthology.org/2024.mwe-1.22/.

37 Sundai Pichai and Demis Hassabis. Introducing Gemini: our largest and most capable AI model, 2023. Accessed: February 19, 2025. URL: https://blog.google/technology/ai/google-gemini-ai/#introducing-gemini.

38 Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

39 Areeg Fahad Rasheed, M Zarkoosh, Safa F Abbas, and Sana Sabah Al-Azzawi. TaskComplexity: A dataset for task complexity classification with in-context learning, FLAN-T5 and GPT-4o benchmarks. *arXiv preprint arXiv:2409.20189*, 2024. doi:10.48550/arXiv.2409.20189.

40 Eugénio Ribeiro, Nuno Mamede, and Jorge Baptista. Exploring the automated scoring of narrative essays in brazilian portuguese using transformer models. In *Proceedings of the 16th International Conference on Computational Processing of Portuguese-Vol. 2*, pages 14–17, 2024. URL: https://aclanthology.org/2024.propor-2.4/.

41 Siti Bealinda Qinthara Rony, Tan Xin Fei, and Sasa Arsovski. Educational justice. reliability and consistency of Large Language Models for automated essay scoring and its implications. *Journal of Applied Learning and Teaching*, 8(1), 2025.

42 Kathrin Seßler, Maurice Fürstenberg, Babette Bühler, and Enkelejda Kasneci. Can ai grade your essays? a comparative analysis of large language models and teacher ratings in multidimensional essay scoring. In *Proceedings of the 15th International Learning Analytics and Knowledge Conference*, pages 462–472, 2025. doi:10.1145/3706468.3706527.

43 Arthur Spirling. Why open-source Generative AI models are an ethical way forward for science. *Nature*, 616(7957):413–413, 2023.

44 Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

45 LearnLM Team, Abhinit Modi, Aditya Srikanth Veerubhotla, Aliya Rysbek, Andrea Huber, Brett Wiltshire, Brian Veprek, Daniel Gillick, Daniel Kasenberg, Derek Ahmed, et al. Learnlm: Improving Gemini for learning. *arXiv preprint arXiv:2412.16429*, 2024.

46 Yuqi Wang, Wei Wang, Qi Chen, Kaizhu Huang, Anh Nguyen, and Suparna De. Zero-shot text classification with knowledge resources under label-fully-unseen setting. *Neurocomputing*, 610:128580, 2024. doi:10.1016/j.neucom.2024.128580.

47 Zhiqiang Wang, Yiran Pang, Yanbin Lin, and Xingquan Zhu. Adaptable and reliable text classification using Large Language Models. *arXiv preprint arXiv:2405.10523*, 2024.

48 Changrong Xiao, Wenxing Ma, Qingping Song, Sean Xin Xu, Kunpeng Zhang, Yufang Wang, and Qi Fu. Human-AI collaborative essay scoring: A dual-process framework with LLMs. In *Proceedings of the 15th International Learning Analytics and Knowledge Conference*, pages 293–305, 2025. doi:10.1145/3706468.3706507.

A Accuplacer and LLMs' sample feedback

Test Results

Test Name	Score
[redacted]	2

Holistic Score Description
The essay demonstrates very little mastery of on-demand essay writing.

Dimension Title	Dimension Description
Purpose and Focus	Your response lacks clear purpose and focus.
Organization and Structure	Your response demonstrates limited organization of ideas.
Development and Support	Your response has limited support for your ideas.
Sentence Variety and Style	Your response shows inconsistent control of sentence variety, word choice, and flow of thought.
Mechanical Conventions	Your response shows limited control of mechanical conventions such as grammar, spelling, and punctuation.
Critical Thinking	Your response shows insufficient reasoning and lacks complexity of thought.

Course Placements - Determined on 07-20-2023

Writing Foundations I

Figure 3 Accuplacer's feedback for DevEd Level 1 = Writing Foundations I.

Gemini Enhanced+'s Feedback. Text: A 322; Classification: DevEd Level 1 Rationale:

> The text exhibits frequent errors in mechanical conventions, including misspellings ("themselfs," "simpley," "mojority," "to," "garentee," "sucurty," "revanew," "Anthoner," "belive," "unles"), punctuation (incorrect comma usage, missing apostrophes), and grammatical errors (subject-verb agreement, pronoun usage). Sentence variety and style are limited, with repetitive sentence structures. Development and support are present but lack coherence and depth. Organization and structure are attempted but weakened by abrupt transitions and lack of a clear thesis in the introduction. While the purpose and focus are identifiable, the arguments lack sufficient clarity. The text exhibits limited critical thinking, relying heavily on anecdotal evidence and generalizations.

A DSL for Swarm Intelligence Algorithms

Kevin Martins
ALGORITMI Research Centre, University of Minho, Braga, Portugal

Rui Mendes
LASI Research Centre, University of Minho, Braga, Portugal

Abstract

We propose a domain-specific language to simplify the expression of Swarm Intelligence algorithms. These algorithms are typically introduced through metaphors, requiring practitioners to manually translate them into low-level implementations.This process can obscure intent and hinder reproducibility. The proposed DSL bridges this gap by capturing algorithmic behavior at a higher level of abstraction. We demonstrate its expressiveness in a few lines of code and evaluate its feasibility through a reference implementation. A discussion is presented that includes empirical comparisons with traditional implementations and future directions of the proposed DSL.

2012 ACM Subject Classification Software and its engineering → Domain specific languages

Keywords and phrases Domain Specific Languages, Swarm Intelligence, Global Optimization

Digital Object Identifier 10.4230/OASIcs.SLATE.2025.2

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Unit Project Scope UID/00319/Centro ALGORITMI (ALGORITMI/UM).
Kevin Martins: Kevin Martins thanks FCT for the grant SFRH/BD/151434/2021.

1 Introduction

Optimization is a popular research field that aims to find the best solution to a given problem by minimizing or maximizing an objective. It is a fundamental aspect of many disciplines, including mathematics, computer science, and engineering [11].

In the presence of gradient-based information, optimization problems can be solved efficiently. However, in many cases, this information is not available or is difficult to obtain. In these scenarios, metaheuristics represent a popular alternative as they are a black-box method for finding near-optimal solutions.

These black-box methods have become popular tools in many fields such as digital image processing, computer vision, networks and communications, power and energy management, machine learning, robotics, medical diagnosis, and others [7].

A particular category of metaheuristics that has attracted much interest in the last two decades is Swarm Intelligence (SI) [26]. SI algorithms are population-based metaheuristics that take inspiration from nature by mimicking the intelligent behavior that emerges from a population of simple organisms in self-organized systems.

In recent years, due to the considerable number of these types of algorithms in the literature, the research community has expressed concerns about the field [34, 31, 33]. The dependence on metaphors as the main motivation behind the inner workings of algorithms can hide similarities between them, leading to duplication of research effort. Furthermore, there is a tendency to reimplement these algorithms from scratch, which hinders reproducibility and replicability [34].

However, the no-free lunch (NFL) theorem [39] suggests that the main problem in this field is that these algorithms are not necessarily good at solving all kinds of problems. In fact, these algorithms often need to be configured and tested to discover which algorithm is most suitable to discover the best solution to a given optimization problem.

This paper proposes a domain-specific language (DSL) for expressing SI algorithms. The goal is to demonstrate the feasibility and practicality of using a purpose-built DSL as a unified and intuitive framework for researchers and practitioners to develop, analyze, combine, and systematically compare SI algorithms.

We show that the proposed DSL captures the essence of SI algorithms at a higher level of abstraction, independent of any specific programming language. Most importantly, it provides a clear separation between metaphor and core algorithmic concepts, such as recombination, probability distributions, and interactions between individuals. This structured representation improves the ability to identify algorithmic similarities, assess novelty, and support reproducibility.

Since optimization is a broad topic, this study focuses on SI algorithms applied to continuous single-objective optimization in real domains.

2 Background

The use of metaphors blurs the specific mechanism in the solution domain, making it difficult to determine the novelty of the contribution of an algorithm [34, 31]. For example, it has been argued that the Harmony Search algorithm can be formulated as a variation of Evolution Strategies [33], the Cuckoo search algorithm as well [5], and the Firefly Algorithm is argued to be similar to Particle Swarm Optimization with a grouping mechanism inspired by the behavior of fireflies [18]. It was also argued that for some algorithms, the metaphor, the mathematical model, and the algorithm are almost entirely different [5, 2].

This overlay between algorithms suggests that metaheuristics have common components that can be combined or that specific parts of these algorithms can be changed to create new variations. Due to the NFL theorem, we argue that this fact is a feature, not a bug, since algorithms can be fine-tuned or modified for a specific optimization problem. In recent years, several approaches have been proposed to automatize the creation and tuning of metaheuristics, which seems to support this fact [23, 4, 10, 20].

Furthermore, the research agenda proposed in [33] advocates that a pure functional description is vital to standardize metaheuristics, opening up opportunities to compose heuristics in novel ways. Although in a more narrow scope, this approach was taken in [29].

Functional programming (FP) is characterized by: (1) the use of pure functions, that is, given the same input, the same output is always produced, and (2) by having no side effects, that is, functions only compute the output and make no changes outside its scope.

Higher-order functions and lazy evaluation are two powerful features with significant contributions to the modularity of functional programming, since they allow modules/functions to be composed, acting like a kind of 'glue' [14]. Therefore, this approach seems well suited for gluing parts of SI algorithms.

FP and DSL are natural partners since both emphasize expressiveness and modularity [13]. Instead of focusing on computing, DSL makes programming in a domain more efficient since expressiveness is bounded by a purpose-built language [12]. Therefore, DSL allows computing-specific improvements and techniques to be detached from the domain, allowing scientific advances in computing and the domain to be seamlessly adopted without disrupting existing code or workflows. This perspective aligns with the proposed DSL; that is, using functional programming principles to model SI algorithms modularity allows domain and computational advances to evolve independently.

DSL can be internal or external [12]. While external DSL is parsed independently of the underlying programming language, internal DSL represents an Application Programming Interface (API) for the underlying programming language. This study focuses on external

DSL, however, the scope of the objective function of the proposed in the DSL is limited. That is, certain specificities of objective functions such as access and manipulation of external files or databases are not addressed in this proposal.

Libraries such as Opytimizer [6], NiaPy [38], and MEALPy [37] offer prebuilt algorithm templates and utilities for benchmarking, but remain grounded in imperative, metaphor-centric programming paradigms. As a result, algorithm implementations often involve boilerplate code and tight coupling between search strategies and problem-specific logic.

In contrast, the proposed purpose-built DSL adopts a high-level declarative approach that abstracts away low-level implementation details. By emphasizing mechanism-first design over metaphor-driven structures, it enables practitioners to express, compose, and hybridize algorithmic components more concisely and flexibly. In addition, it supports contextualized declarative integration of auxiliary features such as parameter tuning, making the language naturally extensible.

The following sections will describe the components of the SI algorithms that were identified and the proposed DSL.

3 Components of Swarm Intelligence algorithms

Many real-world problems can be rewritten as the minimization or maximization of some objective. As stated in [35], these problems can be defined by the couple (S, f), where S is the set of feasible solutions and $f\colon S \to \mathbb{R}$ is the objective function to optimize. f maps every feasible solution $s \in S$ to a real number representing the quality of the solution.

At a higher level, every SI algorithm comprises two main components: the search space and the search strategy. The search space is represented by f. On the other hand, the search strategy is responsible for finding the optimal solution s^* by iteratively generating new solutions s that are submitted for evaluation to the search space that, in turn, responds with the respective quality mapped.

Individuals explore/exploit the search space by generating new solutions. Depending on the number of individuals involved in the search process, metaheuristics can be classified into two types: single-solution, where only one individual explores and exploits the search space, and population-based, where multiple individuals search for the best solution in parallel. SI algorithms adopts a population-based approach.

Table 1 Algorithms included in this study.

Algorithm	Reference
Artificial Bee Colony (ABC)	[15]
Cuckoo Search (CS)	[43]
Differential Evolution (DE)	[32]
Particle Swarm Optimization (PSO)	[17]
Barebones Particle Swarm (BB)	[16]
Fully Informed Particle Swarm (FIPS)	[22]
Firefly Algorithm (FF)	[41]
Jaya	[27]
Sine Cosine Algorithm (SCA)	[24]
Grey Wolf Optimizer (GWO)	[25]
Teaching Learning based Optimization (TLBO)	[28]

The purpose of the search strategy is to use the quality of the solutions generated by the individuals as a guide to find the optimal solution s^*. The search process begins with a set of initial solutions in which every individual generates and holds a solution. Then, the individuals engage in an iterative process, using information about the population to generate new solutions. In some algorithms, individuals maintain the best solution, thus maintaining the current solution if it is better than the candidate.

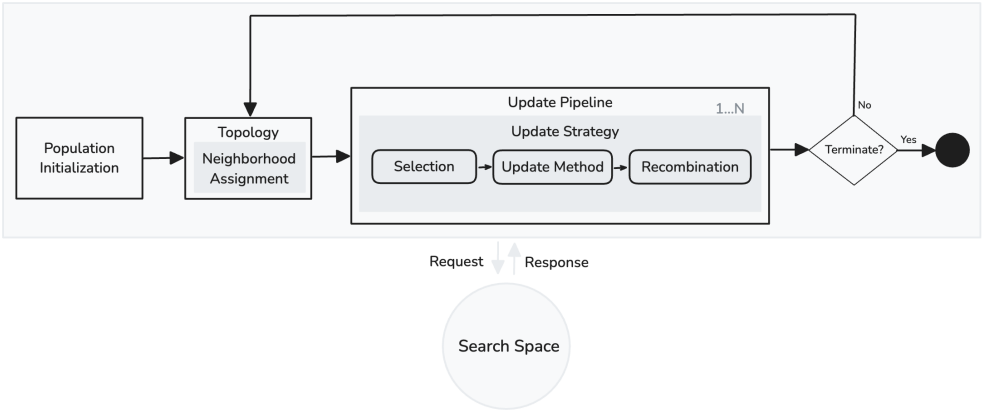


Figure 1 Components of SI algorithms.

As illustrated in Fig. 1, the search strategy can be divided into three main components: population initialization, topology, and update pipeline. The search process will stop once the termination criterion has been satisfied. This study treats these components as functions with specific input and output constraints that must be satisfied. Thus, a valid component implementation can be any function that satisfies its constraints.

With this approach, the search process can be generalized so that building blocks can be swapped in and out to compose new or hybrid strategies.The following sections detail the inner workings of these components. The described building blocks were identified after analyzing the metaheuristics shown in Table 1.

FIPS and BB are variants of PSO, but are included in this study due to their distinct internal mechanisms. FIPS relies on a neighborhood topology in which each particle is influenced by all its neighbors, rather than just the global or local best. BB, on the other hand, eliminates velocity updates and samples new positions using a Gaussian distribution - representing an implicit model that contrasts with the explicit mathematical formulations typical of most SI algorithms.

In addition, although DE is not traditionally classified as a swarm intelligence algorithm, it is a population-based metaheuristic that shares similarities with SI algorithms such as stochasticity and information sharing. It was included in this study because of its distinctive use of crossover mechanisms.

Although this study does not include the latest LSHADE variants, recognized in [19] for their strong empirical performance, the identified components are general enough to model such algorithms. In particular, adaptive population size and historical memory can be expressed as modular functions or higher-order constructs within the proposed framework, preserving its applicability to advanced strategies.

3.1 Population initialization

The search process begins with the initialization of a population of individuals. Initializing the population with a diverse and representative set of solutions that effectively cover the solution space is recommended.

Population initialization generates an initial set of candidate solutions. This can be formalized as a function $f\colon n \to S_0$, where $n \in \mathbb{Z}_{>1}$ is the size of the population, and $S_0 = \{s_1, s_2, \ldots, s_n\}$ is the resulting set of solutions.

Usually, random number generation methods are used to initialize the population. Random number generation can be generated from any probability distribution depending on the problem.

3.2 Topology

Individuals in the population can be organized so that it is possible to manipulate the dissemination of information throughout the population [3]. Individuals can be grouped into neighborhoods, limiting their access to information to the assigned neighborhood. Although this method is popular in the PSO community, it can be generalized to other population-based metaheuristics.

A topology function can be any function $f\colon P \to T$ that, given the population of individuals P as input, produces as output the set of neighborhood $T = \{N_1, N_2, ..., N_n\}$ where $N_k = \{1, 2, ..., n\}$ represents the set of individuals of a given neighborhood.

Usually, all individuals in the population have information about the entire population. This topology is termed globally oriented (GBest). In contrast, locally oriented (LBest) is defined when the adjacent population members are arranged in neighborhoods [21, 8].

Only LBest and GBest topologies were employed in this study. However, since the topology function is triggered in every iteration i, our approach remains valid even when the topology is dynamic, that is, the organization changes over the iterations.

3.3 Update pipeline

In every iteration i, selected individuals update their solution by generating new candidate solutions based on the information provided by their neighborhood. Usually, one update per individual takes place in every generation. Others, like TLBO and ABC, perform multiple updates based on some selection mechanism. Thus, a pipeline where one or more updates occur can be generalized, i.e., in every iteration, a set of updates $U = \{u_1, u_2, ..., u_n\}$ occurs sequentially.

Every update method u_k in the pipeline can be defined by the couple (f, g), where f is the selection function and g is the update function, that is, only selected individuals can generate new candidate solutions.

The selection process can be generalized by a function $f\colon P \to F$ such that, given the population P, it returns the selected individuals $F \subseteq P$. Examples of selection procedures include:

- **All**: All individuals are selected.
- **Random**: k individuals are randomly selected.
- **Probabilistic**: k individuals are selected based on some fixed probability or proportionally to their fitness, that is, better/worst-fitted individuals have higher chances of being selected.
- **Rule-based**: The selection of an individual is governed by a set of rules.

Then, the selected individuals engage in the process of updating their position (solution). The update process can be generalized by a higher order function $g\colon e \times r \rightarrow s_i$ where $s_i = (s_1, s_2, ..., s_d)$ is the generated solution vector with d dimensions, e represents the function responsible for the calculation of candidate solutions and r the recombination method.

To generate the solution vector, equations are applied using information about the search space collected by the neighborhood, for example, the best and worst solutions. The equations usually include random perturbations to promote exploration and exploitation of the search space.

The generated solution is then recombined using some specific method r, that is, applying the generated solution to $k \leq n$ dimensions of the current solution of the individual s_{i-1}. This process is referred to as recombination. Usually, the generated solution is applied to all dimensions. Other methods include:

- **Binomial**: Based on a predefined probability, the dimensions will be randomly selected and updated with the new solution.
- **Exponential**: Starting with a randomly chosen index of the circularly arranged set of dimensions, adjacent indices are selected and updated with the new solution until we fail a probability roll [9].
- **Random**: A fixed number of dimensions are updated randomly.
- **Probabilistic**: An approach similar to the binomial method. However, a maximum of k dimensions are updated.

The candidate solution generated s_i from the update method is then evaluated. Some algorithms are elitist, discarding the candidate solution if it is not better than the individual's best-known solution.

From the algorithms studied, a function evaluation is performed for every update, with FF representing the only gray area. In the proposal paper pseudocode [41], FF has an iterative process in that, in each generation, an individual calculates a new solution and performs function evaluations for every other individual with a better fitness value. However, in the MATLAB code [42], the author suggests that the update method is applied iteratively whenever the individual finds another one with better fitness, and at the end only one evaluation of the objective function evaluation is made.

In order to make our proposal more easily adaptable in parallel computing scenarios, this study assumes that each update method only undergoes one objective function evaluation per individual. This approach allows for the decoupling of the objective function evaluation from the update method, that is, the user provides the update pipeline, and internally the system takes care of the solution evaluation task after each update. Thus, parallel execution of the evaluation task and the update method can be enabled among individuals for every iteration i.

4 Swarm Domain Language

The components identified in the previous chapter can be leveraged to construct a DSL for SI algorithms, which we refer to as Swarm Domain Language (SDL).

Parameter settings are crucial for the performance of an algorithm when searching for solutions to optimization problems. Although purposeful configurations can produce consistent results for a specific problem, they do not guarantee success for all problems. This makes selecting the appropriate parameters a challenging task. Hence, the parameter-tuning feature is also included in this DSL proposal.

Listing 1 High-level overview of the language.

```
SEARCH ( { search_space } )
USING ( { search_strategy } )
UNTIL ( { termination_criteria } )
```

Listing 1 presents a high-level overview of SDL. For better understanding, the following preliminary syntax is introduced:

- `::=` is used to express how placeholders, e.g., `search_space`, are syntactically defined.
- `{...}` is used to define the required components.
- `[...]` is used to define optional components.
- `|` is used to define the list of alternative expressions that can be used.
- `+` indicates that a specific expression can be repeated, that is, one or more if used in conjunction with `{...}` and zero or more if used in conjunction with `[...]`.
- Numbers $\in \mathbb{R}$ are referred as `number`.
- Numbers $\in \mathbb{Z}$ are referred as `integer`.
- `equation` is used whenever a mathematical equation is expected. Arithmetic and trigonometric operations are allowed.
- `expression` is used whenever is expected a number, parameter, `equation` or other production rules that will be described in this section.
- `comparison` is used when comparing two values to determine their relationship: equality (`=`), inequality (`!=`), greater/less than (`>`, `<`), or greater/less than or equal to (`>=`, `<=`).

In the following sections, we will focus on the language definition of the search space, the search strategy, and the termination criteria.

4.1 Search space statement

The search space statement is shown in Listing 2. It begins with the declaration of one or more variables. A variable can be a vector; in this case, the size should be specified. The lower and upper bounds of each variable are specified using real numbers.

Listing 2 Search space statement of the language.

```
search_space ::= {VAR {variable_name} [SIZE({integer})]
    [BOUNDED BY ({lower_bound},{upper_bound})]}+
{MINIMIZE | MAXIMIZE} {equation}
```

When specifying the objective function, the goal of the optimization should be declared. That is, `MINIMIZE` or `MAXIMIZE` should be used to specify the optimization goal.

4.2 Search strategy statement

The search strategy statement is shown in Listing 3. First, the parameter declaration occurs and is optional since some algorithms do not require parameters. Parameters can be set statically, using numbers or integers, dynamically set using expressions, or, in addition, automatically set using `AUTO`. Then, the process described in Section 3 follows.

To perform parameter tuning, replace the static parameter expression with the automatic one as shown in Listing 4. Integer and real numbers are supported to define the boundaries of the search space where parameter tuning should occur. It is worth pointing out that when automatic parameters are defined, the statement `TUNE UNTIL(...)` must be specified, and the maximum number of generation (trials) should be provided, i.e. how many evaluations of the SI algorithm are allowed during parameter tuning.

Listing 3 Search strategy statement of the language.

```
search_strategy ::= [PARAM = { expression | auto }]+
{POPULATION SIZE(integer)} INIT {initialization}
[WITH TOPOLOGY {GBEST | {LBEST [SIZE(integer)}]}] {
    [WITH selector]
    SELECT {ALL | SIZE(integer)}}
    [WHERE comparison]
    [ORDER BY {TRIALS | FIT}]
    [USING {
        [BINOMIAL | EXPONENTIAL] RECOMBINATION
        WITH PROBABILITY {probability}
        | RANDOM RECOMBINATION SIZE(integer)}
    }]
    {UPDATE (
        [helper = expression]+
        {POS = expression}
    ) [WHEN comparison]}

}+ [TUNE UNTIL(GENERATION=integer)]
```

Listing 4 Parameter tunning statement of the language.

```
auto ::= AUTO {INT|FLOAT} BOUNDED BY ({number|integer},{number|integer})
```

The population initialization instruction follows the parameter declaration. The size expression defines the number of individuals of the population, while `initialization` instructs which method to use. Listing 5 shows the currently supported methods.

Listing 5 Initialization methods of the language.

```
initialization ::= RANDOM()
 | RANDOM_UNIFORM([LOW={number}[,HIGH={number}]])
 | RANDOM_[LOG]NORMAL([LOC={number} [,SCALE={number}]])
 | RANDOM_SKEWNORMAL([SHAPE={number} [,LOC={number} [,SCALE={number}]]])
 | RANDOM_{CAUCHY | LEVY}([LOC={number} [,SCALE={number}]])
 | RANDOM_{EXPONENTIAL | RAYLEIGH}([SCALE={number}])
 | RANDOM_BETA([ALPHA={number} [,BETA={number}]])
 | RANDOM_WEIBULL([SHAPE={number} [,SCALE={number}]])
```

The topology of the population is optional. When specifying a Lbest topology, the default size of the neighborhoods is 2 when not provided by the user.

Before moving further, let us introduce the individual memory. It contains a set of properties that are managed internally throughout generations. It is made up of the following properties:

- `POS`: The current position of the individual.
- `BEST`: The best position the individual found.
- `DELTA`: The size of the last step taken by the individual, that is, the difference in position between the current and the previous position.
- `FIT`: The current fitness of the individual.
- `IMPROVED`: If the current position represents an improvement in fitness compared to the previous one.

- `TRIALS`: The number of trials that an individual has performed without improving its fitness.
- `NDIMS`: The number of dimensions that compose the position vector.
- `INDEX`: The index of the individual in the population.

For the update pipeline, the selection, update, and recombination methods are supported. In the selection expression, the selectors in Listing 6 can be optionally specified. If specific selectors are provided, `WHERE` and `ORDER BY` must be omitted; otherwise, these clauses can be used with individual properties in the comparison expression.

Listing 6 Selector statement of the language.

```
selector ::= [ ROULETTE | RANDOM | PROBABILITY {probability} ]
```

Since in some algorithms, the update method can have several instructions, helper variables can be specified as needed. For example, in PSO, velocity can be defined as a helper variable to aid in position calculation. The position of the individual, `POS`, is always required in the update method, as it defines the new position of the individual in the search space. However, with the statement `WHEN`, specific conditions can be defined to update the individual. For example, to update the individual position only if the new position is better than the previous position.

Listing 7 Snippet of expression statement in the language. For simplicity arithmetic and trigonometric operations where omitted.

```
expression ::=  reference|aggregation|utility_function|perturbation|...
reference ::= ALL() | NEIGHBORHOOD() | SWARM_{BEST | WORST}([integer])]
| PICK_{RANDOM | ROULETTE}([UNIQUE] [integer] [WITH REPLACEMENT])
|{RAND|CURRENT}_TO_BEST(WITH PROBABILITY {probability})
aggregation ::= {SUM | AVG | MIN | MAX}({expression})
utility_function ::= IF_THEN({comparison}, {expression}, {expression})
| FILTER({expression}, ({key}) => {expression})
| MAP(expression, ({key}) => {expression})
| REDUCE({expression}, ({key}, {acc}) => expression)
| REPEAT({expression}, {expression})
| DISTANCE({expression}, {expression})
perturbation ::= RANDOM([SIZE={integer}])
| RANDOM_UNIFORM([LOW={expression}[,HIGH={expression}[,SIZE={integer}]]])
| RANDOM_[LOG]NORMAL([LOC={expression} [,SCALE={expression}
    [,SIZE={integer}]]])
| RANDOM_SKEWNORMAL([SHAPE={expression}
    [,LOC={expression} [,SCALE={expression}[,SIZE={integer}]]]])
| RANDOM_{CAUCHY|LEVY}([LOC={expression} [,SCALE={expression}
    [,SIZE={integer}]]])
| RANDOM_{EXPONENTIAL|RAYLEIGH}([SCALE={expression}[,SIZE={integer}]])
| RANDOM_BETA([ALPHA={expression} [,BETA={expression}[,SIZE={integer}]]])
| RANDOM_WEIBULL([SHAPE={expression} [,SCALE={expression}
    [,SIZE={integer}]]])
```

By default, search space helper variables are also available allowing the use of context-specific search space variables. `MAX\_GEN` and `CURR\_GEN` return the maximum and current generation, respectively. The same logic is also applied to `MAX\_EVALS` and `CURR\_EVAL`.

Both `helper` and `POS` are defined using `expression` where mathematical operations can be performed using information derived from the population. In addition, the constructs `reference`, `aggregation`, `utility_function`, and `perturbation` are allowed as shown in Listing 7.

In SI algorithms, the common requirement for calculating a candidate's position is to use the information provided by the population or neighborhood. Thus, `reference` can be used for this purpose. Here, when `SWARM_BEST` or `SWARM_WORST` is used without specifying the size, the best or worst individual in the neighborhood is returned, respectively. Otherwise, the k best/worst will be returned. In addition, the `UNIQUE` clause ensures that individuals are selected without duplication on the update method. Random and roulette selection can be done with or without replacement.

Although in this proposal `SUM`, `AVG`, `MIN`, and `MAX` are the aggregation operators proposed, additional ones could be added. However, utility functions `MAP` and `REDUCE` are supported and can be used to specify more complex aggregation operations.

In fact, complex logic is sometimes required in SI algorithms. `FILTER`, `MAP`, and `REDUCE` provide a way to execute operations on an expression that evaluates into a collection. `FILTER` constructs a new collection by including only the elements of the original collection for which a specified `expression` evaluates to `true`. `MAP` transforms each collection element using `expression`, producing a new collection of transformed values. `REDUCE` computes a single result for the collection elements using the `acc` parameter as the aggregator of results.

The individual and its properties can be accessed in the `key` parameter. For example, for an operation performed on a collection of individuals, if `individual` is defined as the name of the parameter `key`, the expression `individual.fit` gives the fitness of the individual being evaluated.

`DISTANCE` calculates the euclidean distance between two `expressions`, for example, the positions of two individuals. `REPEAT` returns a list of the first `expression` parameter repeated the number of times specified by the second `expression`.

Finally, perturbations can be used to define the new position of the individual. These are the identical probabilistic distributions allowed in the initialization step. However, here, the size of the random number generation can be specified while in initialization its by definition given by the number of dimensions of the search space.

4.3 Termination criteria statement

Listing 8 shows the termination criteria statement. `EVALUATIONS` and `GENERATION` define the maximum allowed number of evaluations and generations, respectively. `FITNESS` defines the minimum or maximum fitness value that must be achieved before termination. If more than one condition is specified, the termination will occur whenever one of them is reached.

Listing 8 Termination criteria statement of the language.

```
termination criteria ::= {
    { {EVALUATIONS|GENERATION}  = {integer} }
    | { FITNESS = {number} }
}+
```

5 Discussion

This section discusses the practical use and evaluation of the proposed DSL.

We first demonstrate its expressiveness through example implementations of SI algorithms using SDL, then present comparative results with traditional approaches. Finally, we highlight the opportunities for extensibility and future directions identified during this study.

5.1 SDL examples

To illustrate the expressiveness and practical utility of the proposed DSL, we present implementations of GWO and BB using SDL. Both algorithms are applied to the Sphere function, a well-known benchmark in continuous optimization.

The Sphere function is defined by:

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2 \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^d$ and $-5.12 \leq x_i \leq 5.12$ for all $i = 1, \ldots, d$. It is a simple, convex, and continuous function with a single global minimum of $f(\mathbf{x}^*) = 0$, at $\mathbf{x}^* = 0, \ldots, 0$.

As shown in Listings 9 and 10, with 13 and 17 lines of SDL for BB and GWO respectively, it was possible to express not only the algorithms but also the objective function, showing the prototyping capabilities of such DSL.

Due to space constraints, the implementation of the remaining algorithms used in this study is available at: `https://github.com/kafm/swarmist/tree/main/examples`. These examples illustrate the expressiveness and practicality of the proposed SDL. Each algorithm can be implemented in a concise and readable manner, with high-level constructs that clearly capture the intended behavior demonstrating the language's effectiveness in reducing boilerplate and improving clarity.

Another key aspect of such approach is that we can create hybrid algorithms, but using building blocks from other algorithms. For instance, we can apply the binomial crossover from DE to BB as shown in Listing 11.

Listing 9 SDL of the Grey wolf optimizer (GWO).

```
SEARCH(
    VAR X SIZE(20) BOUNDED BY (-5.12, 5.12)
    MINIMIZE SUM(X ** 2)
) USING (
    PARAM A = 2
    POPULATION SIZE(40) INIT RANDOM_UNIFORM()
    SELECT ALL (
        UPDATE (
            A = PARAM(A) - CURR_GEN * ( PARAM(A) / MAX_GEN )
            POS = AVG(
                MAP(SWARM_BEST(3), (REF) =>
                    A * ABS( (2 * RANDOM()) * REF.POS - POS)
                )
            )
        ) WHEN IMPROVED = TRUE
    )
) UNTIL (GENERATION = 1000)
```

Listing 10 SDL of the Bare Bones Particle Swarm (BB).

```
SEARCH(
    VAR X SIZE(20) BOUNDED BY (-5.12, 5.12)
    MINIMIZE SUM(X ** 2)
) USING (
    POPULATION SIZE(40) INIT RANDOM_UNIFORM()
    SELECT ALL (
        UPDATE (
            MU= (SWARM_BEST()+BEST)/2
            SD = ABS(SWARM_BEST()-BEST)
            POS = RANDOM_NORMAL(LOC=MU, SCALE=SD)
        ) WHEN IMPROVED = TRUE
    )
) UNTIL (GENERATION = 1000)
```

Listing 11 SDL snippet demonstrating the application of crossover in BB algorithm.

```
PARAM  CR = 0.6
POPULATION SIZE(40) INIT RANDOM_UNIFORM()
SELECT ALL (
    USING BINOMIAL RECOMBINATION WITH PROBABILITY PARAM(CR)
    UPDATE (
        MU= (SWARM_BEST()+BEST)/2
        SD = ABS(SWARM_BEST()-BEST)
        POS = RANDOM_NORMAL(LOC=MU, SCALE=SD)
    ) WHEN IMPROVED = TRUE
)
```

5.2 Experimental Evaluation

To demonstrate the feasibility of the proposal DSL, an example implementation was developed and is available on GitHub: `https://github.com/kafm/swarmist`. Although built using Python, the same principles can be used to port the DSL to other programming languages.

Lark [30] was used as the parsing library. A complete machine-readable grammar for SDL in EBNF format can be found at `https://github.com/kafm/swarmist/blob/main/swarmist/sdl/grammar.py`, which defines every nonterminal, token, and production rule needed for parsing, thus satisfying the requirement for a formal and unambiguous DSL specification.

For parameter tuning, the example implementation used Optuna [1], however, different implementations can use different frameworks or methods.

This implementation was then used to perform the following experiments: 1) comparative results with traditional implementations; and 2) parameter tuning performance evaluation.

We used benchmark functions of the CEC-2017 competition (F1 to F20) [40] with 30 dimensions for all experiments. Opfunu [36], a Python library that implements these benchmark functions, was used. In addition, we set the termination criteria to 2500 epochs and the population size was set to 40 individuals for all algorithms.

5.2.1 Comparative results with traditional implementations

To ensure that the SDL results are consistent with traditional metaheuristic implementations, we compared the SDL results with the Mealpy results, an open source Python library with more than 160 algorithms implemented [37]. For simplicity, we only selected the SI algorithms in our study with single update pipelines, that is, PSO, DE, JAYA, FF, GWO, SCA, and WO.

The same parameters were used for SDL and Mealpy. Regarding PSO, a different variation is implemented in Mealpy since the function receives a minimum and maximum inertia weight as parameters. Thus, we set the minimum at 0.45 and the maximum at 0.73.

We independently ran the Mealpy implementation for each algorithm and their static settings 30 times each and applied the Wilcoxon rank-sum test with $\alpha = 0.05$ to check whether there was a statistically significant difference between the Mealpy-implemented algorithms and the SDL ones. Holm corrections were applied to the p-values because several tests were performed.

We could not statistically conclude that the two settings differ in performance for DE, SCA, and WO (`p-value` > 0.05). SDL implementations of PSO, JAYA, and GWO were significantly better than Mealpy (`p-value` < 0.05). In contrast, for FF, the Mealpy implementation yielded better results (`p-value` < 0.05). The differences found in performance may be related to the specific implementations of the algorithms, for example, the variation of PSO implemented by Mealpy.

Regarding FF, SDL was significantly worse in all problems. By analyzing the Mealpy implementation, we found that a function evaluation was performed wherever an individual found a better solution for every generation, that is, the Mealpy implementation performed many more function evaluations than SDL. Thus, the experiment could have been more fair.

5.2.2 Parameter setting performance evaluation

To assess the effectiveness of parameter tuning capabilities, we compared the results of SDL with tuned parameters with those found in the SDL vs. Mealpy experiment. We excluded FF from this experiment because of the objective function evaluation problem already explained.

We independently performed the SDL parameter settings tuning for each algorithm, running it 30 times. To check for any significant differences between the SDL with parameter tuning, the static SDL settings, and Mealpy, we applied the Wilcoxon rank-sum tests with $\alpha = 0.05$. Since we were performing multiple tests, we used the Holm correction for the p-values. As expected, SDL with parameter settings tuned was statistically better than static settings and Mealpy in all benchmark problems (`p-value` < 0.05).

5.3 Extensibility and Future Directions

The SDL proposed in this study leverages high-level constructs to abstract common SI mechanisms. This abstraction facilitates implementation by reducing the need for imperative code and enabling modular composition of algorithms. In practice, this led to more concise, readable, and reusable definitions.

Although the current work focuses on the definition and execution of algorithms, further extensibility could enhance the long-term utility of the language. For example, the ability to persist and reuse complete algorithm definitions would allow practitioners to encapsulate algorithmic logic once and reuse it across different problems. Listing 12 illustrates a hypothetical SDL syntax for persisting an algorithm, while Listing 13 shows how such an algorithm could be reused.

Listing 12 Snippet of hypothetical SDL for persisting an algorithm.

```
CREATE OR REPLACE ALGORITHM BB(
    POPULATION SIZE DEFAULT 40, INIT DEFAULT RANDOM_UNIFORM) AS ...
```

Listing 13 Snippet of hypothetical SDL for reuse of a previously persisted algorithm.

```
SEARCH(
    VAR X SIZE(20) BOUNDED BY (-5.12, 5.12)
    MINIMIZE SUM(X ** 2)
) USING BB()
```

A similar mechanism could also apply to lower-level abstractions, such as reusable expressions for parameter adaptation or recombination strategies. This would further promote the construction of hybrid algorithms by mixing building blocks at different abstraction levels, ultimately improving the language's expressiveness and extensibility.

Based on these insights, we hypothesize that SDL could support the emergence of an evolving, reusable library of SI algorithm components. This collection, combined with automatic tuning and composition mechanisms, could form the basis for automated algorithm design tailored to specific problem classes.

Although these capabilities were not addressed in this study, they represent promising directions for future work. However, their implementation involves complex design challenges that will require further investigation.

Learning a new language may not be ideal for practitioners, and one may argue that visual interfaces may be preferred over DSLs. However, the textual nature of SDL offers precise version-controlled specifications that improve systematic design and auditing of optimization models.

6 Conclusion

This study introduced a purpose-built DSL designed to support the development and experimentation of SI algorithms. The DSL provides a dedicated and expressive framework that frees researchers from the constraints of metaphor-driven representations, enabling them to model search strategies, set parameters, and combine algorithmic components with greater clarity and flexibility.

The proposed DSL offers a high-level declarative framework for the development and experimentation of SI algorithms by abstracting low-level implementation details and focusing on reusable mechanisms rather than metaphors. Thus, it enables clear, flexible, and concise representations of various algorithmic strategies.

Through illustrative examples and empirical evaluation, we showed that the DSL maintains performance comparable to traditional implementations, while significantly improving clarity, modularity, and reproducibility.

In general, the DSL establishes a foundation for a more systematic design and analysis of SI algorithms, supporting both rigorous experimentation and scalable reuse of algorithmic components.

References

1 Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

2 Claus Aranha, Christian L Camacho Villalón, Felipe Campelo, Marco Dorigo, Rubén Ruiz, Marc Sevaux, Kenneth Sörensen, and Thomas Stützle. Metaphor-based metaheuristics, a call for action: the elephant in the room. *Swarm Intelligence*, 16(1):1–6, 2022. doi:10.1007/S11721-021-00202-9.

3 Tim Blackwell and James Kennedy. Impact of communication topology in particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 23(4):689–702, 2018. doi:10.1109/TEVC.2018.2880894.

4 Anna Bogdanova, Jair Pereira Junior, and Claus Aranha. Franken-swarm: grammatical evolution for the automatic generation of swarm-like meta-heuristics. In *proceedings of the genetic and evolutionary computation conference companion*, pages 411–412, 2019. doi:10.1145/3319619.3321902.

5 Christian L Camacho-Villalón, Marco Dorigo, and Thomas Stützle. An analysis of why cuckoo search does not bring any novel ideas to optimization. *Computers & Operations Research*, 142:105747, 2022. doi:10.1016/J.COR.2022.105747.

6 Gustavo H de Rosa, Douglas Rodrigues, and João P Papa. Opytimizer: A nature-inspired python optimizer. *arXiv preprint arXiv:1912.13002*, 2019.

7 Tansel Dokeroglu, Ender Sevinc, Tayfun Kucukyilmaz, and Ahmet Cosar. A survey on new generation metaheuristic algorithms. *Computers & Industrial Engineering*, 137:106040, 2019.

8 Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the sixth international symposium on micro machine and human science*, pages 39–43. Ieee, 1995.

9 Andries P Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

10 Iztok Fajfar, Árpád Bűrmen, and Janez Puhan. Grammatical evolution as a hyper-heuristic to evolve deterministic real-valued optimization algorithms. *Genetic programming and evolvable machines*, 19:473–504, 2018. doi:10.1007/S10710-018-9324-5.

11 Christodoulos A Floudas and Panos M Pardalos. *Encyclopedia of optimization*. Springer Science & Business Media, 2008.

12 Martin Fowler. *Domain-specific languages*. Pearson Education, 2010.

13 Jeremy Gibbons. Functional programming for domain-specific languages. In *Central European Functional Programming School*, pages 1–28. Springer, 2013. doi:10.1007/978-3-319-15940-9_1.

14 John Hughes. Why functional programming matters. *The computer journal*, 32(2):98–107, 1989. doi:10.1093/COMJNL/32.2.98.

15 Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, 39(3):459–471, 2007. doi:10.1007/S10898-007-9149-X.

16 James Kennedy. Bare bones particle swarms. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706)*, pages 80–87. IEEE, 2003. doi:10.1109/SIS.2003.1202251.

17 James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of International Conference on Neural Networks (ICNN'95), Perth, WA, Australia, November 27 - December 1, 1995*, pages 1942–1948. IEEE, 1995. doi:10.1109/ICNN.1995.488968.

18 Michael A Lones. Metaheuristics in nature-inspired algorithms. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 1419–1422, 2014. doi:10.1145/2598394.2609841.

19 Zhongqiang Ma, Guohua Wu, Ponnuthurai Nagaratnam Suganthan, Aijuan Song, and Qizhang Luo. Performance assessment and exhaustive listing of 500+ nature-inspired metaheuristic algorithms. *Swarm and Evolutionary Computation*, 77:101248, 2023. doi:10.1016/J.SWEVO.2023.101248.

20 Kevin Martins and Rui Mendes. Cherry-picking meta-heuristic algorithms and parameters for real optimization problems. In *Progress in Artificial Intelligence: 21st EPIA Conference on*

Artificial Intelligence, EPIA 2022, Lisbon, Portugal, August 31–September 2, 2022, Proceedings, pages 500–511. Springer, 2022. doi:10.1007/978-3-031-16474-3_41.

21 Rui Mendes, James Kennedy, and José Neves. Watch thy neighbor or how the swarm can learn from its environment. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706)*, pages 88–94. IEEE, 2003. doi:10.1109/SIS.2003.1202252.

22 Rui Mendes, James Kennedy, and José Neves. The fully informed particle swarm: simpler, maybe better. *IEEE transactions on evolutionary computation*, 8(3):204–210, 2004. doi:10.1109/TEVC.2004.826074.

23 Pericles BC Miranda, Ricardo BC Prudêncio, and Gisele L Pappa. H3ad: A hybrid hyper-heuristic for algorithm design. *Information Sciences*, 414:340–354, 2017. doi:10.1016/J.INS.2017.05.029.

24 Seyedali Mirjalili. Sca: a sine cosine algorithm for solving optimization problems. *Knowledge-based systems*, 96:120–133, 2016. doi:10.1016/J.KNOSYS.2015.12.022.

25 Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey wolf optimizer. *Advances in engineering software*, 69:46–61, 2014. doi:10.1016/J.ADVENGSOFT.2013.12.007.

26 Anand Nayyar, Dac-Nhuong Le, and Nhu Gia Nguyen. *Advances in swarm intelligence for optimizing problems in computer science*. CRC press, 2018.

27 R Rao. Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *International Journal of Industrial Engineering Computations*, 7(1):19–34, 2016.

28 R Venkata Rao, Vimal J Savsani, and DP Vakharia. Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems. *Computer-aided design*, 43(3):303–315, 2011. doi:10.1016/J.CAD.2010.12.015.

29 Richard Senington and David Duke. De composing metaheuristic operations. In *Implementation and Application of Functional Languages: 24th International Symposium, IFL 2012, Oxford, UK, August 30-September 1, 2012, Revised Selected Papers 24*, pages 224–239. Springer, 2013. doi:10.1007/978-3-642-41582-1_14.

30 Erez Shinan. Lark: A modern parsing toolkit for Python. https://github.com/lark-parser/lark, 2020. Version 0.14.0 (2023).

31 Kenneth Sörensen. Metaheuristics—the metaphor exposed. *International Transactions in Operational Research*, 22(1):3–18, 2015.

32 Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997. doi:10.1023/A:1008202821328.

33 Jerry Swan, Steven Adriaensen, Mohamed Bishr, Edmund K Burke, John A Clark, Patrick De Causmaecker, Juanjo Durillo, Kevin Hammond, Emma Hart, Colin G Johnson, et al. A research agenda for metaheuristic standardization. In *Proceedings of the XI metaheuristics international conference*, pages 1–3. Citeseer, 2015.

34 Jerry Swan, Steven Adriaensen, Alexander EI Brownlee, Kevin Hammond, Colin G Johnson, Ahmed Kheiri, Faustyna Krawiec, Juan Julián Merelo, Leandro L Minku, Ender Özcan, et al. Metaheuristics "in the large". *European Journal of Operational Research*, 297(2):393–406, 2022.

35 El-Ghazali Talbi. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

36 Nguyen Van Thieu. Opfunu: an open-source python library for optimization benchmark functions. *Journal of Open Research Software*, 12(1), 2024.

37 Nguyen Van Thieu and Seyedali Mirjalili. Mealpy: An open-source library for latest meta-heuristic algorithms in python. *Journal of Systems Architecture*, 139:102871, 2023. doi:10.1016/J.SYSARC.2023.102871.

38 Grega Vrbančič, Lucija Brezočnik, Uroš Mlakar, Dušan Fister, and Iztok Fister. Niapy: Python microframework for building nature-inspired algorithms. *Journal of Open Source Software*, 3(23):613, 2018. doi:10.21105/JOSS.00613.

39 David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997. doi:10.1109/4235.585893.

40 Guohua Wu, Rammohan Mallipeddi, and Ponnuthurai Nagaratnam Suganthan. Problem definitions and evaluation criteria for the cec 2017 competition on constrained real-parameter optimization. *National University of Defense Technology, Changsha, Hunan, PR China and Kyungpook National University, Daegu, South Korea and Nanyang Technological University, Singapore, Technical Report*, 2017.

41 Xin-She Yang. Firefly algorithms for multimodal optimization. In *International symposium on stochastic algorithms*, pages 169–178. Springer, 2009. doi:10.1007/978-3-642-04944-6_14.

42 Xin-She Yang. Firefly algorithm matlab implementation. MATLAB Central File Exchange. Retrieved June 19, 2023., 2011. URL: https://www.mathworks.com/matlabcentral/fileexchange/29693-firefly-algorithm.

43 Xin-She Yang and Suash Deb. Cuckoo search via lévy flights. In *2009 World congress on nature & biologically inspired computing (NaBIC)*, pages 210–214. Ieee, 2009. doi:10.1109/NABIC.2009.5393690.

Elements for Weighted Answer-Set Programming

Francisco Coelho[1]
NOVA-LINCS, University of Évora, Portugal

Bruno Dinis
CIMA, University of Évora, Portugal

Dietmar Seipel
Universität Würzburg, Germany

Salvador Abreu
NOVA-LINCS, University of Évora, Portugal

Abstract

Logic programs, more specifically, answer-set programs, can be annotated with probabilities on facts to express uncertainty. We address the problem of propagating weight annotations on facts (*e.g.* probabilities) of an answer-set program to its stable models, and from there to events (defined as sets of atoms) in a dataset over the program's domain.

We propose a novel approach which is algebraic in the sense that it relies on an equivalence relation over the set of events. Uncertainty is then described as polynomial expressions over variables.

We propagate the weight function in the space of models and events, rather than doing so within the syntax of the program. As evidence that our approach is sound, we show that certain facts behave as expected. Our approach allows us to investigate weight annotated programs and to determine how suitable a given one is for modeling a given dataset containing events. It's core is illustrated by a running example and the encoding of a Bayesian network.

2012 ACM Subject Classification Theory of computation → Program semantics

Keywords and phrases Answer-Set Programming, Stable Models, Probabilistic Logic Programming

Digital Object Identifier 10.4230/OASIcs.SLATE.2025.3

Related Version *Full Version*: `https://arxiv.org/abs/2503.20849`

Funding This work is supported by UID/04516/NOVA Laboratory for Computer Science and Informatics (NOVA LINCS) with the financial support of FCT.IP.

Acknowledgements The authors are grateful to Lígia Henriques-Rodrigues, Matthias Knorr and Ricardo Gonçalves for valuable comments on a preliminary version of this paper, and Alice Martins for contributions on software development.

1 Introduction

Using a logic program (LP) to model and reason over a real world scenario is often difficult because of uncertainty underlying the problem being worked on. Classic LPs represent knowledge in precise and complete terms, which turns out to be problematic when the scenario is characterized by stochastic or observability factors. We aim to explore how answer-set programs (ASPs) plus weight annotated facts can lead to useful characterizations for this class of problems. To setup a working framework, we make the following assumption:

[1] Corresponding Author

▶ **Assumption 1** (System Representation, Data and Propagation). *Consider a* system *whose states are* partially observable *(*i.e. *observations can miss some state information) or* stochastic *(*i.e. *observed values are affected by random noise). We assume that knowledge about such system features a formal specification including* weighted facts *and empirical* data *such that:*

Representation *The system has a formal* representation[2] *in the form of a certain logic program; The program's stable models correspond one-to-one with the system states.*

Data Data *is a set of observations; a single* observation *(of the system states) results from a set of (boolean)* sensors*.*

Propagation *The* weights *in* facts *are* propagated *to the stable models of the representation.*

In this setting, data from observations can be used to estimate some parameteres used in the propagation process and, more importantly, to address the question of "*How accurate is the representation of the system?*". Other probabilistic logic programming (PLP) systems such as `Problog` [7], `P-log` [4] or LP^{MLN} [15], in line with Kifer and Subrahmanian [13], derive probability distributions from program syntax, limiting them to prior information. We question such methods and address stable model (SM) uncertainty by including parameters for unknown quantities, estimable with data. To frame this setting we extend our assumptions:

▶ **Assumption 2** (Sensor Representation and Events).

Sensors *The* sensors *of the system's states are associated to some atoms in the representation;*

Events *An* event *is a set of atoms from the representation.*

Following the terminology set by Calimeri *et al.* [5], sensors activate atoms (a) whereas no activation is represented by *default negation* (naf) $\sim a$. Negated atoms ($\neg a$) work similarly. By assumption 2, events can represent hidden or faulty sensors, like $\{a, \neg a, \sim b, \sim\neg b, \sim c, \neg c\}$, where a and $\neg a$ are activated, b is hidden, and $\neg c$ is consistently activated. Not all events are observations, and some may not uniquely determine system states – how to associate events to stable models and, thus, to system states, is addressed in Section 4.

If we (i) omit the naf-literals; (ii) use $\overline{x}$ to denote the classical $\neg x$; (iii) and use expressions like ab to denote sets of literals such as $\{a, b\}$, then the event $\{a, \neg a, \sim b, \sim\neg b, \sim c, \neg c\}$ can be shortened to the equivalent form $a\overline{a}\overline{c}$. We follow the convention of denoting a model by "*true*" atoms (positive or negative), with "*falsehood*" resulting from default negation [11]. Models include atoms like a or $\overline{b}$ but not literals $\sim a$. Sensor input is represented using positive and negative atoms, considering different sensors or a single sensor yielding positive/negative values.

Weights, not probabilities, are used to define a weight function on SMs, extended to all events. The step from facts to SMs is non-deterministic, and parameters represent non-unique choices, estimable with data. ASPs, based on SMs semantics, use SAT solving [10, 1, 19] or top-down search [2, 3, 18]. Distribution semantics (DS) [25, 23] extends logic programs with probabilistic reasoning. We are particularly interested in the following setting and application scenarios of such an extension to logic programs:

Partial observability A system's state can have *hidden variables*, not reported by the sensors.

Sensor error Information gathered from the sensors can carry *stochastic perturbations.*

Representation induction Combine representations and data to induce *more accurate representations.*

[2] We use "representation" instead of "model" to avoid confusion with the *stable models* of answer-set programs.

Probabilistic tasks Support common probabilistic tasks such as maximum a posteriori (MAP), maximum likelihood estimation (MLE) and Bayesian inference (BI) on the *representation domain i.e.* the set of all events.

Probabilistic tasks and simple examples

Maximum a posteriory (MAP) MAP estimates model parameters by combining prior beliefs with data likelihood to find the most probable values:

$$\hat{\theta}_{\text{MAP}} = \arg\max_{\theta} \mathrm{P}(\theta \mid X)$$

For a biased coin with heads probability θ following a Beta distribution, observing 7 heads in 10 flips, the MAP estimate of θ maximizes the posterior distribution combining the prior Beta and the likelihood of the observations.

Maximum likelihood estimation (MLE) MLE estimates model parameters by maximizing the likelihood of observed data:

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \mathrm{P}(X \mid \theta)$$

For 7 heads in 10 flips, the MLE estimate of the probability of heads is $\theta = 0.7$.

Bayesian inference (BI) Bayesian Inference updates hypothesis probabilities by combining prior beliefs with observed data to produce a new probability distribution.

$$\mathrm{P}(\theta \mid X) = \frac{\mathrm{P}(X \mid \theta)\,\mathrm{P}(\theta)}{\mathrm{P}(X)}$$

Bayesian Inference updates a Beta prior (α, β) with 7 heads and 3 tails from 10 flips, resulting in a Beta posterior $(\alpha + 7, \beta + 3)$.

The remainder of this article is structured as follows: Section 2 provides necessary context. In Section 3 we discuss the syntax and semantics of our proposed language for weighted answer-set programs (WASPs). We also define a weight function over total choices and address the issue of how to propagate these weights from facts to events, in Section 4. This method relies on an equivalence relation on the set of events. Furthermore, we express uncertainty by polynomial expressions over variables which depend on the total choices and on the stable models. By then the *Partial Observability* and *Sensor Error* points are addressed. An evidence that our approach is sound is given by Equation (28) where we show that replacing certain facts (with weight 1.0) by deterministic facts does not change the probability. Some final remarks and ideas for future developments including *Representation Induction* and the *Probabilistic Tasks* are presented in Section 5.

2 Framework

We start by refining assumptions 1 and 2 to set "representation" as an "ASP with weights":

▶ **Assumption 3** (Representation by answer-set programs with weigths)**.**

Answer-set programs and weights *A* representation *of a system is an* answer-set program *that includes* weighted facts*.*

A weighted fact (WF) *or an* annotated fact *has the from "*$a : w$*" where* a *is an atom,* $w \in [0, 1]$*, and "derives" the disjunctive fact "*$a \vee \overline{a}$*". A model may include either* a *or* $\overline{a}$ *but never both.*

Selecting one of $a, \overline{a}$ for each WF in a program will lead to a *total choice (TC)* [3]. Propagating weights from WFs to TCs is relatively straightforward (see Equation (6)) but propagation to events requires a step trough the program's stable models, addressed in Section 4.

About propagating weights from total choices

Propagating weights from TCs to SMs and events faces non-determinism, as seen in program P_1 from ex. 4, where multiple SMs (ab, ac) result from a single TC (a) without clear weight assignment. We use algebraic variables to address this, allowing deterministic propagation. Variable values can be estimated from data, contributing to Representation Induction. Related works use credal sets [6] or Bayesian approaches [28] to handle uncertainty, while our method remains algebraic.

3 Syntax and Semantics of Weighted ASP

We start the formal definition of *weighted answer-set program* with the setup and discussion of a minimal syntax and semantics of propositional ASP, without variables, functors or relation symbols, but enough to illustrate our method to propagate weights from annotated facts to events. From now on "$\neg x$" and "$\overline{x}$" denote classical negation and "$\sim x$" default negation.

Syntax

We slightly adapt Calimeri *et al.*'s approach [5]. Let $\mathcal{A}$ be a finite set of symbols, the *positive atoms*. For $a \in \mathcal{A}$, the expressions a and $\neg a$ (the later a *negative atom*, also denoted $\overline{a}$) are *(classical) atoms*. If a is an atom, the expressions a and $\sim a$ are *(naf-)literals*. A *rule* is of the form

$$h_1 \vee \cdots \vee h_n \leftarrow b_1 \wedge \cdots \wedge b_m$$

where the h_i are atoms and the b_j are literals. The symbol "$\leftarrow$" separates the *head* from the *body*. A rule is a *constraint*[4] if $n = 0$, *normal* if $n = 1$, *disjunctive* if $n > 1$, and a *fact* if $m = 0$.

An *answer-set program (ASP)* is a set P of facts and other rules, denoted, resp. $\mathcal{F}(P)$ and $\mathcal{R}(P)$, or simply $\mathcal{F}$ and $\mathcal{R}$. In a *normal program* all the rules are normal. Notice that programs with constraint or disjunctive rules can be converted into normal programs [9].

Semantics

The standard semantics of an ASP has a few different, but equivalent, definitions [17]. A common definition is as follows [11]: let P be a normal program. The Gelfond/Lifschitz *reduct* of P relative to the set X of atoms results from (i) deleting rules that contain a literal of the form $\sim p$ in the body with $p \in X$ and then (ii) deleting the remaining literals of the form $\sim q$ from the bodies of the remaining rules. Now, M is a *stable model (SM)* of P if it is the minimal model of the reduct of P relative to M. We denote by $\mathcal{S}(P)$, or simply $\mathcal{S}$, the set of stable models of the program P.

---

[3] We use the term "choice" for historical reasons, *e.g.* see [6] even though not related to the usual "choice" elements, atoms or rules from *e.g.* [5]

[4] An "*integrity constraint*" in [5].

Evaluation without grounding

While the most common form to generate stable models is based on *grounding*, a different approach is the one supported by `s(CASP)`, that evaluates ASP programs with function symbols and constraints without grounding, enabling human-readable explanations and addressing grounding-based solvers' issues of exponential growth and unnecessary complete model computation.

WASPs and their Derived Programs

If a is an atom and $w \in [0,1]$, a *weighted fact (WF)* (or a *fact with weight annotation*) is $a : w$. Notice that we have $w \in [0,1]$ but w is interpreted as a *balance* between the *choices* a and $\overline{a}$, and *not a probability*.

Weighted answer-set programs (WASPs) extend ASPs by adding WFs. We denote the set of weighted facts of a program P by $\mathcal{W}(P)$, and $\mathcal{A}_{\mathcal{W}}(P)$ the set of positive atoms in $\mathcal{W}$. When possible we simplify notation as $\mathcal{W}, \mathcal{A}_{\mathcal{W}}$.

Our WASPs definition is restricted to illustrate weight propagation from TCs to events, excluding logical variables, relation symbols, and functors. Weight annotations aren't used on rule heads or disjunctions, but this doesn't limit expressiveness:

$$\alpha : w \leftarrow \beta \qquad \Longrightarrow \qquad \begin{cases} \gamma : w, \\ \alpha \leftarrow \beta \wedge \gamma \end{cases} \tag{1}$$

while annotated disjunctive facts

$$\alpha \vee \beta : w \qquad \Longrightarrow \qquad \begin{cases} \gamma : w, \\ \alpha \vee \beta \leftarrow \gamma, \\ \overline{\alpha} \leftarrow \overline{\gamma}, \quad \overline{\beta} \leftarrow \overline{\gamma}. \end{cases} \tag{2}$$

Derived program

The *derived program* of a WASP is the ASP obtained by replacing each weighted fact $a : w$ by the derived disjunction $a \vee \overline{a}$. The *stable models* of a WASP program are the stable models of its derived program. So, we also denote the set of SMs of a (derived or) WASP program P by $\mathcal{S}(P)$ or $\mathcal{S}$.

Events

An *event* of a program P is a set of atoms from P. We denote the set of events by $\mathcal{E}(P)$ or simply $\mathcal{E}$. An event $e \in \mathcal{E}$ which includes a set $\{x, \overline{x}\}$ is said to be *inconsistent*; otherwise it is *consistent*. The set of consistent events is denoted by $\mathcal{C}$.

▶ **Example 4** (A simple weighted answer-set program)**.** Consider the following WASP :

$$P_1 = \begin{cases} a : 0.3, \\ b \vee c \leftarrow a \end{cases} \tag{3}$$

with weighted facts $\mathcal{W} = \{a : 0.3\}$. The derived program is the ASP

$$P_1' = \begin{cases} a \vee \overline{a}, \\ b \vee c \leftarrow a, \end{cases} \tag{4}$$

with SMs $\mathcal{S} = \{\, \overline{a}, ab, ac \,\}$. The atoms are $\mathcal{A} = \left\{ a, \overline{a}, b, \overline{b}, c, \overline{c} \right\}$ and the events are $\mathcal{E} = 2^{\mathcal{A}}$ [5].

[5] 2^X is the *power set* of X: $A \in 2^X \Leftrightarrow A \subseteq X$.

Total Choices and their Weights

A disjunctive head $a \vee \overline{a}$ in the derived program represents a single *choice*, either a or $\overline{a}$. We define the set of *total choices (TCs)* of a set of atoms by the recursion

$$\begin{cases} \mathcal{T}(\emptyset) = \emptyset, \\ \mathcal{T}(X \cup a) = \bigcup\limits_{t \in \mathcal{T}(X)} (t \cup a) \quad \cup \quad \bigcup\limits_{t \in \mathcal{T}(X)} (t \cup \overline{a}) \end{cases} \tag{5}$$

where X is a set of atoms and a is an atom. The total choices of a WASP P are the TCs of the positive atoms in it's weighted facts: $\mathcal{T}(P) = \mathcal{T}\big(\mathcal{A}_{\mathcal{W}}(P)\big)$. When possible we write simply $\mathcal{T}$. Given a WASP, the *weight of the total choice* $t \in \mathcal{T}$ is given by the product

$$\omega_{\mathcal{T}}(t) = \prod_{\substack{a : w \in \mathcal{W}, \\ a \in t}} w \quad \times \quad \prod_{\substack{a : w \in \mathcal{W}, \\ \overline{a} \in t}} \overline{w}. \tag{6}$$

Here $\overline{w} = 1 - w$, and we use the subscript in $\omega_{\mathcal{T}}$ to explicitly state that this function concerns total choices. Later we'll use subscripts $\mathcal{S}, \mathcal{E}$ to deal with weight functions of stable models and events, $\omega_{\mathcal{S}}, \omega_{\mathcal{E}}$. Some stable models are entailed from some total choices while other SMs are entailed by other TCs. We write $\mathcal{S}(t)$ to represent the set of stable models entailed by the total choice $t \in \mathcal{T}$. Our goal can now be rephrased as to know how to propagate the weights of the program's total choices, $\omega_{\mathcal{T}}$, in Equation (6) to the program's events, $\omega_{\mathcal{E}}$ to be defined later, in Equations (21a) and (21b).

Propagation of weights

As a first step to propagate weight from total choices to events, consider the program P_1 of Equation (3) and a possible propagation of $\omega_{\mathcal{T}} : \mathcal{T} \to [0, 1]$ from total choices to the stable models, $\omega_{\mathcal{S}} : \mathcal{S} \to [0, 1]$ (still informal, see Equation (16)). It might seem straightforward, in ex. 4, to set $\omega_{\mathcal{S}}(\overline{a}) = 0.7$ but there is no *explicit* way to assign values to $\omega_{\mathcal{S}}(ab)$ and $\omega_{\mathcal{S}}(ac)$. We represent this non-determinist by a parameter θ as in

$$\begin{aligned} &\omega_{\mathcal{S}}(ab) = 0.3\,\theta, \\ &\omega_{\mathcal{S}}(ac) = 0.3\,(1 - \theta) \end{aligned} \tag{7}$$

to express our knowledge that ab and ac are models entailed from a specific choice and, simultaneously, the inherent non-determinism of that entailment. In general, it might be necessary to have several such parameters, each associated to a given stable model s (in Equation (7), $s = ab$ in the first line and $s = ac$ in the second line) and a total choice t ($t = a$ above), so we write $\theta_{s,t}$. Obviously, for reasonable $\theta_{s,t}$, the total choice t must be a subset of the stable model s.

Unless we introduce some bias, such as $\theta = 0.5$ as in $\mathtt{LP}^{\mathtt{MLN}}$ [15], the values for $\theta_{s,t}$ can't be determined just with the information given in the program. But it might be estimated with the help of further information, such as an empirical distribution from a dataset. Further discussion of this point is outside the scope of this paper.

Now consider the program

$$\begin{cases} a : 0.3, \\ b \leftarrow a \wedge {\sim}b \end{cases} \tag{8}$$

that has a single SM, $\overline{a}$. Since the weights are not interpreted as probabilities, there is no need to have the sum on the stable models equal to 1. So the weights in the TCs of Equation (8) only set

$$\omega_{\mathcal{S}}(\overline{a}) = 0.7.$$

In this case, if we were to derive a probability of the SMs, normalization would give $\mathrm{P}(\overline{a}) = 1.0$.

Also facts without annotations can be transformed into facts with weight 1:

$$a \qquad \Longrightarrow \qquad a : 1.0. \tag{9}$$

The method that we are proposing does not follow the framework of Kifer and Subrahmanian [13] and others, where the syntax of the program determines the propagation from probabilities explicitly set either in facts or other elements of the program. Our approach requires that we consider the semantics, *i.e.* the stable models of the program, independently of the syntax that provided them. From there we propagate weights to the program's events and then, if required, normalization provides the final probabilities. Moreover, we allow the occurrence of variables in the weights, in order to deal with the non-determinism that results from the non-uniqueness of SMs entailed from a single TC. These variables can be later estimated from available data.

Related Approaches and Systems

The core problem of setting a semantics for probabilistic logic programs, the propagation of probabilities from total choices to stable models in the case of ASP or to other types in other logic programming systems (*e.g.* to possible worlds in `Problog`) has been studied for some time [13, 25]. For example, the *credal set* approach [6], defines $\mathrm{P}_{\mathcal{T}}$ in a way similar to Equation (6) but then, for $a \in \mathcal{A}, t \in \mathcal{T}$, the probability $\mathrm{P}(a \mid t)$ is unknown but bounded by $\underline{\mathrm{P}}(a \mid t)$ and $\overline{\mathrm{P}}(a \mid t)$, that can be explicitly estimated from the program.

`Problog` [8, 28] extends `Prolog` with probabilistic facts so that a program specifies a probability distribution over possible worlds. A *world* is a model of $T \cup R$ where T is a total choice and R the set of rules of a program. The semantics is only defined for *sound* programs [24] *i.e.*, programs for which each possible total choice T leads to a well-founded model that is two-valued or *total*. The probability of a possible world that is a model of the program is the probability of the total choice. Otherwise the probability is 0 [24, 27]. Another system, based on Markov Logic [22], is $\mathtt{LP}^{\mathtt{MLN}}$ [15, 16], whose models result from *weighted rules* of the form $a \leftarrow b \wedge n$ where a is disjunction of atoms, b is conjunction of atoms and n is constructed from atoms using conjunction, disjunction and negation. For each model there is a unique maximal set of rules that are satisfied by it and the respective weights determine the weight of that model, that can be normalized to a probability.

Towards Propagating Weights from Total Choices to Events

The program P_1 in Equation (3) from ex. 4 showcases the problem of propagating weights from total choices to stable models and then to events. The main issue arises from the lack of information in the program on how to assign *un-biased* weights to the stable models. This becomes crucial in situations where multiple stable models result from a single total choice.

Our assumptions 1–3 enunciate that a WASP program represents a *system*; the *states* of that system, which are partially observable and stochastic, are associated to the program's stable models; and state *observations* are encoded as events, *i.e.* sets of atoms of the program. Then:

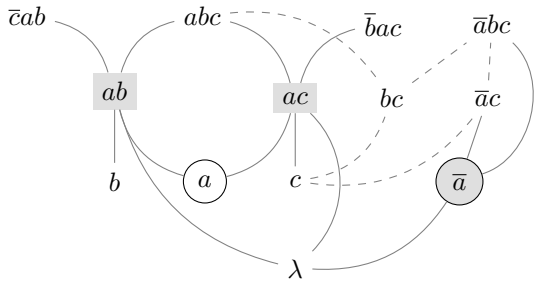


Figure 1 This diagram shows events related to the stable models of program P_1. Circle nodes are total choices, shaded nodes are SMs. Solid lines show SM relations, dashed lines show subset/superset relations. The set of events in all SMs, Λ, is $\{\lambda\}$ because $\overline{a} \cap ab \cap ac = \emptyset$.

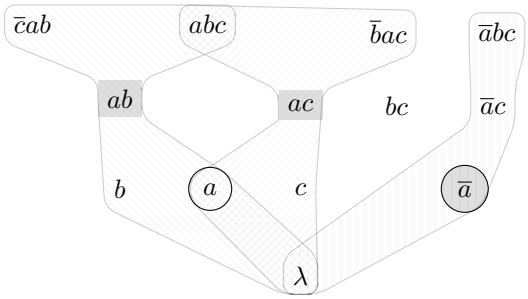


Figure 2 Classes of (consistent) events related to the stable models of P_1 are defined through sub-/super-set relations. In this picture we can see, for example, that $\{\overline{c}ab, ab, b\}$ and $\{a, abc\}$ are part of different classes, represented by different fillings. As before, the circle nodes are total choices and shaded nodes are stable models. Notice that bc is not in a filled area.

1. With a weight set for the stable models, we extend it to any event in the program domain.
2. If statistical knowledge is available, it's considered external and doesn't affect the propagation procedure.
3. However, that knowledge can be used to estimate the parameters $\theta_{s,t}$ and to "score" the program.
4. If a program is one of many candidates, its score can be used as fitness by algorithms searching for optimal programs in a dataset.
5. If events are not consistent with the program, then we ought to conclude that the program is wrong and must be changed accordingly.

We address propagating weights from stable models to events using parameters like θ. This defines a function $\mu_{\mathcal{E}}$ that can be normalized to set probabilities $\mathrm{P}_{\mathcal{E}}$, ensuring consistent probabilistic reasoning with the ASP program.

4 Propagating Weights

The diagram in Figure 1 shows the challenge of propagating weights from total choices to stable models and then to general events, where node values depend on neighbours. This leads to interpretation issues, such as assigning unexplained values to nodes like bc. We propose basing propagation on the event's relation to stable models instead.

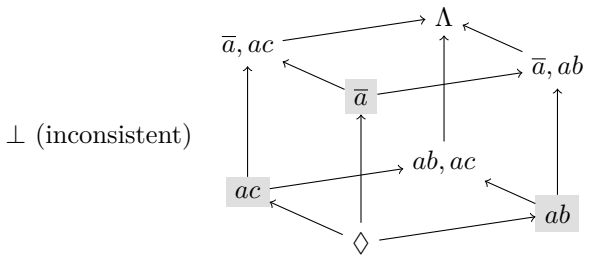


Figure 3 This diagram shows the lattice of stable cores from ex. 4. Nodes are stable cores derived from stable models, plus the inconsistent class. The bottom node is the independent events class, with no relation to SMs. The top node represents events related to all SMs (*consequences*). Shaded nodes are SMs.

4.1 An Equivalence Relation

Our approach to propagating weights views stable models as *prime factors* or "irreducible events". Events are considered based on their relation to SMs. In ex. 4, a relates to ab and ac, while c only relates to ac. Thus, a and c relate to different SMs, but a and abc relate to the same SMs. We formalize this relation. The *stable core (SC)* of the event $e \in \mathcal{E}$ is

$$[\![e]\!] := \{\, s \in \mathcal{S} \mid s \subseteq e \vee e \subseteq s \,\} \tag{10}$$

where $\mathcal{S}$ is the set of stable models.

Notice that the minimality of stable models implies that either e is a stable model or at least one of $\exists s\,(s \subseteq e)\,, \exists s\,(e \subseteq s)$ is false *i.e.,* no stable model contains another. We now define an equivalence relation so that two events are related if either both are inconsistent or both are consistent and, in the latter case, with the same stable core.

▶ **Definition 5** (Equivalence Relation on Events). *For a given program, let $u, v \in \mathcal{E}$. The equivalence relation $u \sim v$ is defined by*

$$u, v \notin \mathcal{C} \vee \big(u, v \in \mathcal{C} \wedge [\![u]\!] = [\![v]\!]\big)\,. \tag{11}$$

This equivalence relation defines a partition on the set of events, where each class holds a unique relation with the stable models. In particular we denote each class by:

$$[e]_\sim = \begin{cases} \bot := \mathcal{E} \setminus \mathcal{C} & \text{if } e \in \mathcal{E} \setminus \mathcal{C}, \\ \big\{u \in \mathcal{C} \;\big|\; [\![u]\!] = [\![e]\!]\big\} & \text{if } e \in \mathcal{C}. \end{cases} \tag{12}$$

where $\bot$ denotes the set $\mathcal{E} \setminus \mathcal{C}$ of *inconsistent* events, *i.e.* events that contain $\{x, \overline{x}\}$ for some atom x.

Let λ be the empty set event (notice that $\lambda = \emptyset \in \mathcal{E}$) [6], and Λ the *consequence class* of (consistent) events related with all the stable models. Then

$$[\lambda]_\sim = \Lambda. \tag{13}$$

The combinations of stable models, *i.e.* the stable cores, together with the set of inconsistent events ($\bot$) forms a set of representatives for the equivalence relation $\sim$. Since all

[6] We adopt the notation "λ" for *empty word*, from formal languages, to distinguish "$\emptyset \in \mathcal{E}$" from "$\emptyset \subset \mathcal{E}$".

events within a consistent equivalence class have the same stable core, we are interested in functions (including weight assignments), that are constant within classes. A function $f : \mathcal{E} \to Y$, where Y is any set, is said to be *coherent* if

$$\forall e \in \mathcal{E}\ \forall u \in [e]_\sim \left(f(u) = f(e)\right). \tag{14}$$

Considering coherent functions, in the specific case of Equation (3), instead of dealing with the $2^6 = 64$ events, we need to consider only the $2^3 + 1 = 9$ classes, well defined in terms of combinations of the stable models, to define coherent functions. In general, a program with n atoms and m stable models has 2^{2n} events and $2^m + 1$ stable cores – but easily $m \gg n$.

4.2 From Total Choices to Events

The "propagation" phase, traced by Equation (6) and Equations (16)–(21b), starts with the weight of total choices, $\omega_\mathcal{T}(t)$, propagates it to the stable models, $\omega_\mathcal{S}(s)$, and then, within the equivalence relation from Equation (11), to a coherent weight of events, $\omega_\mathcal{E}(e)$. So we are specifying a sequence of functions

$$\omega_\mathcal{T} \longrightarrow \omega_\mathcal{S} \longrightarrow \omega_\mathcal{R} \longrightarrow \omega_\mathcal{E} \tag{15}$$

on successive larger domains

$$\mathcal{T} \longrightarrow \mathcal{S} \longrightarrow [\mathcal{E}]_\sim \longrightarrow \mathcal{E}$$

such that the last function ($\omega_\mathcal{E}$) is a finite coherent function on the set of events and thus, as a final step, it can easily be used to define a probability distribution of events by normalization:

$$\omega_\mathcal{E} \longrightarrow \mathrm{P}_\mathcal{E}.$$

Total choices and Stable models

Let's start by looking into the first two steps of the sequence of functions Equation (15): $\omega_\mathcal{T}$ and $\omega_\mathcal{S}$. The weight $\omega_\mathcal{T}$ of the total choice $t \in \mathcal{T}$ is already given by Equation (6). Recall that each total choice $t \in \mathcal{T}$, together with the rules and the other facts of a program, defines the set $\mathcal{S}(t)$ of stable models associated with that choice. Given a total choice $t \in \mathcal{T}$, a stable model $s \in \mathcal{S}$, and formal variables or values $\theta_{s,t} \in [0, 1]$ such that $\sum_{s \in \mathcal{S}(t)} \theta_{s,t} = 1$, we define

$$\omega_\mathcal{S}(s, t) := \begin{cases} \theta_{s,t} & \text{if } s \in \mathcal{S}(t) \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

The $\theta_{s,t}$ parameters in Equation (16) express the *program's* lack of information about the weight assignment, when a single total choice entails more than one stable model. We address this issue by assigning a possibly unknown parameter, *i.e.* a formal algebraic variable ($\theta_{s,t}$) associated with a total choice (t) and a stable model (s). This allows the expression of a quantity that does not result from the program's syntax but can be determined or estimated given more information, *e.g.* observed data.

As sets, two stable models can have non-empty intersection. But because different SMs represent different states of a system –which are *disjoint events* – we assume that the algebra of the stable models is σ-additive.

▶ **Assumption 6** (Stable models as disjoint events). *For any set X of stable models and any total choice t,*

$$\omega_{\mathcal{S}}(X,t) = \sum_{s\in X} \omega_{\mathcal{S}}(s,t)\,. \tag{17}$$

Equation (17) is the basis for Equation (19a) and effectively extends $\omega_{\mathcal{S}} : \mathcal{S}\times\mathcal{T}\to\mathbb{R}$ to $\omega_{\mathcal{S}} : 2^{\mathcal{S}}\times\mathcal{T}\to\mathbb{R}$. Now the pre-condition of Equation (16) can be stated as $\omega_{\mathcal{S}}\big(\mathcal{S}(t)\,,t\big) = 1$.

Classes

The next step in the sequence of Equation (15) is the function $\omega_{\mathcal{R}}$ on $[\mathcal{E}]_\sim$. Each class of the relation $\sim$ (eq. 11) is either the inconsistent class ($\bot$) or is associated with a stable core, *i.e.* a set of stable models. Therefore, $\omega_{\mathcal{R}}$ is defined considering the following two cases.

Inconsistent class

This class contains inconsistent events; they should not be observed and have weight zero.

$$\omega_{\mathcal{R}}(\bot,t) := 0. \tag{18}$$

Consistent classes

To be coherent, the propagation function must be constant within a class and its value dependent only on the stable core:

$$\omega_{\mathcal{R}}\big([e]_\sim,t\big) := \omega_{\mathcal{S}}\big(\llbracket e\rrbracket\,,t\big) = \sum_{s\in\llbracket e\rrbracket} \omega_{\mathcal{S}}(s,t)\,. \tag{19a}$$

and we further define the following:

$$\omega_{\mathcal{R}}\big([e]_\sim\big) := \sum_{t\in\mathcal{T}} \omega_{\mathcal{T}}(t)\,\omega_{\mathcal{R}}\big([e]_\sim,t\big) \tag{19b}$$

Equation (19a) states that the weight of a class $[e]_\sim$ is the weight of its stable core ($\llbracket e\rrbracket$) and Equation (19b) *averages* Equation (19a) over the total choices. Notice that Equation (19a) also applies to the independent class, $\Diamond = \big\{e \;\big|\; \llbracket e\rrbracket = \emptyset\big\}$, because events in this class are not related with any stable model:

$$\omega_{\mathcal{R}}(\Diamond,t) = \sum_{s\in\emptyset} \omega_{\mathcal{S}}(s,t) = 0. \tag{20}$$

Events and Probability

Each consistent event $e\in\mathcal{E}$ is in the class defined by its stable core $\llbracket e\rrbracket$. So, denoting the number of elements in X as $\#X$, we set:

$$\omega_{\mathcal{E}}(e,t) := \begin{cases} \frac{\omega_{\mathcal{R}}\left([e]_\sim,t\right)}{\#[e]_\sim} & \text{if } \#[e]_\sim > 0,\\ 0 & \text{otherwise.}\end{cases} \tag{21a}$$

and, by averaging over the total choices:

$$\omega_{\mathcal{E}}(e) := \sum_{t\in\mathcal{T}} \omega_{\mathcal{T}}(t)\,\omega_{\mathcal{E}}(e,t)\,. \tag{21b}$$

The Equation (21b) is the main goal of this paper: propagate the weights associated to facts of an WASP to the set of all events of that program. In order to get a probability from Equation (21b), concerning the *Probabilistic Tasks* goal, we define the *normalizing factor*:

$$Z := \sum_{e \in \mathcal{E}} \omega_{\mathcal{E}}(e) = \sum_{[e]_\sim \in [\mathcal{E}]_\sim} \omega_{\mathcal{R}}\left([e]_\sim\right), \tag{22}$$

and now Equation (21b) provides a straightforward way to define the *probability of a single event* $e \in \mathcal{E}$:

$$\mathrm{P}_{\mathcal{E}}(e) := \frac{\omega_{\mathcal{E}}(e)}{Z}. \tag{23}$$

Equation (23) defines a coherent *prior*[7] probability of events and, together with external statistical knowledge, can be used to learn about the *initial* probabilities of the atoms.

The effect of propagation

To assess weight propagation from SMs to events, compare $\mathrm{P}_{\mathcal{E}}$ with syntactically induced probabilities from fact weights. It is sufficient to check if $\mathrm{P}_{\mathcal{E}}(t) = \mathrm{P}_{\mathcal{T}}(t)$ for all total choices. Normalize TCs weights to get a probability distribution. For $t \in \mathcal{T}$,

$$\mathrm{P}_{\mathcal{T}}(t) = \frac{\omega_{\mathcal{T}}(t)}{\sum_{\tau \in \mathcal{T}} \omega_{\mathcal{T}}(\tau)} \tag{24}$$

And now we ask if these probabilities coincide in $\mathcal{T}$: $\forall t \in \mathcal{T}\left(\mathrm{P}_{\mathcal{E}}(t) = \mathrm{P}_{\mathcal{T}}(t)\right)$?

It is easy to see that, in general, this cannot be true. While the domain of $\mathrm{P}_{\mathcal{T}}$ is the set of total choices, for $\mathrm{P}_{\mathcal{E}}$ the domain is much larger, including all the events. Except for trivial programs, some events other than total choices will have non-zero weight: If a program has a consistent event $e \in \mathcal{C} \setminus \mathcal{T}$ such that $\mathrm{P}_{\mathcal{E}}(e) \neq 0$ then there is at least one $t \in \mathcal{T}$ such that

$$\mathrm{P}_{\mathcal{T}}(t) \neq \mathrm{P}_{\mathcal{E}}(t)\,. \tag{25}$$

The essential, perhaps *counter-intuitive*, conclusion of Equation (25) is that we are dealing with *two distributions*: $\mathrm{P}_{\mathcal{T}}$, restricted to the total choices, results *syntactically* from the annotations of the program, while $\mathrm{P}_{\mathcal{E}}$, extended to events, results from both the annotations and the program's *semantics*, *i.e.* the stable models. For ex. 4:

$\mathrm{P}_{\mathcal{T}}(a) = 0.3$ from the program and $\mathrm{P}_{\mathcal{E}}(a) = \dfrac{3}{64}$ from Equation (29).

Now $\mathrm{P}_{\mathcal{E}} : \mathcal{E} \to [0,1]$ can be extended to $\mathrm{P}_{\mathcal{E}} : 2^{\mathcal{E}} \to [0,1]$ by abusing notation and setting, for $X \subseteq \mathcal{E}$,

$$\mathrm{P}_{\mathcal{E}}(X) = \sum_{x \in X} \mathrm{P}_{\mathcal{E}}(x)\,. \tag{26}$$

It is straightforward to verify that the latter satisfies the Kolmogorov axioms of probability.

We can now properly state the following property about *certain facts* such as $a : 1.0$. Consider a program A with the weighted fact $\alpha : 1.0$ and B that results from A by replacing that fact by the deterministic fact α. Then

$$\forall e \in \mathcal{E}\left(\omega^A_{\mathcal{E}}(e) = \omega^B_{\mathcal{E}}(e)\right). \tag{27}$$

[7] In the Bayesian sense that future observations might update this probability.

Normalization of Equation (27) entails that, for $\mathrm{P}^A_{\mathcal{E}}$ given by Equation (23) for A and $\mathrm{P}^B_{\mathcal{E}}$ for B, then

$$\forall e \in \mathcal{E}\left(\mathrm{P}^A_{\mathcal{E}}(e) = \mathrm{P}^B_{\mathcal{E}}(e)\right). \tag{28}$$

▶ **Example 7** (Probability of Events). The coherent *prior* probability of events of program P_1 in ex. 4 is

$[\![e]\!]$	$\bot$	$\Diamond$	$\overline{a}$	ab	ac	$\overline{a}, ab$	$\overline{a}, ac$	ab, ac	Λ
$\mathrm{P}_{\mathcal{E}}(e)$	0	0	$\frac{7}{207}$	$\frac{1}{23}\theta$	$\frac{1}{23}\overline{\theta}$	0	0	$\frac{3}{46}$	$\frac{10}{23}$

(29)

To compute the probability of $e \in \mathcal{E}$ find the column of the event's stable core:

- $\omega_{\mathcal{E}}(ab) = \frac{\theta}{23}$, because ab is the only SM related with ab so $[\![ab]\!] = \{ab\}$ and the weight value is found in the respective column of Equation (29).
- $\omega_{\mathcal{E}}(abc) = \frac{3}{46}$ because $abc \supset ab$ and $abc \supset ac$. So $[\![abc]\!] = \{ab, ac\}$.
- $\omega_{\mathcal{E}}(bc) = 0$ because, since there is no SM s that either $s \subset bc$ or $bc \subset s$, $[\![bc]\!] = \emptyset$ *i.e.* $bc \in \Diamond$.
- $\omega_{\mathcal{E}}(\overline{a}) = \frac{7}{207}$ and $\omega_{\mathcal{E}}(a) = \frac{3}{46}$.

Notice that $\omega_{\mathcal{E}}(\overline{a}) + \omega_{\mathcal{E}}(a) \neq 1$. This highlights the fundamental difference between $\omega_{\mathcal{E}}$ and $\omega_{\mathcal{T}}$ (*cf.* Equation (25)), where the former results from the lattice of the stable cores and the latter directly from the explicit assignment of probabilities to literals. Related with this case, consider the *complement* of a consistent event e, denoted by $\complement e$. To calculate $\mathrm{P}_{\mathcal{E}}(\complement e)$ find the classes in $[\mathcal{E}]_\sim$ that are not $[e]_\sim$, *i.e.* the complement of e's class within $[\mathcal{E}]_\sim$[8], $\complement[e]_\sim$. Considering that $[\mathcal{E}]_\sim$ is in a one-to-one correspondence with the stable cores plus $\bot$,

$$[\mathcal{E}]_\sim \simeq \left\{\bot, \Diamond, \{\overline{a}\}, \{ab\}, \{ac\}, \{\overline{a}, ab\}, \{\overline{a}, ac\}, \{ab, ac\}, \Lambda\right\}.$$

In particular, for $\omega_{\mathcal{E}}(\complement a)$, since $[\![a]\!] = \{ab, ac\}$ then $\complement[a]_\sim = [\mathcal{E}]_\sim \setminus [a]_\sim$ and $\mathrm{P}_{\mathcal{E}}(\complement a) = \mathrm{P}_{\mathcal{E}}([\mathcal{E}]_\sim \setminus [a]_\sim) = 1 - \mathrm{P}_{\mathcal{E}}(a)$. Also, $\mathrm{P}_{\mathcal{E}}(\complement\overline{a}) = 1 - \mathrm{P}_{\mathcal{E}}(\overline{a})$.

While not illustrated in our examples, this method also applies to programs that have more than one probabilistic fact, like

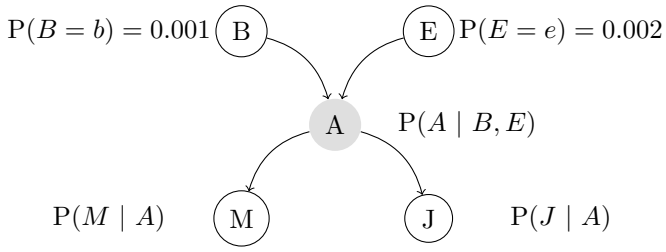
$$\begin{cases} a : 0.3, \qquad b : 0.6, \\ c \vee d \leftarrow a \wedge b. \end{cases}$$

4.3 Encoding of Bayesian Networks

Our approach generalizes to Bayesian networks similarly to previous works [6, 21, 12, 26]. Acyclic propositional programs can be seen as Bayesian networks with binary variables, where the structure is the dependency graph, and variables correspond to atoms with probabilities from facts and rules. Conversely, any binary Bayesian network can be specified by an acyclic non-disjunctive WASP. A classic example in Bayesian networks involves the events Burglary (B), Earthquake (E), Alarm (A), and calls from Mary (M) and John (J) [20]. These events have conditional probabilities, such as the alarm going off given a burglary or earthquake, and neighbors calling if they hear the alarm. The example illustrates reasoning under uncertainty. We follow the convention of representing the (upper case) random variable X by the (lower case) positive atom x. From Figure 4 we obtain the following specification:

$$\begin{cases} b : 0.001, \\ e : 0.002 \end{cases}$$

[8] All the usual set operations hold on the complement. For example, $\complement\complement X = X$.



P(M | A)	m	$\overline{m}$
a	0.9	0.1
$\overline{a}$	0.05	0.95

P(J | A)	j	$\overline{j}$
a	0.7	0.3
$\overline{a}$	0.01	0.99

P(A | B, E)		a	$\overline{a}$
b	e	0.95	0.05
b	$\overline{e}$	0.94	0.06
$\overline{b}$	e	0.29	0.71
$\overline{b}$	$\overline{e}$	0.001	0.999

Figure 4 The Earthquake, Burglary, Alarm model.

For the table giving the probability $\mathrm{P}(M \mid A)$ we obtain, cf. Equation (1), the program

$$\begin{cases} m : 0.9 \leftarrow a, \\ m : 0.05 \leftarrow \overline{a} \end{cases}$$

Similarly, for the probability $\mathrm{P}(J \mid A)$ we obtain

$$\begin{cases} j : 0.7 \leftarrow a, \\ j : 0.01 \leftarrow \overline{a}, \end{cases}$$

Finally, for the probability $\mathrm{P}(A \mid B \wedge E)$ we obtain

$$\begin{cases} a : 0.95 \leftarrow b \wedge e, \quad a : 0.94 \leftarrow b \wedge \overline{e}, \\ a : 0.29 \leftarrow \overline{b} \wedge e, \quad a : 0.001 \leftarrow \overline{b} \wedge \overline{e}. \end{cases}$$

One can then proceed as in the previous subsection and analyze this example. The details of such analysis are not given here since they are analogous, albeit admittedly more cumbersome.

5 Discussion and Future Work

This work introduces weight assignments using algebraic expressions from ASPs, with much to explore regarding their full expressive power, including recursion and logical variables. Bayesian Networks' theory and tools can be adapted, while connections to Markov Fields [14] and program selection applications are future work. The equivalence relation identifies subset cases, and better relations between SMs are possible. Inconsistent events' weights are set to 0, but inconsistencies from noise in observations may require reconsideration.

References

1 Weronika T Adrian, Mario Alviano, Francesco Calimeri, Bernardo Cuteri, Carmine Dodaro, Wolfgang Faber, Davide Fuscà, Nicola Leone, Marco Manna, Simona Perri, et al. The asp system dlv: advancements and applications. *KI-Künstliche Intelligenz*, 32:177–179, 2018. doi:10.1007/S13218-018-0533-0.

2 Marco Alberti, Elena Bellodi, Giuseppe Cota, Fabrizio Riguzzi, and Riccardo Zese. cplint on swish: Probabilistic logical inference with a web browser. *Intelligenza Artificiale*, 11(1):47–64, 2017. doi:10.3233/IA-170106.

3 Joaquín Arias, Manuel Carro, Zhuo Chen, and Gopal Gupta. Justifications for goal-directed constraint answer set programming. *arXiv preprint arXiv:2009.10238*, 2020.

4 Chitta Baral, Michael Gelfond, and Nelson Rushton. Probabilistic reasoning with Answer Sets. *Theory and Practice of Logic Programming*, 9(1):57–144, 2009. doi:10.1017/S1471068408003645.

5 Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca, and Torsten Schaub. Asp-core-2 input language format. *Theory and Practice of Logic Programming*, 20(2):294–309, 2020. doi:10.1017/S1471068419000450.

6 Fabio Gagliardi Cozman and Denis Deratani Mauá. The joy of probabilistic answer set programming: semantics, complexity, expressivity, inference. *International Journal of Approximate Reasoning*, 125:218–239, 2020. doi:10.1016/J.IJAR.2020.07.004.

7 Luc De Raedt, Angelika Kimmig, Hannu Toivonen, and M Veloso. Problog: A probabilistic Prolog and its application in link discovery. In *IJCAI 2007, Proceedings of the 20th international joint conference on artificial intelligence*, pages 2462–2467. IJCAI-INT JOINT CONF ARTIF INTELL, 2007.

8 Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2015. doi:10.1017/S1471068414000076.

9 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer set solving in practice*. Springer Nature, 2022.

10 Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124, 2011. doi:10.3233/AIC-2011-0491.

11 Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080. Cambridge, MA, 1988.

12 Werner Kießling, Helmut Thöne, and Ulrich Güntzer. Database support for problematic knowledge. In *Advances in Database Technology - EDBT'92: 3rd International Conference on Extending Database Technology Vienna, Austria, March 23–27, 1992 Proceedings 3*, pages 421–436. Springer, 1992. doi:10.1007/BFB0032446.

13 Michael Kifer and VS Subrahmanian. Theory of generalized annotated logic programming and its applications. *The Journal of Logic Programming*, 12(4):335–367, 1992. doi:10.1016/0743-1066(92)90007-P.

14 Ross Kindermann and J. Laurie Snell. *Markov random fields and their applications*, volume 1 of *Contemporary Mathematics*. American Mathematical Society, Providence, RI, 1980.

15 Joohyung Lee and Yi Wang. Weighted rules under the stable model semantics. In *Fifteenth international conference on the principles of knowledge representation and reasoning*, 2016.

16 Joohyung Lee and Zhun Yang. Lpmln, Weak Constraints, and P-log. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

17 Vladimir Lifschitz. Twelve definitions of a stable model. In *International Conference on Logic Programming*, pages 37–51. Springer, 2008. doi:10.1007/978-3-540-89982-2_8.

18 Kyle Marple, Elmer Salazar, and Gopal Gupta. Computing stable models of normal logic programs without grounding. *arXiv preprint arXiv:1709.00501*, 2017. arXiv:1709.00501.

19 Ilkka Niemelä and Patrik Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *Logic Programming And Nonmonotonic Reasoning: 4th International Conference, LPNMR'97 Dagstuhl Castle, Germany, July 28–31, 1997 Proceedings 4*, pages 420–429. Springer, 1997.

20 Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. The Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann, San Mateo, CA, 1988.

21 Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis lectures on artificial intelligence and machine learning*, 10(2):1–189, 2016.

22 Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62:107–136, 2006. doi:10.1007/S10994-006-5833-1.

23 Fabrizio Riguzzi. *Foundations of Probabilistic Logic Programming: Languages, Semantics, Inference and Learning*. River Publishers, New York, 1 edition, September 2022. doi:10.1201/9781003338192.

24 Fabrizio Riguzzi and Terrance Swift. Well–definedness and efficient inference for probabilistic logic programming under the distribution semantics. *Theory and practice of logic programming*, 13(2):279–302, 2013. doi:10.1017/S1471068411000664.

25 Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *International Conference on Logic Programming*, 1995.

26 Helmut Thöne, Ulrich Güntzer, and Werner Kießling. Increased robustness of bayesian networks through probability intervals. *International Journal of Approximate Reasoning*, 17(1):37–76, 1997. doi:10.1016/S0888-613X(96)00138-7.

27 Allen Van Gelder, Kenneth A Ross, and John S Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM (JACM)*, 38(3):619–649, 1991. doi:10.1145/116825.116838.

28 Victor Verreet, Vincent Derkinderen, Pedro Zuidberg Dos Martires, and Luc De Raedt. Inference and learning with model uncertainty in probabilistic logic programs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10060–10069, 2022. doi:10.1609/AAAI.V36I9.21245.

Vulnerability Detection Across Different CI/CD Platforms

Vasco Manuel Oliveira
Checkmarx, Braga, Portugal
ALGORITMI Research Centre/LASI, Dep. of Informatics, University of Minho, Braga, Portugal

Alberto Simões
Checkmarx, Braga, Portugal
2Ai – School of Technology, IPCA, Barcelos, Portugal
LASI – Associate Laboratory of Intelligent Systems, Guimarães, Portugal

Pedro Rangel Henriques
ALGORITMI Research Centre / LASI, Dep. of Informatics, University of Minho, Braga, Portugal

Abstract

In recent years, more attention has been paid to the security of the software supply chain (SSC). While at first SSC was just seen as the dependency of other libraries, today SSC is broader, considering all the environment where a software application is developed, from the actors, hardware and auxiliary tools.

This work focuses on a specific part of software supply chain security: the tools used for continuous integration and continuous deployment (CI/CD) and their vulnerabilities and risks. These tools are widely used by organizations to accelerate their software development, testing, and delivery, making any security issue present in these tools problematic for the organization. This is especially true given that most tools are open-source, making these tools the primary targets for exploits.

We will present a quick introduction to SSCS and CI/CD and provide a practical solution to detect risks and vulnerabilities in CI/CD tools, emphasizing the modular approach, allowing the system to easily scale to detect new risks and vulnerabilities, as well as to support new CI/CD tools.

2012 ACM Subject Classification Software and its engineering → Software verification and validation

Keywords and phrases Software Supply Chain, Software Supply Chain Security, CI/CD Platforms

Digital Object Identifier 10.4230/OASIcs.SLATE.2025.4

Funding *Alberto Simões*: This work has been partly supported by Fundação para a Ciência e Tecnologia, under framework of the Strategic Fundings UIDB/05549/2020, UIDP/05549/2020 and LASI-LA/P/0104/2020.
Pedro Rangel Henriques: This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Unit Project Scope UID/00319/Centro ALGORITMI (ALGORITMI/UM)

1 Introduction

The Software Supply Chain (SSC) includes the environment in which the software is developed. Focuses on the actors, the hardware, and the software used. This complex environment is susceptible to security risks, vulnerabilities, and attacks. Although detecting vulnerabilities in actors and hardware is almost impossible, new work has been done to protect the software used during the software development cycle. This is where this work fits, and all references to SSC Security (SSCS) during the remainder of this article are specific to the security of the software used during the development process.

The SSC consists of four different blocks [12]: the Source, the Build, the Dependencies, and the Deployment or Packaging phase. Continuous Integration and Continuous Deployent (CI/CD) tools can be used on most of them [3]: during the Source phase for code analysis,

during the Build phase for automating the compiling and unit testing, and during the Deployment or Packaging, to create deliverables and deploy the software for production. Examples of CI/CD tools are Github Actions [4] or Jenkins [7].

1.1 Motivation

CI/CD tools enable faster, more flexible, and diverse software delivery. However, they have also expanded the attack surface, giving attackers more ways to exploit vulnerabilities. Consequently, the number of attacks abusing flaws in the CI/CD ecosystem in the industry has been significantly increasing [13]. It is therefore essential to develop effective methods for identifying, understanding, and managing the risks inherent in these environments.

1.2 Objectives

This article seeks to investigate and document theoretical knowledge and present a practical solution in the field of CI/CD security, with an emphasis on enhancing their security in the context of Software Supply Chain Security.

Since this is a very recent topic and it still needs a lot of investigation, our first goal was to learn how to create a detection software that scans CI/CD tools configurations for risks and vulnerabilities, with an emphasis on acting with modularity, by seamlessly being able to implement rules for existing and newly found risks and vulnerabilities, and being easy to extend for new and different CI/CD tools.

2 Background and State of the Art

This section introduces the concept of Software Supply Chain and the relevance of its security. Follows a discussion on what are Continuious Integration/Continuous Deployment tools and their risks and vulnerabilities in the context of SSC.

2.1 Software Supply Chain

In general, a Supply Chain is a network of interconnected entities that work together to transform raw materials into finished products for consumers. This chain begins with some raw material extraction or production, proceeds through the manufacturing, transportation and distribution processes and concludes with retail sales to many consumers [11].

In computing, Software Supply Chain refers to all components, tools and processes involved in creating and deploying software products. These chains include code, configurations, open-source libraries, proprietary binaries, plugins, and dependencies, all coordinated through build tools and repositories. Each part of this interconnected chain brings functionality but also introduces vulnerabilities, as dependencies can propagate risks from one element to the entire system. The complexity and interdependencies inherent in these Supply Chains make them particularly vulnerable to attacks, with bad actors exploiting gaps in these chains, such as misconfigurations or outdated software, to infiltrate and compromise systems.

To manage those risks, organizations usually request a Software Bill of Materials, a detailed list of all third-party components in a project, enabling transparency and compliance with industry standards. However, using a Software Bill of Materials is not enough, with those vulnerabilities continuing to exist across infrastructure, software and codebase levels, as well as actors and processes involved. To strengthen the Software Supply Chain Security, organizations are increasingly adopting CI/CD pipelines, integrating automated scanning

and testing tools to detect misconfigurations, exposed secrets, and dependency vulnerabilities early in the development cycle. By implementing these security checks at every stage of the pipeline, teams can reduce exposure and respond swiftly to threats, helping to secure the software ecosystem against evolving attacks [6].

There are already some tools that provide security to the Software Supply Chain, such as Legit Security [9], a platform that protects the integrity of the SSC by automatically discovering and securing pipelines, infrastructure and other elements of the SSC. Another example is the OX Security platform [14], which offers an end-to-end approach to securing the software development lifecycle by providing visibility, risk analysis, and active protection across the entire pipeline. However, these tools are not open-source, which limits their scalability and adaptability, especially as new threats and attacks against the Software Supply Chain continue to emerge on a daily basis.

2.2 Continuous Integration/Continuous Deployment

Continuous Integration/Continuous Deployment, also known as CI/CD, is a software development methodology that, aligned with agile principles[1], aims to accelerate the software development lifecycle. Originating from the eXtreme Programming methodology[2] [1], CI/CD has become central to more agile software development.

Continuous Integration (CI)

CI/CD consists of two core practices, with the first practice being Continuous Integration or CI, which consists of frequent integrations along with automated testing of the code pushed by developers to a shared repository. It ensures that the committed code changes do not include any bugs, guaranteeing the stability and functionality of the codebase. This process includes various elements such as:

- **Automated Build Process:** Automating the application's build process, ensuring that the latest code changes are always in a deployable state.
- **Instant Feedback:** Developers get prompt feedback on their code changes, enabling them to quickly address any issues.
- **Enhanced Code Quality:** CI/CD pipelines often include static code analysis and security scanning to identify vulnerabilities.
- **Performance Monitoring:** Performance monitoring is implemented to ensure that recent code changes do not negatively impact the application's performance.

Continuous Delivery/Deployment (CD)

The second practice can be approached in two distinct ways, depending on whether the process is partially or fully automated, respectively:

- **Continuous Delivery:** involves ensuring that the application is always in a deployable state after passing the integration stage. The final deployment to production requires manual approval.
- **Continuous Deployment:** builds upon this by fully automating the process. Code changes that pass automated tests are deployed directly to production without human intervention. This approach enables faster and more frequent delivery of updates, ensuring consistent and reliable deployments.

[1] The agile principles are a set of guidelines that promote flexible and efficient software development.
[2] XP is a software development methodology intended to improve software quality and responsiveness to changing customer requirements.

These two practices together form a CI/CD pipeline, accelerating software delivery by automating and optimizing the processes of integrating, testing, and deploying code [17].[3]

CI/CD Tools

Thus, CI/CD tools allow for these two practices. This automation minimizes human intervention, reduces errors, and ensures faster delivery of high-quality software. CI/CD tools, such as GitHub Actions [4] or Jenkins [7], provide the infrastructure necessary to define and execute this kind of pipelines, allowing for the automation of these workflows [16, 17]. A CI/CD pipeline, or CI/CD workflow, is a sequence of automated steps designed to streamline the process of delivering new software versions [5].

2.3 CI/CD Risks and Vulnerabilities

To properly illustrate how CI/CD tools are susceptible to attacks, this subsection discusses a few examples of CI/CD security risks, based on the top-ten CI/CD security risks, published by OWASP [13]. Despite not being the most recent publication, it remains one of the most widely referenced resources for understanding possible vulnerabilities and risks in CI/CD ecosystems.

The main goal of this list is to help defenders identify focus areas for securing their CI/CD ecosystem, by providing risks that cover the full CI/CD pipeline lifecycle, from source code to production, and are based on real-world attacks and incidents. All of these risks include a clear description, real examples or use cases, consequences of exploitation and recommendations/mitigations. This was only made possible through extensive research into attack vectors associated with CI/CD, as well as the analysis of high-profile breaches and security flaws. As the information in this list is very extensive, only a few selected risks will be displayed in this article, but all of them can be viewed on the OWASP website[4].

Insufficient Flow Control Mechanisms

Attackers with access to CI/CD components (SCM[5], CI, artifact repository[6], etc.) can push malicious code or artifacts due to weak enforcement of approvals and reviews. CI/CD pipelines prioritize speed, often relying on automation with minimal human intervention.

Without proper flow control, an attacker could:

- Push malicious code that gets automatically deployed.
- Exploit weak branch protection or auto-merge rules.
- Upload tampered artifacts that are later deployed.
- Directly modify production code or infrastructure without verification.

To mitigate these vulnerabilities, the defenders should adopt the following strategies:

- Implement branch protection rules and restrict exclusions.
- Limit auto-merge usage and ensure strict validation.

---

[3] Besides this section, whenever the acronym CD is mentioned, it is referring to Continuous Deployment and not Continuous Delivery, unless it is stated otherwise.

[4] See `https://owasp.org/www-project-top-10-ci-cd-security-risks/`.

[5] tools and systems used to track and manage changes to source code, such as GitHub, GitLab and Bitbucket.

[6] An artifact repository is a system for storing, managing, and retrieving build artifacts (like compiled code, libraries, packages or container images), which are typically generated during the CI/CD pipeline process.

- Require additional approval before triggering production deployments.
- Prefer allowing artifacts that were created by a pre-approved CI service account to flow through the pipeline.
- Detect and correct drifts between production code and CI/CD sources.

Poisoned Pipeline Execution (PPE)

PPE is an attack where an adversary with access to a source control system, but not the build environment, modifies CI/CD pipeline configurations to execute malicious code during the build process.

PPE exploits permissions in a source control repository to inject malicious commands into a CI/CD pipeline. Pipelines that execute unreviewed code, from pull requests or commits to arbitrary branches, are particularly vulnerable. Once executed, the attack runs within the pipeline's context, potentially leading to severe security breaches. There are three types of PPE:

- **Direct PPE (D-PPE):** The attacker modifies the CI configuration file, directly in the repository, by doing a direct push to an unprotected branch or through a pull request. Once the pipeline is triggered, the malicious commands run in the build node.
- **Indirect PPE (I-PPE):** Occurs when direct modification of the CI configuration file is restricted. The attacker injects malicious code into referenced files (like Makefiles, scripts, test cases, or tool configurations) that the pipeline executes, and when the pipeline runs, these files execute their malicious instructions.
- **Public PPE (3PE):** Targets public repositories where anyone can submit pull requests. If the pipeline executes unreviewed contributions, an anonymous attacker can inject malicious code. This can expose internal assets, especially if public and private pipelines share the same CI environment.

Once the malicious code is executed, it runs within the pipeline's context and can carry out a variety of harmful operations, such as compromising the build environment, stealing sensitive data, or deploying further malicious code into production. To mitigate PPE risks, the next strategies should be adopted:

- Restrict write access to CI/CD configuration files.
- Enforce rigorous code reviews for changes that affect pipeline configurations.
- Separate CI configuration from source code when possible.
- Limit the exposure of files that can be referenced by the pipeline to trusted sources only.

Ungoverned Usage of 3rd Party Services

This risk associated with granting third-party services access to an organization's CI/CD systems without proper governance. It expands the attack surface due to the ease of providing third-party services access to sensitive resources.

Organizations often integrate third-party services into their CI/CD systems to increase functionality. However, the methods for connecting these services (like OAuth, SSH keys, or access tokens) are simple to implement, often without requiring additional permissions or approvals. This can result in third parties having extensive access, from reading code in a single repository to full administrative control. These services are also easily integrated into build pipelines, giving them access to resources that could expose the system to security risks.

Lack of governance and visibility can prevent organizations from maintaining proper Role-based access control and least privilege[7]. A single compromised third party could be used by attackers to manipulate code or trigger malicious actions in build systems, potentially affecting broader systems within the organization. To intercept those ungoverned third-party services, the following strategies should be adopted:

- Implement vetting procedures for third-party services before granting access to the environment, ensuring the principle of least privilege.
- Maintain visibility over third-party access, covering integration methods, granted permissions, and actual usage.
- Limit third-party access to only necessary resources and regularly review access controls.
- Periodically remove unused or unnecessary third-party services from the environment.

3 Security Frameworks

To effectively analyze and detect potential vulnerabilities in CI/CD components, the systems that will be developed rely on established security frameworks that facilitate pattern matching and standardized reporting. Two key technologies are used to support these capabilities: YARA-X, a high-performance pattern matching engine designed for security analysis, and SARIF (Static Analysis Results Interchange Format), a standardized format for reporting the results of static analysis tools.

3.1 YARA-X

YARA-X [18] is a widely used pattern matching tool designed for malware research and detection, and has the primary goal of enhancing the performance, security and usability compared with its predecessor, YARA, aiming to completely replace it in the future.

YARA-X enables users to define rules to identify malware families or other patterns of interest using textual and binary signatures. These rules consist of three main components:

- **Metadata:** Descriptive attributes of the rule, such as a description, threat level and status.
- **Strings:** A collection of textual or binary patterns used for detection.
- **Condition:** A boolean expression that, matched with the Strings components, determines whether the defined patterns match a given input.

Listing 1 presents an example of a rule made to detect a hypothetical malware family, Silent Banker.[8] This rule defines three different patterns (`$a`, `$b`, and `$c`) and triggers a match if any of them is found in the analyzed data.

3.2 YARA-X Dictionary Module

During the usage of YARA-X we faced the problem that it is designed for full text search, taking no advantage of structured information. Most tools use configuration files that are not flat. Their structure can be useful, making the process of matching specific keywords in specific fields easier.

---

[7] *Least Privilege* is a security concept that restricts users, applications, and systems to the minimum level of access necessary to perform their functions.

[8] Silent Banker is a monitoring trojan, type of malware that downloads disguised as a legitimate program, that captures screen shots and logs keystrokes.

Listing 1 YARA-X rule example, "Silent Banker".

```
rule silent_banker : banker {
    meta:
        description = "This is just an example"
        threat_level = 3
        in_the_wild = true

    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or $b or $c
}
```

YARA-X has built-in modules and the possibility for custom ones, which allows the user to add functions and different ways of retrieving data from the file provided.

Taking advantage of this extensibility mechanism, we developed the Dictionary module, specially created for loading structured file formats, like JSON or YAML and storing it in a dot notation, allowing access to keys by their path. The module stored the data in a structure called Dictionary, which consists of a list of key-value pairs. Listing 2 shows an example of such a configuration file, while Table 1 shows the stored data.

Listing 2 Github Actions pipeline example.

```
name: CI Pipeline
on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '14'
      - run: npm install
      - run: npm test
```

Beyond the structure, the module offers an API to facilitate access to specific fields:

- `full_key(key)`: Verifies if the given key exists in the dictionary. (e.g. `jobs.build.runs-on` would be true if that key exists);
- `is_key(key)`: Verifies if the given key is contained in one or more of the dictionary keys. (e.g. "jobs", "build", "runs-on", "jobs.build" and "build.runs-on" would all be true if `jobs.build.runs-on` exists);

Table 1 Dictionary structure derived from the example pipeline.

Key	Value
name	CI Pipeline
on.push.branches[0]	main
on.pull_request.branches[0]	main
jobs.build.runs-on	ubuntu-latest
jobs.build.steps[0].uses	actions/checkout@v4
jobs.build.steps[1].name	Set up Node.js
jobs.build.steps[1].uses	actions/setup-node@v3
jobs.build.steps[1].with.node-version	14
jobs.build.steps[2].run	npm install
jobs.build.steps[3].run	npm test

- `full_value(value)`: Verifies if the given value exists in the dictionary. (e.g. `actions/checkout@v4` would be true if that value exists in the dictionary);
- `is_value(value)`: Verifies if the given value exists in the dictionary, and if used with a wildcard "`*`" anything that is inside that character counts for the match. (e.g. `actions/checkout@*"` would be true for `actions/checkout@v4` or `actions/checkout@v2`);
- `get_value(key)`: Given a full key, this function returns the value for that key (e.g. the key "name" would return the value "CI Pipeline");
- `any_contains(key, value)`: Verifies if a value belongs to a specific key. If the key has an array (`[]`), the function returns true if any of the values of that array match the given value. If the wildcard "`*`" is used inside the value parameter, then everything that is inside that character will count as a match (e.g. the key `on.build.steps[].uses` would be true for the values `actions/checkout@*` and `actions/setup-node@*`);
- `in_key(key, string)`: Uses the methods `is_key` and `any_contains` to verify if a given string belongs to the key, in cases that the vulnerability would happen for a key or a value with the same string (e.g. for the key "name" it would be true for "CI Pipeline" even if that string was a key and not a value);
- `shows_first(key, key)`: Verifies if the first given key shows first on the dictionary to the second given key (e.g. would be always true if the first key was "name" and the second any other existing key).

3.3 SARIF Format

The Static Analysis Results Interchange Format (SARIF) is an industry standard format based in JSON for the output of static analysis tools [2]. Thus, to guarantee interoperability, the developed software should be able to export results in this format. An example of such a result in SARIF format is shown later in this article (see Listing 4).

4 Vulnerability Detection

The vulnerability detection in the CI/CD area can be divided in two smaller areas, since we can scan workflows from CI/CD tools (e.g. Github Actions pipeline) as well as plugins that belong to a CI/CD tool (e.g. Jenkins plugin). As will be explained in this section, these areas differ significantly and require different approaches, although they share some scanning methodologies.

4.1 CI/CD Workflows

Different CI/CD workflows, or CI/CD pipelines, can have different formats, rules and syntaxes that make them unique and distinguishable from each other. So how exactly can we detect these workflows in a modular way.

Workflow Formatter

CI/CD workflows can be defined in various formats, depending on the CI/CD tool in use, including YAML, XML, and JSON, with Jenkins being the only exception, using a unique Groovy format.

All of these formats can easily be converted into a dictionary object. By converting each workflow to a dictionary object, before scanning, we ensure consistent parsing across different tools.

Several libraries already support parsing YAML, XML and JSON files to dictionary objects (e.g., Python's `yaml`, `json`, and `xmltodict` libraries). While there is no third-party solution to convert Groovy based Jenkinfiles into dictionaries, it is still achievable using a simple custom parsing function.

Automatic Tool detector

It is also possible to detect automatically which CI/CD tool a specific workflow belongs to by scanning for certain patterns that are characteristic of that specific tool inside the given workflow.

For example, a workflow containing patterns like "jobs:", "steps:", "uses: actions/", "github.", and "runs-on" likely indicates a GitHub Actions pipeline.

By collecting multiple pattern sets for each tool and evaluating how many of them match a given workflow, we can calculate a confidence score and make an accurate tool prediction.

Workflow Scan

A workflow can be described as a dictionary, so instead of just using the simple YARA-X features for pattern matching, we can use the YARA-X's Dictionary Module (see Section 3.2) to scan vulnerabilities more accurately.

This is made by using the YARA-X Python API to compile rules and scan the workflows. By converting a rule file into a string and passing it to the YARA-X `compile()` method, we create a `Rules` object that contains those rules. This object provides a `scan()` method, which we use to scan the provided workflow by first converting the already formatted dictionary into a JSON string and encoding it in `utf-8`.

Listing 3 shows an example of how to use the Dictionary Module.

By using the Dictionary module, we can guarantee that there will be a Pull Request Target that is trying to run a command using a checkout action [10], all while verifying if the action checkout is being made before running the command.

Since different CI/CD tools use different rules and syntaxes, we generally can not use detection rules made for a tool to another, however it is the only thing that needs to be different, meaning that we can just make a detection rule file using YARA-X rules and the Dictionary module for each CI/CD tool, easily creating new rulesets for new CI/CD tools and adding rules to existing ones.

Listing 3 PR Target Rule with YARA-X and dictionary module.

```
import "dictionary"

rule check_pull_request_target {
    meta:
        name = "Pull request target"
        description = "Detects the use of pull_request_target trigger in GitHub
            Actions workflows, which can lead to security vulnerabilities specially
            when combined with an explicit PR checkout."
        impact = "May lead to malicious PR authors (i.e. attackers) being able to
            obtain repository write permissions or stealing repository secrets."
        recommendation = "Avoid using pull_request_target if the workflow doesn't
            need write repository permissions and doesn't use any repository
            secrets. Assign repository privileges only where needed explicitly
            through pull_request and workflow_run."
        reference = "https://securitylab.github.com/[...]-preventing-pwn-requests/"
        gravity = "high"
    condition:
        dictionary.in_key("on","pull_request_target") and
            dictionary.any_contains("jobs.build.steps[].uses","actions/checkout@*")
        and dictionary.is_key("jobs.build.steps[].run") and
            dictionary.shows_first("jobs.build.steps[].uses","jobs.build.steps[].run")
}
```

4.2 CI/CD Plugins

Only a few CI/CD tools, such as Jenkins and TeamCity, currently support plugins. However, the scanning module developed in this research to analyze such plugins can also be applied to plugins for other tools (e.g. VSCode or Visual Studio) or future CI/CD tools that implement plugin systems.

Store Results

Plugins are typically packaged as archive files (e.g. `.zip`, `.hpi`), so, to scan these plugins, we must first extract their contents to access the source code, however, this task can take a lot of time.

A good practice to circumvent this problem is to store the results of the already scanned plugins in a database, ensuring that a plugin only needs to be decompiled once.

Given that plugins can suffer changes over time due to updates or even tampering, we must store not just the plugins, but their versions (in case of an update) and hashes (in case of tampering) as well.

Plugin's Scan

Plugins will also be scanned using YARA-X, though not with the Dictionary module, as plugin code does not follow a unified structure like CI/CD workflows.

With YARA-X we can make rules with additional information via metadata, which is going to be used to get the rule's description, author and version, as well as capture patterns within a file. With the YARA-X pattern detection, the pattern that was detected gives the match content, offset and length, and with the offset we can easily get in which line the pattern was detected as well.

This is also made by using the YARA-X Python API to compile rules and scan plugin files.

5 Security Services

Our research team built an API for each different CI/CD vulnerability detection area according to the discussion presented in the previous section. These APIs, that will be described in this section, serve as an easy way to interact with the scanning engines, allowing external tools or users to submit configurations or plugin data for analysis and receive structured vulnerability reports in return.

Both APIs that we developed were implemented using the FastAPI [15] framework, which provides high performance, automatic documentation, and easy integration with Python based tools and services.

5.1 Workflow Scan API

This is an API designed to scan vulnerabilities in CI/CD configuration files, in a modular way. It uses YARA-X and the Dictionary module, specially made for this API, to scan workflows more easily and more accurately. After analyzing a workflow, the API returns the scan results in the standardized SARIF or JSON format. This API was tested using the default configuration workflows of each CI/CD tool.

With modularity as its core objective, this API was developed with the following priorities in mind:

- Support for scanning multiple CI/CD configuration files from different CI/CD platforms;
- The ability to easily add new rules or rulesets to refine and expand detection capabilities across platforms;
- Converting any workflow into a unified dictionary object format to facilitate scanning;
- Automatically recognizing, with high accuracy, which CI/CD platform the configuration belongs to.

API Endpoints

The API provides the following endpoints:

`POST /scan` – Scans a given CI/CD configuration file for vulnerabilities.

- **Parameters**:
  - `file` – The configuration file to be scanned (uploaded via form-data).
  - `rule_type` – (Optional) The name of the CI/CD platform the configuration belongs to (e.g., `github_actions`, `jenkins`, `circleCI`).
  - `format` – (Optional) The format that the results will be displayed on. Supported values are `sarif` (default) or `json`.
  - `automatic` – (Optional) Boolean flag to enable automatic detection of the CI/CD platform. When set to `true`, the `rule_type` parameter is not required.
- **Returns**: The scan results in SARIF or JSON format. Listing 4 shows an example of a result in SARIF.

`POST /add_rule` – Adds a new YARA-X rule to an existing ruleset.

- **Parameters**:
  - `file` – A `.txt`, `.yar`, or `.yara` file containing the new rule (uploaded via form-data).
  - `rule_type` – The name of the existing ruleset to which the rule should be added (e.g., `github_actions`, `jenkins`, `circleCI`).
- **Returns**: A message confirming the successful addition of the rule, or an error if the rule is invalid or already exists, or the `rule_set` does not exist.

`POST /add_ruleset` – Creates a new ruleset for a CI/CD platform.

- **Parameters**:
  - `file` – A `.txt`, `.yar`, or `.yara` file containing a complete ruleset (uploaded via form-data).
  - `rule_type` – The name for the new ruleset.
- **Returns**: A message confirming the creation of the ruleset or an error if the file is invalid or a ruleset with that name already exists.

Listing 4 Example of a Result given in SARIF format of scanning a workflow that uses a PR Target.

```
{"version": "2.1.0",
  "$schema": "https://json.schemastore.org/sarif-2.1.0.json",
  "runs": [{
     "tool": {
       "driver": {
         "name": "SecureChain - CI/CD",
         "version": "1.0"}},
     "invocations": [{
         "startTimeUtc": "2025-04-23T13:43:42Z",
         "endTimeUtc": "2025-04-23T13:43:42Z",
         "executionSuccessful": true}],
     "results": [{
         "ruleId": "check_pull_request_target",
         "level": "warning",
         "message": {
           "text": (Description),
           "markdown": (Markdown with Metadata),},
         "locations": [{
            "physicalLocation": {
              "artifactLocation": {
                "uri": "pull_request_target.yml"
              }}}]}]}
```

5.2 Plugin Scan API

This API is designed to scan any type of plugins, identifying possible risks and vulnerabilities within those plugins; however, currently it is only able to scan Jenkins plugins. It uses YARA-X to detect these risks and vulnerabilities inside these plugins, supporting not just literal string matching but more string matching mechanisms, such as hexadecimal strings, regular expressions, and many more features. This API was tested using 117 randomly selected publicly available plugins in the Jenkins Marketplace [8].

Just like the Workflow Scan API, this API has the same core objective: "modularity"; and so, creating the rules is as easy and as modular as the previous API. However, when it comes to the file extraction engine itself, it depends a lot on what tool the plugin that is being scanned belongs to, so it is needed to create a special detector for each tool.

API Endpoints

The API provides the following endpoints:

`POST /plugin_file` – Scans a single plugin file for potential risks and vulnerabilities.

- **Parameters**:
  - `file` – The plugin file to be scanned (e.g., a `.hpi` file), uploaded via form-data.
  - `cfr` – (Optional) The decompiler to use during analysis. Supported values are `cfr` (default) or `jadx`.
  - `format` – (Optional) The format that the results will be displayed on. Supported values are `sarif` (default) or `json`.
- **Returns**: A JSON or SARIF file with the detected plugins and plugin files matches, with the YARA rules and their metadata. If no issues are found, a message indicating the absence of matches is returned. Listing 5 shows an example of a result in JSON.

`POST /plugin_list` – Scans a list of plugins provided in a single file for batch analysis.

- **Parameters**:
  - `file` – A JSON file containing multiple plugins to be scanned.
  - `cfr` – (Optional) The decompiler to use during analysis. Supported values are `cfr` (default) or `jadx`.
- **Returns**: A JSON or SARIF file containing scan results for each plugin. Each entry includes rule matches or a message indicating that no matches were found.

`POST /plugin_url` – Downloads and scans a plugin from a given URL.

- **Parameters**:
  - `url` – The URL from which to download the plugin.
  - `cfr` – (Optional) The decompiler to use during analysis. Supported values are `cfr` (default) or `jadx`.
- **Returns**: A JSON or SARIF file with the YARA rule matches if any are detected. Otherwise, a message stating that no matches were found is returned.

Listing 5 Example of a Result given in JSON format of scanning a plugin.

```
{"results": {
   "malicious-plugin-test": {
     "ExamplePlugin.java": {
       "suspicious_commands": {
         "category": "Commands",
         "metadata": [[
             "description",
             "Detects suspicious system commands often used in reconnaissance
                 or post-exploitation phases"],
           ["author",
             "Vasco Oliveira <vasco.oliveira@checkmarx.com>"],
           ["version",
             "1.0"]],
         "matches": [[
             "uname -a",
             21]
         ]}}}}}
```

6 Conclusion

This article serves as an introduction to Software Supply Chain Security and Continuous Integration/Continuous Deployment. It discusses how to develop scanning software that can scan CI/CD tools and their components in a modular way. The scanners and the APIs we provided were designed in a way to ensure that future improvements can be seamlessly integrated, requiring only the addition of new rules to detect known and new risks and vulnerabilities. While this publication focuses on the system itself, future publications will point out the risks and vulnerabilities to which these systems are susceptible. No empirical evaluation or performance benchmarks were included, since such analysis was outside the scope of this work.

The software is not yet publicly available but will be disclosed as open-source and published on Checkmarx's GitHub repository.

For future work, there is still much to be done, with the next step being to scan reusable components of CI/CD tools, such as actions for GitHub Actions and orbs in CircleCI. Since this work focuses on the framework for vulnerability and risk detection, the final step will be the responsibility of the application security team, who will develop the YARA-X rules used to identify risks in CI/CD components.

References

1 Kent Beck. *Extreme Programming Explained: Embrace Change*. XP series. Addison-Wesley Professional, illustrated edition, 2000.

2 Microsoft Corporation. SARIF. `https://sarifweb.azurewebsites.net/`.

3 Clint Gibler and Francis Odum. An Overview of Software Supply Chain Security. `https://tldrsec.com/p/supply-chain-security-overview`, 2023.

4 GitHub. GitHub actions, 2024. URL: `https://github.com/features/actions`.

5 Red Hat. What is a CI/CD pipeline? `https://www.redhat.com/en/topics/devops/what-cicd-pipeline`, 2022.

6 Senior Technical Content Marketing Manager Jacob Schmitt. Software supply chain: What it is and how to keep it secure. `https://circleci.com/blog/secure-software-supply-chain/#what-is-the-software-supply-chain`, 2023.

7 Jenkins. Jenkins, 2024. URL: `https://www.jenkins.io/`.

8 Jenkins. Jenkins plugins index. `https://plugins.jenkins.io/`, 2025.

9 Legit Security. Software supply chain security, 2025. URL: `https://www.legitsecurity.com/software-supply-chain-security`.

10 Jaroslav Lobačevski. Keeping your GitHub Actions and workflows secure: Preventing pwn requests. `https://securitylab.github.com/resources/github-actions-preventing-pwn-requests/`, 2021.

11 McKinsey & Company. What is supply chain? `https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-supply-chain`, 2023.

12 Open Source Security Foundation and Google. Supply-chain Levels for Software Artifacts (SLSA). `https://slsa.dev`, 2021. Accessed: 2025-04-28.

13 Foundation OWASP. OWASP Top 10 CI/CD Security Risks. `https://owasp.org/www-project-top-10-ci-cd-security-risks/#`, 2022.

14 OX Security. Ox security, 2025. URL: `https://www.ox.security/`.

15 Sebastián Ramírez. Fastapi. `https://fastapi.tiangolo.com/`. Accessed: 2025-04-21.

16 Pooya Rostami Mazrae, Tom Mens, Mehdi Golzadeh, and Alexandre Decan. On the usage, co-usage and migration of ci/cd tools: A qualitative analysis. *Empirical Software Engineering*, 28(52), 2023. `doi:10.1007/s10664-022-10285-5`.

17 Bhaskara Sahithi Shanmukhi. Implementing and using ci/cd: Addressing key challenges faced by software developers. *International Journal of Scientific Research in Engineering and Management (IJSREM)*, 08(08), 2024.

18 VirusTotal. YARA-X. `https://virustotal.github.io/yara-x/`, 2024.

CVTool: Automating Content Variants of CVs

Julio Beites Gonçalves
ALGORITMI Research Centre/LASI – DI, University of Minho, Braga, Portugal

Maria João Varanda Pereira
Research Centre in Digitalization and Intelligent Robotics (CeDRI),
Laboratório Associado para a Sustentabilidade e Tecnologia em Regiões de Montanha (SusTEC),
Instituto Politécnico de Bragança, Portugal

Pedro Rangel Henriques
ALGORITMI Research Centre/LASI – DI, University of Minho, Braga, Portugal

Abstract

As academic professionals, we frequently need to create different versions of our CVs for project applications, career evaluations, or competitions. These versions may be chronologically structured or skill-oriented, covering specific periods and written in various languages. Even when using a LaTeX document as a base, numerous modifications are required each time an updated CV version is requested for a specific purpose.
The primary objective of the project reported in this paper is to design and implement a web-based system (CVTool) that simplifies the management of LaTeX CV content while ensuring flexibility. The CVTool is built on a domain-specific language that enables the creation of various filters, allowing for the automatic content adjustment while preserving the original format. Information is extracted from the LaTeX document, and users can specify the sections, dates, skills they want to highlight, and the language in which the CV should be generated. Since the approach relies on an internal data representation derived from the original LaTeX document, it offers users the flexibility to manage content efficiently and extract the necessary information with ease.

2012 ACM Subject Classification Software and its engineering → Domain specific languages; Software and its engineering → Parsers

Keywords and phrases Latex CV, CV Versioning, DSL, CV Parsing, CV Templates

Digital Object Identifier 10.4230/OASIcs.SLATE.2025.5

Funding This work has been supported by FCT -– Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.
Maria João Varanda Pereira: The work of Maria João was supported by national funds through UID/05757 – Research Centre in Digitalization and Intelligent Robotics (CeDRI); and SusTEC, LA/P/0007/2020 (DOI: 10.54499/LA/P/0007/2020).

1 Introduction

The use of markup languages, especially LaTeX, to specify curricula vitae allows the structural organisation of information relating to an individual's academic, teaching, and professional activities. For many, it is a way to easily maintain data in text format without worrying too much about the formatting or visual aspect of the document [2]. In addition, the titles of the sections, used to organise the CV information, can be seen as "tags" utilised to annotate the different parts, allowing for searching and filtering. This point of view was taken into account to plan a solution to overcome the need to customise the CV (i.e., generate specific versions), while maintaining the original document as a repository. This paper introduces CVTool and discusses the design of its filtering DSL as well as its implementation.

The biggest challenge is that these filters are not only based on the LaTeX commands used in the document, which allow identifying parts and subparts and their titles, but also the chronological information that must appear transversally in all items of the different

sections. The same kind of filter should be specified when a theme filter based on keywords must be applied. This type of filter is essential when we need a tailored CV related to a certain period or subject. Furthermore, it is also possible to define the language in which the CV should be written. The result will then be a CV that maintains the original format but is written in the desired language and contains information relating to the desired sections and dates. It is understood that when applying these filters, there must be a concern to maintain the consistency of the CV structure.

While user-friendly platforms like LinkedIn or Europass simplify CV creation through predefined templates [3, 6], they often lack flexibility for deeper customisation. In contrast, LaTeX offers full control over structure and layout but requires technical expertise [7]. This gap between ease of use and adaptability highlights the need for a solution like CVTool, which combines LaTeX's flexibility with dynamic content management capabilities. After this introduction, the article is divided into 5 more sections: Section 2 presents the created DSL; Section 3 explores the challenges related to Latex syntax when building the document parser; Section 4 presents the system architecture components; Section 5 is dedicated to an use case for demonstrating the tool; Section 6 concludes the paper by presenting details regarding the website deployment and outlining potential directions for future work.

2 LaTeX Syntax Analysis: Challenges and Considerations

This section examines the primary challenges encountered when parsing LaTeX documents and managing their content, particularly in the context of filtering, customising, and generating new versions of CV files. LaTeX typesetting language presents unique difficulties due to its mixture of structural commands, metadata, and content, which must be accurately processed while ensuring document integrity and consistency.

2.1 Overview of LaTeX Syntax

A LaTeX document typically consists of two main components: commands and content. Commands, which are prefixed with a backslash (e.g., *\section*, *\begin*, *\textbf*), define the structure and formatting of the document. Content elements, on the other hand, consist of plain text, labels, equations, figures, tables, and other elements that form the body of the document. In specific cases, these content elements can also refer to structural meanings.

A typical LaTeX file follows a structured syntax, starting with a preamble where document settings and packages are defined. The document body begins with the *\begin{document}* command and ends with the *\end{document}* command. Within the document body, various structural elements, such as sections, subsections and paragraphs, are defined using LaTeX commands. Additionally, LaTeX allows for complex elements like mathematical symbols, references, and bibliographies, which are handled with specific commands and environments.

In the context of CV management, the LaTeX file contains not only the content that appears in the rendered document but also specific LaTeX commands that define its structure, style, and formatting. Handling content and commands accurately, preserving the document integrity and the user expectations, introduces a hard contextual analysis to be implemented by the parser.

2.2 Analysing LaTeX Structural Commands

A key challenge in processing LaTeX documents is distinguishing between LaTeX commands that refer to structural elements and those that represent actual text content. Despite having a similar syntax, these commands serve distinct purposes and must be handled accordingly based on their context.

For example, both *\begin{verbatim}* and *\section{Work experience}* consist of a command followed by a text element. In the case of *\begin{verbatim}* command, "verbatim" is a LaTeX keyword that instructs the compiler to treat the enclosed content as raw text, without any formatting or processing. As such, it should not be altered in a translation filter. On the other hand, *\section{Work experience}* contains "Work experience" as the text element, which is part of the document content and may need to be treated.

This was solved by properly identifying the LaTeX grammar productions and associating the different processing actions.

2.3 Analyzing Date Elements

Applying a date filter to a LaTeX document presents a considerable challenge: precisely identifying numeric values that correspond to dates. Dates can appear in various formats, such as "1996," "1996/1998," or even detailed forms like "2016.Jul.12." Complicating matters further, these same numeric values might also serve other purposes, such as reference pages, or chapters of an article (e.g., "96/98"), depending on the context in which they are used. For instance, consider the following CV document excerpt:

```
\namedItem[Publication]{
    JNICT, PBIC/TIT/2090/2;
    \textsf{UCM Doctoral Program in Informatics}, UCM/Madrid, 1998.
}
```

In this example, "1998" represents a date range, while "2090" is a reference identifier, but both tokens follow the same number format (*dddd*). Differentiating between these two similar elements requires an in-depth understanding of the context in which they appear, highlighting the need for a robust approach capable of analysing the context of numeric values within a document and correctly handling them.

Given the vast variety of formats in which dates can be written, covering all possible cases is impractical and prone to error, as new formats or user input styles may emerge. Additionally, determining whether a numeric value represents a date or something else, such as a serial number, requires precise contextual interpretation.

To address this challenge effectively and preserve LaTeX's features, users should adopt a standardised approach for specifying date ranges. Specifically, users should encapsulate date elements within a custom command:

```
\daterange{date element}
```

This command must be defined in the document's preamble using the following directive:

```
\newcommand{\daterange}[1]{#1}
```

Rather than limiting users to a fixed date format, this approach allows them to input dates in any preferred style. This is done by adding more productions to the LaTeX grammar representing the date entries.

2.4 Handling LaTeX Metadata and Ensuring Document Integrity

When working with LaTeX documents, maintaining the integrity of the file structure is paramount. LaTeX relies on specific metadata commands and structural elements to ensure successful compilation and correct rendering. These commands, such as *\documentclass*, *\begin{document}*, and *\end{document}*, are essential components of the document's framework. Improper handling or modification of these elements can lead to structural issues.

A valid LaTeX document requires balanced and properly ordered commands. For example, every *\begin{...}* command must have a corresponding and correctly placed *\end{...}*. Similarly, if a *{* is introduced in the document, it must always have a matching *}*. Failure to maintain this balance – whether due to accidental omission, incorrect placement, or other errors – can cause compilation failures or lead to unexpected rendering issues, breaking the document's structure.

Preserving the correct order of elements and commands is crucial to preserving document integrity. Commands in the preamble, such as *\usepackage{...}* or *\author{...}*, must appear before the *\begin{document}* command to configure the document correctly. Altering this sequence can disrupt the LaTeX interpreter, causing errors or rendering issues.

Additionally, it is crucial to differentiate between metadata commands that pertain to the document's settings and those containing user-generated content. Commands that define structural metadata, such as *\usepackage{...}*, should remain unchanged to preserve the document's integrity. However, commands like *\ecvnationality{...}*, which include user-specific content, require careful handling to ensure accuracy without compromising the overall structure.

Attending to the challenges presented, the system must incorporate mechanisms capable of enforcing these principles, ensuring balanced command pairs, maintaining the correct order of elements, and accurately distinguishing between metadata and user content. Once again, these mechanisms are based on the LaTeX grammar structure and the semantic actions associated with them.

2.5 Contextual Interpretation of LaTeX Data Elements

Processing LaTeX documents involves more than identifying individual commands and content; it also requires understanding how certain elements depend on their context for proper interpretation. In many cases, specific data items within a document only make sense when grouped with other related elements.

For example, consider the following LaTeX snippet:

```
\namedItem[UCM]{
\emph{Languages, Ontologies, and Automatic Grammar Generation},
\textsf{UCM Doctoral Program in Informatics},
UCM/Madrid, 2016.Jul.12.
}
```

If a filter matches the token *UCM Doctoral Program in Informatics, UCM/Madrid, 2016.Jul.12*, it is critical to manage the entire *\namedItem* block it belongs to. Deleting only the matched token would result in a fragment like this:

```
\namedItem[UCM]{
\emph{Languages, Ontologies, and Automatic Grammar Generation},
}
```

This incomplete structure not only fails to accurately represent the original data but also introduces the risk of errors in the document, such as rendering issues or compilation failures. Proper handling requires recognising that the matched token is part of a larger unit, which should be treated holistically during operations such as deletion, filtering, or transformation. To address this challenge, the LaTeX grammar productions and their corresponding processing actions accurately establish the relationships between tokens and embed contextual metadata within each token's internal representation to enable precise subsequent processing.

3 System Components

The system architecture presented in Figure 1 is composed of modular components designed for the efficient parsing, transformation, and generation of LaTeX-based CV documents.

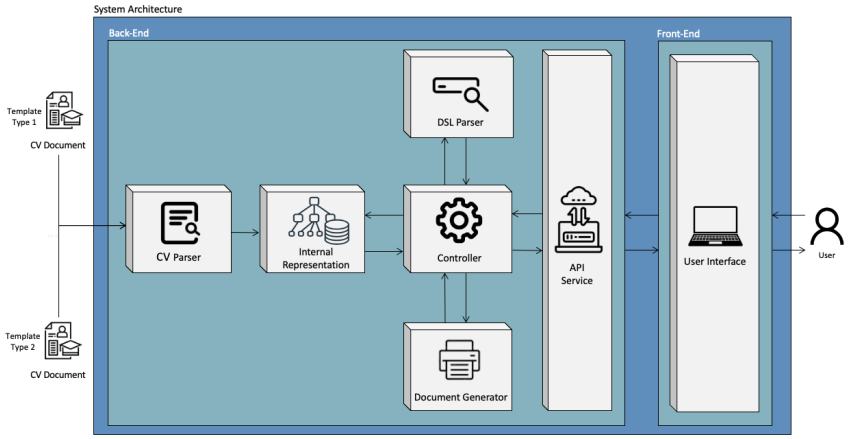


Figure 1 System Architecture.

At its core, the CV Parser component extracts and structures data from the given LaTeX template, which is then stored temporarily in the Internal Representation – a transient in-memory data structure that ensures consistency and fast access during processing. For that, the LaTeX context-free grammar is used to construct the CV Parser together with the *Earley* analysis algorithm and *MyInterpreter* interface provided by Lark. These two components have distinct roles in handling the document's structure: the Earley parser provides the structural foundation by analysing the LaTeX grammar and building a tree that captures the document's hierarchy and content. MyInterpreter allows for efficient processing of each LaTeX element by visiting the nodes generated by the Earley parser and converting them into a structured format that will be used as the Internal Representation. The parser manipulates the matched tokens from the parse tree and organises them into four distinct types of entries – command, element, LB, and RB – as defined in the grammar. Each token is processed and converted into a Python data structure named *Dictionary*, preserving essential information such as the content, hierarchical level, and contextual placement in the document. The parser constructs a structured and coherent internal representation by organising the parsed tokens into well-defined entries. This approach ensures consistent data handling across different templates, which is essential for uniform processing and accurate output generation in later stages. Each entry within the internal representation follows the structure outlined below:

- **Id**: A unique identifier that reflects the order of the elements within the document;
- **Tipo**: The entry type, such as “command”, “element”, “RB”, or “LB”;
- **Valor**: The actual content of the token;
- **Nivel**: The nesting level of the entry, indicating its position within nested structures;
- **Section**: The section of the document to which the entry is related, or empty if not applicable;
- **Subsection**: The subsection of the document to which the entry is related, or empty if not applicable;
- **Reserved**: A flag indicating whether the token is a reserved LaTeX token.

Following this structure, a complete LaTeX document is represented as a list of dictionaries within our internal representation.

To provide a clearer understanding of how the internal representation works, below is presented a small excerpt of LaTeX data and its corresponding structure after being parsed into the internal representation. This comparison illustrates how each LaTeX element is captured and transformed into its standardized dictionary format.

Listing 1 LaTeX example data.

```
\begin{europecv}
\section{Work experience}
\namedItem[Full Professor]{
 Dates: since 2010
}
```

The corresponding internal representation of this LaTeX example can be defined as the following list of dictionaries:

Table 1 Internal Representation of the LaTeX Code from Listing 1.

ID	Tipo	Valor	Nivel	Section	Subsection	Reserved
1	command	begin	0			
2	LB	{	1			
3	element	europecv	1			True
4	RB	}	0			
5	command	section	0	Work experience		
6	LB	{	1	Work experience		
7	element	Work experience	1	Work experience		False
8	RB	}	0	Work experience		
9	command	namedItem	0	Work experience		
10	element	[Full Professor]	0	Work experience		False
11	LB	{	1	Work experience		
12	element	Dates: since 2010	1	Work experience		False
13	RB	}	0	Work experience		

Once the internal representation is constructed, the DSL Parser takes over by interpreting user-defined DSL queries. It validates their syntax and translates them into executable commands. These commands are handled by the Controller component, which orchestrates the flow of data, manages the system logic, and coordinates interactions with other modules [5]. The Document Generator component receives the processed data and converts it into valid LaTeX files, preserving the intended formatting and structure. The API Service acts as

the communication layer between the back end and the front end, managing requests and responses. Finally, the User Interface enables users to input requirements, interact with the system, and retrieve generated documents, ensuring a seamless and user-friendly experience. For this project, the Back-End was developed using Python, leveraging its extensive ecosystem of libraries and frameworks in the field of Natural Language Processing. The Front-End was implemented using React, a widely adopted JavaScript library for building user interfaces.

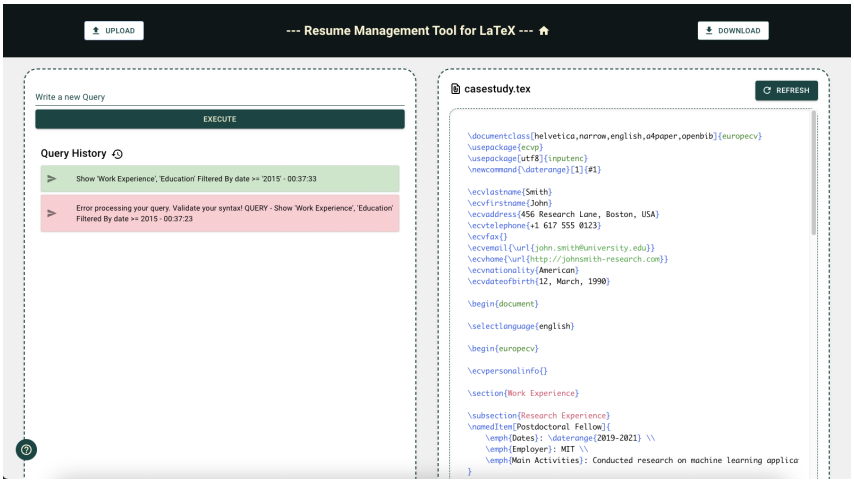


Figure 2 Tool Page.

The tool page (Figure 2) is the central workspace of the system, providing an interactive interface for users to customise and manage their LaTeX-based CV documents. This page is divided into two main sections: The file display panel on the right, where the uploaded LaTeX file is shown, and the query interface on the left, where users input commands to filter and manipulate the CV content. Upon uploading a LaTeX file, users can utilise the query input area to introduce specific commands, which are then processed by the system. Each query is added to a stack in the Query History, where the system provides colour-coded feedback: successful queries are highlighted in green, while failed ones appear in red, accompanied by an error message detailing the issue. This immediate feedback mechanism enables users to iteratively refine their CVs, with each validated query reflecting real-time updates in the displayed file. Once users are satisfied with the final version, they can download the customised LaTeX file directly from the page.

4 DSL for LaTeX Curriculum Content Management

Domain Specific Languages (DSLs) are specialized programming languages designed to address the needs of a specific domain [4], offering abstractions and syntax tailored to that area. Compared to General Purpose Languages (GPLs) [8], which are designed for a wide variety of applications, DSLs focus on expressiveness and efficiency in handling specific tasks related to a particular area of expertise, reducing complexity and improving productivity [10]. Within this, DSLs play a critical role in bridging the gap between technical systems and domain-specific expertise, allowing non-programmers to engage with software systems in ways that are intuitive and directly aligned with their expertise.

The DSL developed for managing curriculum content in LaTeX provides users with four primary operations, **“Show”**, **“Translate”**, **“Reorder”**, and **“Drop”**. Each one serves a different purpose, yet still follows defined rules. Below, we present the available command structure.

SHOW *Sections List* **FILTERED BY** *Conditions*
TRANSLATE FROM *Language* **TO** *Language*
REORDER *Sections List*
DROP *Sections List*

4.1 DSL Grammar

To ensure accurate interpretation of user queries, a context-free grammar was developed to define the structure of the Domain-Specific Language. This grammar enables a clear mapping between natural user inputs and the corresponding operations within the LaTeX CV management system. The choice of a context-free grammar is grounded in established compiler construction techniques, where such grammars are fundamental to formally specifying the syntax of programming languages [1]. It supports four main query types as described before – selection, translation, reordering, and deletion – defined by dedicated production rules.

Listing 2 Excerpt of the DSL Grammar Rules.

```
query: selectquery
     | translatequery
     | reorder_query
     | drop_query

selectquery: "SHOW"i sectionlist whereclause?

translatequery: "TRANSLATE"i "FROM"i source "TO"i output

reorder_query: "REORDER"i sectionlist

drop_query: "DROP"i sectionlistdrop

sectionlist: ALL
           | section (VIRG section)*

sectionlistdrop: section (VIRG section)*

whereclause: "FILTERED BY"i condition

andcondition: "(" condition ")"
            | simplecondition ("AND"i simplecondition)*

condition: andcondition ("OR"i andcondition)*

simplecondition: SECTION sectionoperator value
               | SUBSECTION sectionoperator value
               | DATE dateoperator value
               | THEME sectionoperator value
```

The grammar is written using a simplified EBNF-like notation, commonly used for formally specifying the syntax of context-free languages. Rules are defined with a colon (:), alternatives are separated by a vertical bar (|), optional elements are marked with a question mark (?), and repetition is indicated by an asterisk (*). Terminal strings use quotation marks, and the suffix `i` allows case-insensitive matching to make user input more flexible.

Using the Python library Lark [9] that provides a Parser Generator, the input grammar listed above is transformed into an Earley parser. Specifically, the MyInterpreter interface traverses the parse tree and transforms matched tokens into a structured dictionary format, preserving the information and additional contextual details. This structure serves as a foundation for further processing stages and execution of the user's intent, ensuring a flexible yet robust interaction model between the user and the system.

4.2 Query Examples and Expected Outputs

Consider the following examples to demonstrate the practical application of these commands and to illustrate to the user the versatility and usability of each query. These examples showcase how users can interact with and customise their CV files, highlighting the system's flexibility.

4.2.1 Show Command

- **Selecting Specific Sections:**

```
Show 'Work Experience', 'Education'
```

Expected Output: Returns a file only with the "Work Experience" and "Education" sections.

- **Selecting All Sections:**

```
Show *
```

Expected Output: Returns a file with all sections. This should generate the same document.

- **Filtering Sections by Date :**

```
Show * Filtered By Section = 'Education' and Date >= '2010'
```

Expected Output: Returns the entire file, but filters the "Education" Section to include only entries from 2010 onward.

- **Filtering a Section by One Date and a Subsection by a Different Date :**

```
Show *
        Filtered By (Section = 'Education' and Date > '2010')
        OR (Subsection = 'Work Experience' and Date = '2010')
```

Expected Output: Returns the entire file, filtering the specified section and subsection according to the given dates. Note: The specified subsection is not associated with the given section.

- **Filtering the document by Date, Except for a Specified Section:**

```
Show *
        Filtered By Section != 'Education' and Date > '2010'
```

Expected Output: Filter the entire document by the specified date, except for the specified section. Note: The condition *Date > '2010'* is applied to all sections except *"Education"*, which is excluded from the filter. The priority is given to excluding the "Education" section before applying the date filter to the rest.

4.2.2 Translate Command

- **Translating a CV:**

```
Translate From 'fr' To 'en'
```

Expected Output: Translates the document from French to English.

The translation functionality is implemented using the *GoogleTranslator* class from the *deep_translator* Python library. This library facilitates text translation by interfacing with the Google Translate service, allowing for automatic conversion between a wide range of languages.

4.2.3 Reorder Command

- **Reordering the file sections**

```
Reorder 'Education', 'Work Experience', 'Published Work'
```

Expected Output: Retrieves a file containing only the specified sections displayed in the order defined by the user.

4.2.4 Drop Command

- **Deleting sections**

```
Drop 'Education', 'Published Work'
```

Expected Output: Retrieves the file without the specified sections or subsections.

5 Case Study

This section presents a case study to illustrate the effect of a sequence of commands on a specific CV. Dr. John Smith, a researcher in Natural Language Processing (NLP), is applying for a position in a university's Language Engineering department. To support his application, he must tailor his CV to meet the institution's specific criteria. The original CV is shown below and includes various sections, not relevant to the target application.

EUROPEAN CURRICULUM VITAE FORMAT

PERSONAL INFORMATION

Name	**John SMITH**
Address	**456 Research Lane, Boston, USA**
Telephone	**+1 617 555 0123**
Email	john.smith@university.edu
Homepage	http://johnsmith-research.com
Nationality	American
Date of birth	12, March, 1990

WORK EXPERIENCE

RESEARCH EXPERIENCE

- Postdoctoral Fellow
  Dates: 2019-2021
  Employer: MIT
  Main Activities: Conducted research on machine learning applications in robotics.

- Research Assistant
  Dates: 2015-2019
  Employer: Harvard University
  Main Activities: Developed algorithms for natural language processing tasks.

TEACHING EXPERIENCE

- Teaching Assistant
  Dates: 2015-2018
  Employer: Harvard University
  Main Activities: Assisted in teaching courses on algorithms and data structures.

VOLUNTEER WORK

- STEM Outreach Volunteer
  Dates: 2014-2020
  Organization: Local Schools
  Activities: Conducted coding workshops for high school students.

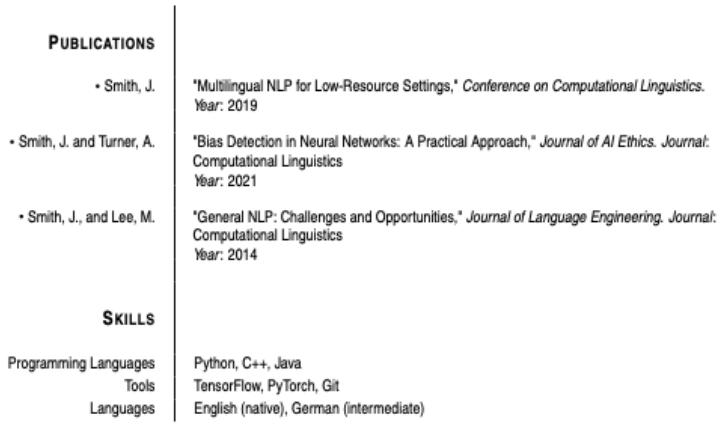
EDUCATION

- Ph.D.
  Year: 2019
  Institution: Harvard University
  Degree: Computer Science

- B.Sc.
  Year: 2014
  Institution: University of California
  Degree: Computer Science

Page 1 - Curriculum vitæ of John Smith

Figure 3 Original CV, Page 1.



PUBLICATIONS

- Smith, J. "Multilingual NLP for Low-Resource Settings," *Conference on Computational Linguistics.* *Year*: 2019
- Smith, J. and Turner, A. "Bias Detection in Neural Networks: A Practical Approach," *Journal of AI Ethics. Journal*: Computational Linguistics *Year*: 2021
- Smith, J., and Lee, M. "General NLP: Challenges and Opportunities," *Journal of Language Engineering. Journal*: Computational Linguistics *Year*: 2014

SKILLS

Programming Languages	Python, C++, Java
Tools	TensorFlow, PyTorch, Git
Languages	English (native), German (intermediate)

Figure 4 Original CV, Page 2.

The institution provides the following requirements for CV submission. Each requirement is addressed using one or more queries supported by the tool.

- **AR-01: Focus on NLP-related education, research, and publications.**
  Query: `Show 'Education', 'Work Experience', 'Publications'`

- **AR-02: Exclude unrelated roles (e.g., volunteer work, unrelated teaching).**
  Query: `Drop 'Volunteer Work', 'Teaching Experience'`

- **AR-03: Include only information from 2015 onward.**
  Query: `Show * Filtered by date >= '2015'`

- **AR-04: Include only publications relevant to NLP.**
  Query: `Show * Filtered by section = 'Publications' and theme = 'NLP'`

- **AR-05: Present sections in the following order: education, research, publications.**
  Query: `Reorder 'Education', 'Work Experience', 'Publications'`

- **AR-06: Provide both English and Portuguese versions.**
  Query: `Translate from 'en' to 'pt'`

After executing the defined queries, a new version of the CV LaTeX file is created, resulting in a new CV structure:

EUROPEAN CURRICULUM VITAE FORMAT

PERSONAL INFORMATION

Name	**John SMITH**
Address	**456 Research Lane, Boston, USA**
Telephone	**+1 617 555 0123**
Email	john.smith@university.edu
Homepage	http://johnsmith-research.com
Nationality	American
Date of birth	12, March, 1990

EDUCACAO

• Doutorado	*Ano*: 2019 *Instituicao*: Universidade de Harvard *Grau*: Ciencia da Computacao

EXPERIENCIA DE TRABALHO

EXPERIENCIA EM PESQUISA

• Bolsista de Pos-Doutorado	*Datas*: 2019-2021 *Empregador*: COM *Principais Atividades*: Conduziu pesquisas sobre aplicacoes de aprendizado de maquina em robotica.
• Assistente de Pesquisa	*Datas*: 2015-2019 *Empregador*: Universidade de Harvard *Principais Atividades*: Desenvolveu algoritmos para tarefas de processamento de linguagem natural.

PUBLICACOES

• Smith, J.	"PLN multilingue para ambientes de poucos recursos" *Conferencia sobre Linguistica Computacional*. *Ano*: 2019

Figure 5 Filtered CV.

6 Conclusion

The developed CVTool provides a complete web solution for managing LaTeX-based Curriculum Vitae documents through a custom query language and structured processing pipeline. To ensure functional reliability, a suite of automated tests was implemented using the *pytest* framework. These tests validated the behaviour of the CV parser, the domain-specific query language interpreter, and the controller logic responsible for applying transformations and generating output files. The inclusion of these tests throughout development contributed to the system's robustness and facilitates future maintenance.

Deployment was structured around Docker and Docker Compose to guarantee consistency across development and production environments. The backend and frontend services were containerised, with environment-specific configurations handled through dedicated .env files. This approach simplifies deployment and supports scalability. The system is publicly available and can be accessed at: `http://cvtool.epl.di.uminho.pt`, allowing users to explore the platform and evaluate its capabilities in real use cases.

To the best of our knowledge, there is currently no comparable tool offering a similarly integrated solution for LaTeX CV management. While this observation is made with due caution, it underscores the novelty and potential impact of the proposed system.

While the system achieves its intended goals, there is clear potential for further development. Enhancing query feedback through real-time syntax validation, introducing user accounts for persistent session management, improving LaTeX formatting recommendations, and supporting additional features such as template switching or change tracking are all viable next steps. Continued expansion of the testing framework and refinement of the document generation process will also strengthen the system's reliability and user experience. These improvements represent a well-defined path for evolving the platform into a more comprehensive and adaptable tool for LaTeX CV management.

References

1 A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers Principles, Techniques and Tools*. aw, 1986.

2 Victoria Baramidze. Latex for technical writing. *Journal of Technical Science and Technologies*, 2(2):44–50, 2020. doi:10.31578/jtst.v2i2.63.

3 Sergio Maia Dias, Alda Lopes Gancarski, and Pedro Rangel Henriques. Automatic generation of cvs from online social networks. In José-Luis Sierra-Rodríguez, José-Paulo Leal, and Alberto Simões, editors, *Languages, Applications and Technologies*, pages 258–263, Cham, 2015. Springer International Publishing. doi:10.1007/978-3-319-27653-3_25.

4 Martin Fowler. *Domain-specific languages*. Pearson Education, 2010.

5 Madiha Hameed, Muhammad Abrar, Ahmer Siddiq, and Tahir Javeed. Mvc software design pattern in web application development. *International Journal of Scientific & Engineering Research*, 5(5):17–20, 2014.

6 Markus Knauff and Jelica Nejasmic. An efficiency comparison of document preparation systems used in academic research and development. *PloS one*, 9(12):e115069, 2014.

7 Helmut Kopka, Helmut Kopka, P. W. Daly, and Patrick W. Daly. *Guide to LaTeX*. Addison-Wesley, 1999.

8 Tomaz Kosar, Nuno Oliveira, Marjan Mernik, Maria João Varanda Pereira, Matej Crepinsek, Daniela da Cruz, and Pedro Rangel Henriques. Comparing general-purpose and domain-specific languages: An empirical study. *ComSIS – Computer Science and Information Systems Journal, Special issue on Advances in Languages, Related Technologies and Applications*, 7(2):248–264, May 2010. doi:10.2298/CSIS1002247K.

9 Lark Developers. Lark – a modern parsing library for Python. `https://lark-parser.readthedocs.io/`, 2024. Accessed: 2025-04-25.

10 Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005. doi:10.1145/1118890.1118892.

Beyond the Score: Exploring the Intersection Between Sociodemographics and Linguistic Features in English (L1) Writing Placement

Miguel Da Corte
University of Algarve, Campus de Gambelas, Faro, Portugal
INESC-ID Lisboa, Portugal

Jorge Baptista
University of Algarve, Campus de Gambelas, Faro, Portugal
INESC-ID Lisboa, Portugal

Abstract

This study examines the intersection of sociodemographic characteristics, linguistic features, and writing placement outcomes at a community college in the United States of America. It focuses on 210 anonymized writing samples from native English speakers (L1) that were automatically classified by ACCUPLACER and independently assessed by two trained raters. Disparities across gender and race using 40 top-ranked linguistic features selected from Coh-Metrix, CTAP, and Developmental Education-Specific (DES) sets were analyzed. Three statistical tests were used: one-way ANOVA, Tukey's HSD, and Chi-square. ANOVA results showed racial differences in nine linguistic features, especially those tied to syntactic complexity, discourse markers, and lexical precision. Gender differences were more limited, with only one feature reaching significance (Positive Connectives, $p = 0.007$). Tukey's HSD pairwise tests showed no significant gender group variation but revealed sensitivity in DES features when comparing racial groups. Chi-square analysis indicated no significant association between gender and placement outcomes but suggested a possible link between race and human-assigned levels ($\chi^2 = 9.588$, $p = 0.048$). These findings suggest that while automated systems assess general writing skills, human-devised linguistic features and demographic insights can support more equitable placement practices for all students entering college-level programs.

2012 ACM Subject Classification Social and professional topics → Student assessment; Social and professional topics → Adult education; Computing methodologies → Language resources; Computing methodologies → Lexical semantics; Social and professional topics → Race and ethnicity; Social and professional topics → Men; Social and professional topics → Women

Keywords and phrases Developmental Education (DevEd), sociolinguistic variation, text classification, Machine Learning, placement equity

Digital Object Identifier 10.4230/OASIcs.SLATE.2025.6

Funding This work was supported by Portuguese national funds through FCT (Reference: UIDB/50021/2020, DOI: 10.54499/UIDB/50021/2020) and by the European Commission (Project: iRead4Skills, Grant number: 1010094837, Topic: HORIZON-CL2-2022-TRANSFORMATIONS-01-07, DOI: 10.3030/101094837).

1 Introduction and Objectives

Community colleges play a vital role in expanding access to higher education opportunities for high school graduates across the United States of America [21]. According to this country's Department of Education[1] definitions, community colleges are institutions responsible for

[1] `https://www.ed.gov/higher-education` (Last accessed: 29th July 2025; all URLs in this paper were checked on this date.)

providing access to the first two years of a Bachelor's degree, certificates in specialized trades, and/or the academic preparation for occupational credentials for work-ready programs. Operating, in most cases, under an open-admission policy, these institutions are also equipped to provide the academic support needed for students who are linguistically underprepared to join and complete an academic program.

At Tulsa Community College (TCC)[2], where this study took place, students who require academic support in key areas, particularly writing, are placed in a foundational literacy program known as Developmental Education (DevEd). Depending on their demonstrated writing skills prior to beginning an academic program, students can be placed into one of the following three levels (adapted version of the institution's official course descriptions):

- **DevEd Level 1**: for support with frequent grammar, spelling, and punctuation issues, and lacking cohesion;
- **DevEd Level 2**: for support with some structural improvements still needing targeted support;
- **College Level**: no support needed; skills show academic-level writing with minimal errors.

DevEd placement is typically determined using automated text classification systems. At TCC, ACCUPLACER[3] [41] is the system currently used to assess students' writing and assign placement levels accordingly. Existing literature raises concerns about the accuracy and fairness of such systems. For example, in [16] it was mentioned that automated placement tools misplace approximately one-third of students, either assigning them to courses that exceed their preparedness (overplacement) or to levels that do not reflect their actual skills (underplacement), raising both pedagogical and ethical concerns.

Placement decisions often show a mismatch between individual student data and overall institutional demographics, posing questions about whether those placed into non-college-level programs (e.g., DevEd) truly reflect the broader student population at their institutions [19, 35]. A contributing factor to these concerns is that, although some sociodemographic information is voluntarily self-disclosed by test-takers, most placement algorithms are designed to operate without reference to these factors (e.g., native language, race, gender) in an effort to ensure neutrality and avoid bias. However, excluding this information may limit the understanding of the underlying sociolinguistic context that shapes students' writing development; thus, potentially contributing to disparities in placement outcomes across different student groups [2, 31].

To further explore this gap, we investigate the relationship between sociodemographic variables, linguistic features in student writing, and placement outcomes. Our analysis draws on a corpus of 210 anonymized essays, randomly selected from 1,384 participants who self-identified as native English (L1) speakers, with each essay automatically and manually assessed for skill level. To validate linguistic differences and placement patterns, we employ a suite of statistical techniques, including one-way ANOVA, Tukey's Honestly Significant Difference (HSD) test [1, 36], and Chi-square tests of independence.

While a range of demographic characteristics exists (e.g., age, first-generation status, academic load), this study focuses specifically on gender and race for two reasons. First, these variables were consistently and reliably self-reported across both institutional and

[2] https://www.tulsacc.edu/
[3] https://accuplacer.collegeboard.org/

testing datasets, ensuring compatibility and completeness for statistical analysis. Second, other variables – such as age or socioeconomic status – were either unavailable, inconsistently reported, or insufficiently balanced to support valid statistical comparisons.

Grounded within the language resources and corpus linguistics domain, this study aims to contribute to the development of support tools for language teaching and assessment. It also connects with Machine Learning (ML) applications in Natural Language Processing and addresses ethical dimensions surrounding the fairness and transparency of automated placement systems. To support this investigation, we propose the following research questions:

1. How do the sociodemographic characteristics of the students selected for analysis align with the broader population of ACCUPLACER test-takers?
2. To what extent do gender and race correlate with the use of linguistic features in student writing at the developmental and college-entry levels?
3. Are there significant associations between these demographic variables (gender and race) and the placement outcomes – both automated (ACCUPLACER) and human-assigned?

Our paper is organized as follows: Section 2 reviews existing research on writing assessment and placement, linguistic features, and demographic disparities. Section 3 describes the institutional context, participant demographics, corpus, feature selection, and statistical procedures. Section 4 presents findings from the ANOVA and Tukey's HSD analyses. Section 5 examines associations between demographic variables and placement outcomes, both through ACCUPLACER and human assessment. We conclude by summarizing our key findings.

2 Related Work

Students' writing skills and placement into college-level or remedial courses are often assessed through standardized systems like ACCUPLACER. Existing literature raises concerns about the validity and fairness of such instruments [5, 14, 26], particularly given the disproportionate placement of students from historically marginalized groups into non-college level or DevEd courses [3, 35]. Placement into DevEd courses has been consistently associated with lower rates of retention[4] and completion[6], posing questions about the economical and social consequences of automated placement decisions [25, 30].

While systems like ACCUPLACER aim to reduce human biases by standardizing the assessment process, they often struggle to capture subtle indicators of students' early writing development. These "black box" tools typically emphasize surface-level correctness over deeper communicative competence, limiting their ability to assess whether students can meet academic writing demands [31]. As a result, students often leave the placement process without a clear understanding of what constitutes college-readiness in writing [18] or what traits of their written production may require remediation [29]. Researchers have argued that placement practices must move beyond raw scores to account for their broader consequences [22], as achievement gaps continue to persist across demographic groups. To help uncover potential mechanisms behind disproportionate placement, we explore the role of certain linguistic features, particularly those associated with developing writers in DevEd contexts, by examining how these features differ across racial groups and are also associated with placement levels.

[4] Percentage of first-time, full-time students who return the following year to continue with their academic program, according to the National Center for Education Statistics (NCES).[5]

[5] `https://nces.ed.gov/`

[6] Percentage of students who successfully complete a course with a grade of "C" or higher, relative to all students originally enrolled, as defined by NCES.

Recent studies further highlight disparities in writing placement outcomes by sex and racial background. For example, [6] reports that female students often outperform male peers in writing, while students from racially and minority groups, particularly Black students, tend to score lower on standardized assessments and, once placed in developmental courses, their completion rate is significantly low [13, 30]. Persistently low success rates, especially among Black students, suggest that DevEd courses may not adequately address students' linguistic needs. Equitable placement must therefore be paired with instruction that is both culturally responsive and linguistically informed. To address these challenges, assessment frameworks should integrate demographic considerations to: (i) prioritize direct measures of writing abilities to better understand students' linguistic diversity [27]; (ii) eliminate systemic barriers in classification, placement, and pedagogy that impede student success [25, 40]; and (iii) provide better (and more accurate) support for linguistically underprepared students [33, 39].

Efforts to improve transparency in writing placement tasks through linguistic feature analysis have gained momentum in recent years [24, 31]. Prior studies [10, 12] show that high-quality annotated corpora and careful feature selection significantly enhance the reliability of automated classification systems [17]. Human annotations, in particular, provide fine-grained insights into students' writing strengths and weaknesses, yielding improved classification outcomes across multiple ML models [21, 23, 32].

These findings align with broader research advocating for the integration of feature-rich platforms [42], such as Coh-Metrix[7] [28] and CTAP[8] [7], to enhance writing assessment practices [37]. Refining linguistic feature sets represents a critical step toward advancing placement equity [15], reducing misclassification risks, and supporting more targeted instructional interventions for students, including those enrolled in DevEd programs. This study contributes to these goals by offering new insights into placement outcomes through a linguistically informed lens.

3 Methodology

3.1 Institutional Demographics

This study builds on prior research into native English (L1) writing proficiency and placement practices [11], focusing on pre-enrollment students from the 2021–2022 and 2023–2024 academic years. In 2023–2024, TCC reported an unduplicated headcount of 14,538 students. As shown in Figure 1, White students made up nearly half the population, followed by Hispanic/Latino, Black or African American, and American Indian or Alaska Native students.

In terms of gender and academic load, as illustrated in Figure 2, approximately two-thirds of students were female and one-third male. Regarding academic load, 69% were enrolled part-time (taking about two courses per semester), while 31% were enrolled full-time (taking four to five courses). The average student age was 23, with 57% aged 24 or younger. Additionally, about 24% was first-generation college students, defined by NCES as those whose parents did not complete a college degree.

Given this demographic profile, writing placement and literacy emerge as central concerns, with nearly 50% of students placed into at least one DevEd course. The average pass rate for DevEd Levels 1 and 2 in 2023–2024 was 56%. Meanwhile, TCC's retention rate is approximately 65%, and the graduation rate for full-time students stands at 27%. These metrics are critical as institutions work to support student completion and transition into the workforce.

[7] https://soletlab.asu.edu/coh-metrix/
[8] https://sifnos.iwm-tuebingen.de/ctap/

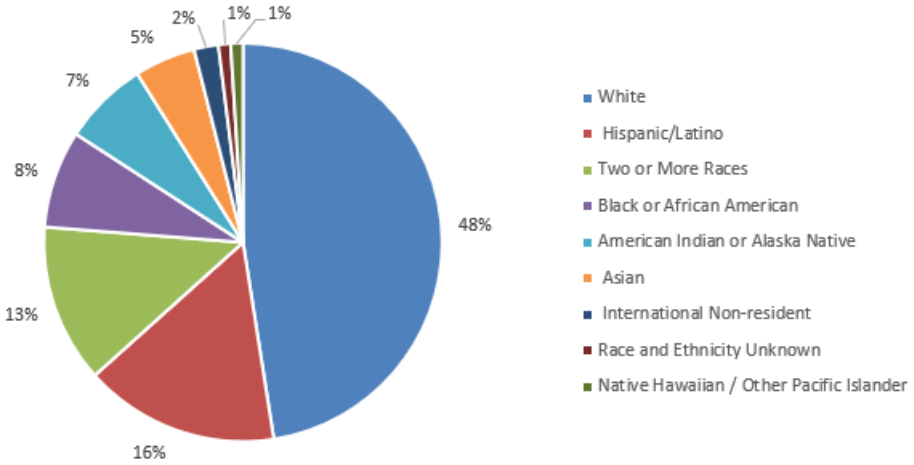


Figure 1 Distribution of the overall student population by race.

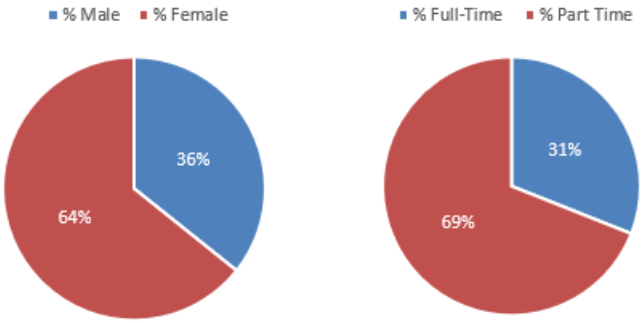


Figure 2 Distribution of the overall student population by gender and academic load.

3.2 Participants Demographics and Corpus Selection

To evaluate the representativeness of our sample relative to the broader ACCUPLACER test-taking population, as guided by our first research question, we examined institutional data from students who completed the ACCUPLACER exam during the same time period. Table 1 summarizes this data by gender, race, and native language.

A total of 2,003 students (unduplicated count) completed the ACCUPLACER test, producing a written sample that was automatically assessed and classified by proficiency level. In terms of gender, the test-taking population was composed of 62% females and 37% males, closely mirroring the broader institutional demographics. Regarding race, approximately 41% of students self-identified as White, followed by Black or African American (15%), American Indian (10%), and Latino/Hispanic (9%) students, proportions that closely align with the overall student population. First-language data further indicated that 69% of test-takers reported English as their native language, comparable to the 66% institutional ratio.

Building on prior investigations[9], this study narrows its analysis to the 1,384 participants who self-identified as native English (L1) speakers. Placement-level breakdowns for this group were available by gender only and are summarized in Table 2.

[9] An earlier phase of this research analyzed a random subset of 450 essays (from the 2,003 total); demographic data was not examined as placement was the focus.

Table 1 Demographic overview of ACCUPLACER test-takers during the 2021-2022 and 2023-2024 academic years.

Demographics	Group	Test-Takers	%
Gender	Female	1,244	62%
	Male	733	37%
	Other/Undisclosed	26	1%
	Total	**2,003**	**100%**
Race	American Indian	208	10%
	Black or African American	301	15%
	White	813	41%
	Latino/Hispanic	180	9%
	Other/Undisclosed	501	25%
	Total	**2,003**	100%
Native Language	English (L1)	**1,384**	**69%**
	Non-English	619	31%
	Total	**2,003**	**100%**

Table 2 ACCUPLACER placement by gender of English L1 test-takers.

Placement	Male	%	Female	%	Other/Undisclosed	%	Total
DevEd Level 1	24	21.82%	39	35.45%	2	1.82%	65
DevEd Level 2	225	22.59%	443	44.48%	5	0.50%	673
College Level	252	28.25%	378	42.38%	12	1.35%	642
Score not available	1	20.00%	3	60.00%	0	0.00%	4
Total	502	25.06%	863	43.09%	19	0.95%	1,384

From this table, we observed that approximately 53% of native English speakers were placed into one of the two DevEd levels (Level 1 or Level 2), with a higher concentration in Level 2. Placement into DevEd was slightly more common among female students. Notably, this 53% placement rate closely mirrors the broader institutional trend of approximately 50% outlined in Section 3.1.

To conduct a more detailed analysis, a subsample of 300 essays was randomly selected from the 1,384 native English speakers (≈22%). Within this subsample, complete and consistent demographic information was available for 210 participants, which formed the final dataset for this investigation. Among these participants, three racial groups are represented: American Indian (33; ≈16%), Black or African American (51; ≈24%), and White (125; ≈60%). One additional participant did not disclose their race but self-reported English as their native language (L1). Overall, the racial distribution in this subsample closely reflects both institutional demographics and those of the broader ACCUPLACER test-taking population.

Using the same placement classification criteria, we categorized these 210 participants into developmental and college-level groups, as summarized in Table 3.

Table 3 ACCUPLACER placement by gender; final dataset (n=210).

Placement	Male	%	Female	%	Other/Undisclosed	%	Total
DevEd Level 1	21	21%	33	33%	1	1%	**55**
DevEd Level 2	20	20%	57	57%	2	2%	**79**
College Level	23	23%	51	51%	2	2%	*76*
Total	64	21%	141	47%	5	2%	210

From this table, it is observed that students classified as DevEd (in bold) accounted for approximately 64% of the placements within the subsample (210 participants), while

those deemed proficient in writing (College Level; in italics) made up approximately 36%. Although smaller in size, this dataset, anchored in consistent demographic reporting, provides a foundation for the analyses examining the intersections between sociodemographic variables and linguistic features.

It is pertinent to note that all writing samples analyzed in this study were drawn from the institution's official entrance exam database. The writing task required students to compose an argumentative response (in English) to a standardized prompt encouraging reflection on a social, educational, or ethical issue. Students were instructed to complete the task in a proctored environment, in a single sitting, and without access to dictionaries, the internet, or AI tools.

3.3 Feature Selection

In addition to the ACCUPLACER classifications, all 210 essays with complete demographic information were manually annotated and evaluated for skill level by two trained raters following classification guidelines developed by the authors [8, 9]. To mitigate potential bias, multiple safeguards were implemented: all essays were anonymized, raters were blinded to students' demographic information, and a calibration session was conducted prior to annotation to ensure scoring consistency. In alignment with TCC's Review Board protocols, raters had no access to race, gender, or other sociodemographic data at any stage. While these measures reduced the risk of bias, we acknowledge that implicit bias may still persist.

The annotation process involved marking each text with a set of 11 Developmental Education-Specific (DES) linguistic features. These features captured both negative patterns (errors and deviations from proficiency) and positive patterns (indicators of advanced language use). They were organized into four clusters – Orthographic (ORT), Grammatical (GRAMM), Lexical and Semantic (LEXSEM), and Discursive (DISC), each reflecting a critical dimension of foundational writing proficiency, as summarized in Table 4.

Table 4 DevEd-specific (DES) features summary.

Pattern Description	Features Clustered
Orthographic patterns (ORT): representing the foundational language skills needed to represent words and phrases.	(-) Grapheme (addition, omission, transposition, and capitalization) (ORT) (-) Word split (WORDSPLIT) (-) Punctuation used & Contractions (PUNCT)
Grammatical patterns (GRAMM): evidencing the quality of text production.	(-) Word omitted (WORDOMIT) (-) Word repetition (WORDREPT) (-) Verb agreement (VAGREE) (-) Pronoun-alternation referential (ALTERN)
Lexical & Semantic patterns (LEXSEM): contributing to the structuring of a writer's discourse.	(-) Word precision (PRECISION) (+) Multiword expressions (MWE)
Discursive patterns (DISC): exhibiting the writer's ability to produce extended discourse.	(+) Argumentation with reason (REASON) (+) Argumentation with example (EXAMPLE)

Beyond the DES features, additional linguistic features were integrated from prior work [11], which included 106 from COH-METRIX and 328 from CTAP. Although 434 features[10] were originally extracted, only a subset was used in this study. Using Information Gain rankings from ML experiments, the most predictive features of placement were selected. Notably, two DES features – *EXAMPLE* and *ORT* – ranked 9th and 31st, respectively.

To emphasize the value of human-derived features, the remaining 9 DES features were retained, resulting in a final feature set of 40, as observed in Table 5. These DES features

[10] A detailed description of these features can be found in the documentation of these tools.

Table 5 Summary of highest-ranked features by Information Gain scores.

Rank	Source	Features	Info. gain
1	COH-Metrix	Paragraph length, number of sentences in a paragraph, mean	0.144
2	COH-Metrix	Lexical diversity, VOCD, all words	0.142
3	CTAP	Lexical Sophistication: Easy noun tokens (NGSL)	0.124
4	COH-Metrix	LSA given/new, sentences, mean	0.124
5	CTAP	Lexical Richness: Type Token Ratio (STTR NGSLeasy Nouns)	0.118
6	COH-Metrix	Lexical diversity, MTLD, all words	0.115
7	CTAP	Number of POS Feature: Plural noun Types	0.109
8	COH-Metrix	LSA given/new, sentences, standard deviation	0.109
9	**DES**	**EXAMPLE**	**0.103**
10	CTAP	POS Density Feature: Existential There	0.102
11	COH-Metrix	Positive connectives incidence	0.100
12	COH-Metrix	LSA overlap, adjacent sentences, mean	0.100
13	CTAP	Number of POS Feature: Existential there Types	0.099
14	CTAP	Number of POS Feature: Preposition Types	0.099
15	COH-Metrix	WordNet verb overlap	0.098
16	CTAP	Number of Syntactic Constituents: Postnominal Noun Modifier	0.097
17	CTAP	Number of Word Types (including Punctuation and Numbers)	0.097
18	CTAP	Lexical Sophistication Feature: SUBTLEX Logarithmic Word Frequency (AW Type)	0.096
19	CTAP	Number of POS Feature: Existential there Tokens	0.096
20	CTAP	Lexical Sophistication: Easy noun types (NGSL)	0.094
21	CTAP	Number of POS Feature: Verbs in past participle form Types	0.092
22	COH-Metrix	LSA verb overlap	0.092
23	CTAP	Number of Unique Words	0.092
24	CTAP	Number of Tokens with More Than 2 Syllables	0.090
25	CTAP	Number of Word Types with More Than 2 Syllables	0.086
26	CTAP	Lexical Richness: HDD (excluding punctuation and numbers)	0.086
27	CTAP	POS Density Feature: Possessive Ending	0.086
28	CTAP	Number of Syntactic Constituents: Complex Noun Phrase	0.084
29	CTAP	Number of POS Feature: Plural noun Tokens	0.083
30	CTAP	Referential Cohesion: Global Lemma Overlap	0.083
31	**DES**	**ORT**	**0.082**

offer a finer-grained assessment of students' communicative effectiveness in writing prior to beginning their academic programs [20] and have been documented before and associated with "discrete traits" of academic writing quality [29, 43]. Given ongoing concerns about disparities in placement outcomes across demographic lines, integrating these features with the sociodemographic variables identified for this study (gender and race) allows for a more comprehensive understanding of how writing traits intersect with placement decisions.

3.4 Statistical Tests

Three commonly used statistical tests, validated by the literature, were employed in this study: a one-way ANOVA, Tukey's HSD, Chi-square test of independence.

The one-way ANOVA test was selected to determine whether statistically significant differences existed in linguistic feature means across groups with more than two categories, specifically gender and race, as supported in the literature [36]. For the interpretation of the p-values, we used the following thresholds, commonly mentioned in the literature [38]:

$p < 0.01$ indicates highly significant group differences;
$0.01 \leq p < 0.05$ indicates statistically significant group differences;
$0.05 \leq p < 0.10$ indicates marginally significant group differences;
$p \geq 0.10$ indicates not significant group differences

While a multi-way ANOVA could have been used to test for interaction effects between gender and race, it was not applied here due to sample size limitations across intersecting subgroups (race $\times$ gender). To meet the assumptions of normality and homogeneity of

variances, which are required for valid interaction testing, we opted for separate one-way ANOVAs to analyze each variable independently.

Following the ANOVA calculations, we conducted Tukey's HSD tests to perform pairwise comparisons. This post-hoc analysis identified which specific groups differed significantly and gauged the magnitude of those differences. Tukey's HSD has also been used in prior writing studies examining complexity and accuracy in student writing proficiency [4].

Finally, to examine potential associations between the demographic variables in this study and placement levels, both ACCUPLACER and human assigned, we used the Chi-square test of independence, appropriate for the categorical nature of placement outcomes.

4 Correlating Sociodemographics and Linguistic Features

Correlation with Gender

To address the second research question, namely, to what extent gender and race correlate with the use of linguistic features in student writing, we applied a one-way ANOVA test to the 210 essays completed by native English speakers. The linguistic features described in Section 3.3 were analyzed against the available demographic data for this subset.

Through a one-way ANOVA test, the means of the 40 continuous (ratios) linguistic variables were compared across three gender groups (male, female, other/undisclosed). In preparation for the calculations, we adopted the conventional $\alpha = 0.05$ threshold. The degrees of freedom between groups ($\mathrm{df_B}$) were calculated using: $\mathrm{df}_B = k - 1$, yielding $\mathrm{df}_B = 2$ given the three gender groups. For the degrees of freedom within groups ($\mathrm{df_W}$), the following formula was used: $\mathrm{df}_W = N - k$. While we have 210 individual data points, 3 groups' means are calculated; therefore, $\mathrm{df}_W = 207$, which is the number of degrees of freedom that remain for estimating variability within the groups. For subsequent experiments the same formulas were employed. All statistical calculations in this study were performed using Python.

When the ANOVA calculations were performed, 36 out of the 40 linguistic features showed no statistically significant differences across the three gender groups, as summarized in Table 6. Of the remaining four features, *Positive connectives incidence* (in bold), which captures words that aid in discourse flow and cohesion (e.g., *however*, *similarly*), showed a p-value indicating a significant difference across gender groups ($p = 0.007$), a textual characteristic previously examined in writing assessment research [37]. The other three features (in italics), *Lexical Richness: HDD (excluding punctuation and numbers)* (p-value = 0.051); *Lexical Richness: Type Token Ratio (STTR NGSLeasy Nouns)* (p-value = 0.094); and *Number of POS Feature: Existential there Types* (p-value = 0.064), indicating marginally significant group variation, which warrants further investigation.

Correlation with Race

Following the same statistical approach described in Section 4, we examined whether differences emerged in the use of all 40 linguistic features across four racial groups (American Indian, Black or African American, White, Other/Undisclosed). This time $\mathrm{df}_B = 3$ given the four racial groups. While we still have 210 individual data points, 4 groups' means are calculated; therefore, $\mathrm{df}_W = 206$.

As shown in Table 7, 31 features showed no meaningful differences across racial groups (compared to 36 features in the gender-based analysis).

The remaining nine features, particularly those connected to syntax complexity, discourse markers, and lexical precision, revealed differences among racial groups. Out of the nine,

Table 6 ANOVA summary of linguistic features by gender group (M = male; F = female; O = other/undisclosed). Significance levels are indicated as follows: S = significant, M = marginally significant, N = not significant.

Features	Mean			Grand Mean	F-Stat.	*p*	Interp.
	(M)	(F)	(O)				
Positive connectives incidence	79.792	91.356	92.644	87.863	5.020	**0.007**	**S**
Lexical Richness:							
HDD (excluding punctuation and numbers)	0.752	0.809	0.830	0.792	3.013	*0.051*	*M*
Lexical Richness: Type Token Ratio							
(STTR NGSLeasy Nouns)	64.155	81.123	74.464	75.793	2.393	*0.094*	*M*
Number of POS Feature: Existential there Types	0.656	0.695	1.400	0.700	2.791	*0.064*	*M*
Paragraph length, number of sentences in a paragraph, mean	16.766	18.184	15.800	17.695	0.581	0.560	N
LSA overlap, adjacent sentences, mean	0.192	0.195	0.157	0.193	0.442	0.644	N
LSA given/new, sentences, mean	0.292	0.305	0.271	0.301	1.946	0.145	N
LSA given/new, sentences, standard deviation	0.138	0.135	0.114	0.135	1.042	0.355	N
Lexical diversity, MTLD, all words	74.905	71.166	79.811	72.512	0.884	0.415	N
Lexical diversity, VOCD, all words	76.060	75.993	79.409	76.095	0.026	0.975	N
LSA verb overlap	0.095	0.099	0.089	0.098	0.619	0.540	N
WordNet verb overlap	0.516	0.524	0.498	0.521	0.258	0.773	N
Lexical Sophistication Feature: SUBTLEX Log. Word Freq. (AW Type)	4.155	4.227	4.106	4.202	1.678	0.189	N
Lexical Sophistication: Easy noun tokens (NGSL)	38.188	43.660	41.800	41.948	1.391	0.251	N
Lexical Sophistication: Easy noun types (NGSL)	23.578	24.858	24.600	24.462	0.253	0.777	N
Number of POS Feature: Existential there Tokens	1.156	1.227	1.600	1.214	0.198	0.820	N
Number of POS Feature: Plural noun Tokens	19.453	19.660	20.200	19.610	0.012	0.988	N
Number of POS Feature: Plural noun Types	13.391	12.695	14.200	12.943	0.246	0.782	N
Number of POS Feature: Preposition Types	16.094	16.326	17.400	16.281	0.137	0.872	N
Number of POS Feature: Verbs in past participle form Types	5.328	5.184	8.200	5.300	1.401	0.249	N
Number of Syntactic Constituents: Complex Noun Phrase	38.813	39.348	45.400	39.329	0.259	0.772	N
Number of Syntactic Constituents: Prenominal Noun Modifier	20.453	19.106	21.400	19.571	0.340	0.712	N
Number of Tokens with More Than 2 Syllables	94.875	93.780	94.600	94.133	0.011	0.989	N
Number of Unique Words	104.922	101.567	113.600	102.876	0.291	0.748	N
Number of Word Types (including Punctuation and Numbers)	163.219	166.014	174.800	165.371	0.092	0.913	N
Number of Word Types with More Than 2 Syllables	71.359	67.532	76.200	68.905	0.352	0.704	N
POS Density Feature: Existential There	0.003	0.003	0.004	0.003	0.125	0.883	N
POS Density Feature: Possessive Ending	0.002	0.001	0.003	0.002	0.568	0.568	N
Referential Cohesion: Global Lemma Overlap	0.609	0.726	0.770	0.691	1.125	0.327	N
MWE	0.072	0.071	0.061	0.071	0.121	0.886	N
VDISAGREE	0.002	0.003	0.003	0.003	0.683	0.506	N
ORT	0.051	0.046	0.030	0.047	0.855	0.427	N
PUNCT	0.042	0.043	0.041	0.043	0.044	0.957	N
WORDOMIT	0.006	0.008	0.005	0.007	0.821	0.442	N
PRECISION	0.011	0.011	0.006	0.011	0.436	0.648	N
WORDREPT	0.002	0.002	0.003	0.002	0.538	0.585	N
REASON	0.005	0.006	0.005	0.005	0.196	0.822	N
WORDSPLIT	0.002	0.002	0.001	0.002	0.306	0.737	N
EXAMPLE	0.002	0.003	0.002	0.002	0.154	0.858	N
ALTERN	0.001	0.002	0.001	0.001	0.084	0.920	N

four of them (bolded), namely, *Positive Connectives Incidence*, *PUNCT*, *WORDOMIT*, and *PRECISION*, yielded *p*-values that indicated significant differences across race groups. Two features (also bolded), *Lexical Sophistication (SUBTLEX Logarithmic Word Frequency)* and *Number of Syntactic Constituents: Prenominal Noun Modifier*, produced *p*-values suggesting statistically significant group differences. The remaining three (in italics), *WordNet Verb Overlap*, *ORT*, and *REASON*, were considered marginally significant, indicating that how these groups exhibit (or not) these linguistic features in their written productions requires further investigation.

Five of these features belong to the DES feature set, suggesting that racial group differences may be more pronounced when assessment focuses on more targeted, humanly-devised indicators of developmental writing. Notably, the *Positive Connectives Incidence* feature emerged as significant in both the gender and race analyses, potentially highlighting its importance in capturing variation in students' writing patterns. The differences here analyzed were not uniformly distributed across all features, indicating that group identity

Table 7 ANOVA summary of linguistic features by race group (1 = American Indian; 2 = Black or African American; 3 = White; 5 = Other/Undisclosed). Significance levels are indicated as follows: S = significant, St = statistically significant; M = marginally significant, N = not significant.

Features	Mean				Grand Mean	F-Stat.	*p*	Interp.
	(1)	(2)	(3)	(5)				
PUNCT	0.037	0.056	0.038	0.091	0.043	6.674	**0.000**	**S**
WORDOMIT	0.006	0.013	0.006	0.011	0.007	8.796	**0.000**	**S**
PRECISION	0.009	0.018	0.009	0.023	0.011	5.871	**0.001**	**S**
Positive connectives incidence	78.282	96.936	86.524	108.571	87.863	4.439	**0.005**	**St**
Lexical Sophistication Feature:								
SUBTLEX Log. Word Freq. (AW Type)	4.169	4.301	4.167	4.634	4.202	3.634	**0.014**	**St**
Number of Syntactic Constituents:								
Prenominal Noun Modifier	20.546	15.706	20.992	7.000	19.571	2.913	**0.035**	**St**
WordNet verb overlap	0.502	0.553	0.513	0.576	0.521	2.200	*0.089*	*M*
ORT	0.045	0.059	0.042	0.091	0.047	2.553	*0.057*	*M*
REASON	0.005	0.008	0.005	0.000	0.005	2.509	*0.060*	*M*
Paragraph length, number of sentences in a paragraph, mean	17.576	16.353	18.368	6.000	17.695	1.038	0.377	N
LSA overlap, adjacent sentences, mean	0.197	0.213	0.183	0.283	0.193	1.746	0.159	N
LSA given/new, sentences, mean	0.304	0.300	0.300	0.314	0.301	0.061	0.980	N
LSA given/new, sentences, standard deviation	0.141	0.141	0.132	0.165	0.135	1.385	0.249	N
Lexical diversity, MTLD, all words	73.130	68.164	74.002	87.500	72.512	0.981	0.403	N
Lexical diversity, VOCD, all words	75.327	70.684	78.469	80.586	76.095	0.687	0.561	N
LSA verb overlap	0.094	0.100	0.098	0.086	0.098	0.264	0.851	N
Lexical Richness: HDD (excluding punctuation and numbers)	0.773	0.783	0.801	0.825	0.792	0.354	0.787	N
Lexical Richness: Type Token Ratio								
(STTR NGSLeasy Nouns)	70.466	74.924	78.026	16.900	75.793	0.625	0.599	N
Lexical Sophistication: Easy noun tokens (NGSL)	40.000	39.765	43.584	13.000	41.948	1.081	0.358	N
Lexical Sophistication: Easy noun types (NGSL)	23.303	22.726	25.592	10.000	24.462	1.341	0.262	N
Number of POS Feature: Existential there Tokens	0.909	1.275	1.280	0.000	1.214	0.713	0.546	N
Number of POS Feature: Existential there Types	0.576	0.647	0.760	0.000	0.700	1.133	0.337	N
Number of POS Feature: Plural noun Tokens	19.455	17.412	20.632	9.000	19.610	1.112	0.345	N
Number of POS Feature: Plural noun Types	13.970	11.059	13.496	6.000	12.943	1.719	0.164	N
Number of POS Feature: Preposition Types	15.758	15.863	16.624	12.000	16.281	0.532	0.661	N
Number of POS Feature:								
Verbs in past participle form Types	5.121	4.412	5.736	2.000	5.300	1.620	0.186	N
Number of Syntactic Constituents: Complex Noun Phrase	37.697	36.059	41.256	19.000	39.329	1.309	0.272	N
Number of Tokens with More Than 2 Syllables	92.909	85.392	98.496	35.000	94.133	1.316	0.270	N
Number of Unique Words	102.546	93.961	106.920	63.000	102.876	1.395	0.246	N
Number of Word Types								
(including Punctuation and Numbers)	160.727	155.137	171.280	102.000	165.371	1.113	0.345	N
Number of Word Types with More Than 2 Syllables	69.515	59.686	72.840	27.000	68.905	2.101	0.101	N
POS Density Feature: Existential There	0.003	0.003	0.004	0.000	0.003	0.516	0.672	N
POS Density Feature: Possessive Ending	0.002	0.002	0.001	0.000	0.002	0.052	0.984	N
Referential Cohesion: Global Lemma Overlap	0.605	0.689	0.719	0.200	0.691	0.680	0.565	N
MWE	0.069	0.078	0.069	0.120	0.071	0.844	0.471	N
VDISAGREE	0.003	0.003	0.002	0.011	0.003	0.967	0.409	N
WORDREPT	0.001	0.003	0.002	0.000	0.002	0.754	0.521	N
WORDSPLIT	0.002	0.002	0.002	0.000	0.002	0.071	0.976	N
EXAMPLE	0.002	0.002	0.003	0.000	0.002	0.617	0.605	N
ALTERN	0.002	0.002	0.001	0.000	0.001	0.368	0.776	N

may influence only specific aspects of writing.

Beyond the One-way ANOVA

While the one-way ANOVA identified overall group differences by gender and race, Tukey's HSD post-hoc test was used to determine which specific groups differed from one another.

For the gender analysis, we zeroed in on the statistical differences highlighted in Section 4[11]. The sample analyzed consisted of 141 females (F) and 64 males (M), for a total of 205 participants. The "Other/Undisclosed" category, which had only 5 participants, was excluded in an attempt to make comparisons involving the other two groups as statistically stable as possible. Consequently, for the Tukey's HSD test, the following values were used: $k = 2$ (groups: Male, Female); $\mathrm{df}_W = 203$; $\alpha = 0.05$; $q_{\text{critical}} = 2.788$.

[11] For brevity, Tukey's HSD tests were only conducted on this feature set; other features, including those identified for racial groups, will be analyzed in future work.

Table 8 Tukey's HSD test results for selected features for all gender groups: M = males; F = females. Significance levels are indicated as follows: N = not significant.

Feature	Mean Dif. (F-M)	Stand. Err.	q-Stat	Sign.
Positive connectives incidence	11.564	5.161	2.241	N
Lexical Richness: HDD (excluding punctuation and numbers)	0.058	0.033	1.736	N
Lexical Richness: Type Token Ratio (STTR NGSLeasy Nouns)	16.968	10.970	1.547	N
Number of POS Feature: Existential there Types	0.039	0.023	1.671	N

From Table 8, we see that all pairwise comparisons by gender were not statistically significant at $\alpha = 0.05$, as the calculated q-statistics for all four linguistic features examined fell below the critical value of $q_{\text{critical}} = 2.788$. Although some mean differences, particularly for *Positive Connectives Incidence* were notable, they did not exceed the threshold for significance. Consequently, we cannot reject the null hypothesis, which posits no meaningful differences in the use of linguistic features across gender groups. These findings should be interpreted with caution, as the limited sample size and unequal group distributions could limit the statistical power to detect more subtle effects.

Table 9 Tukey's HSD test results for selected features for all race groups: 1 = American Indian; 2 = Black or African American; 3 = White.

Feature	Mean Diff			Stand. Error	q-Stat		
	(1–2)	(1–3)	(2–3)		(1–2)	(1–3)	(2–3)
PUNCT	0.019	0.001	0.018	0.006	**3.247**	0.174	**3.074**
WORDOMIT	0.007	0.000	0.007	0.002	**3.606**	0.052	**3.658**
PRECISION	0.009	0.001	0.008	0.003	**3.065**	0.204	**2.860**
Positive connectives incidence	18.654	8.243	10.411	6.275	**2.973**	1.314	1.659
Lexical Sophistication Feature:							
SUBTLEX Log Word Freq (AW Type)	0.132	0.002	0.134	0.057	2.319	0.032	2.350
Number of Syntactic Constituents:							
Prenominal Noun Modifier	4.840	0.447	5.286	2.429	1.992	0.184	2.176
WordNet verb overlap	0.050	0.011	0.040	0.025	1.990	0.419	1.570
ORT	0.013	0.003	0.017	0.008	1.717	0.413	2.131
REASON	0.002	0.001	0.003	0.001	1.614	0.538	2.152

We then turned to the post-hoc analysis for the race groups. For Tukey's HSD test, the following values were used: $k = 3$ [race groups: American Indian (1), Black or African American (2), and White (3)]. The Other/Undisclosed race category previously mentioned included only one student. Due to this small count, this category was excluded from the statistical analysis in an attempt to preserve the reliability of the test results. For $\text{df}_W = 208$; $\alpha = 0.05$; $q_{\text{critical}} = 2.788$.

As shown in Table 9, statistically significant differences (in bold) were found for *WORDOMIT*, *PUNCT*, and *PRECISION* between Groups 1 and 2 as well as Groups 2 and 3. A borderline yet statistically significant difference in *Positive Connectives Incidence* was also observed between Groups 1 and 2 only. Based on the individual means already reported in Table 7, students who identified as Black or African American tended to exhibit a higher incidence of omitted or left-out parts of speech compared to both American Indian and White students. These omissions may affect the overall coherence of their writing. In contrast, American Indian and White students showed nearly identical omission ratios.

Students identifying as Black or African American also used punctuation (e.g., for joining clauses) less frequently than their peers, and exhibited a higher rate of imprecise word usage to describe a concept, as captured by the *PRECISION* feature. On the contrary, Black or African American students showed a higher incidence of *Positive Connectives* than American Indian students, suggesting greater use of linking expressions such as *however* or *similarly*.

These differences may reflect dialectal or regional variation rather than deficits in academic ability [34]. However, given that placement decisions are tied to institutional standards that may not fully account for this linguistic diversity, it is essential that pedagogical interventions not be seen as merely corrective measures but as opportunities to support students in expanding their register and linguistic skills to better navigate academic expectations.

It is equally important to acknowledge that these differences were identified from samples produced at a single point in time, and that external factors, such as test anxiety or unfamiliarity with the writing prompt, may have also influenced students' performance.

5 Assessing Level Assignment with Demographics

Assesment with Gender

To evaluate the possible association between gender and ACCUPLACER placement levels, in response to our third research question, a Chi-square test of independence was performed as the final step in this analysis. Table 10 presents the observed counts, expected counts under the null hypothesis of independence, and the individual Chi-square contributions $(O-E)^2/E$ for each cell.

Table 10 Observed vs. Expected Counts for ACCUPLACER Placement by Gender with Chi-square Contributions.

Placement Level	Gender	Observed	Expected	$(O - E)^2 / E$
DevEd Level 1	Male	21	16.859	1.017
	Female	33	37.141	0.462
DevEd Level 2	Male	20	24.039	0.679
	Female	57	52.961	0.308
College Level	Male	23	23.102	0.000
	Female	51	50.898	0.000
Total	-	205	205	2.467

The expected counts were calculated based on the assumption that gender and placement level are independent, and the $(O-E)^2/E$ values represent each cell's contribution to the overall Chi-square statistic. The total sum of these contributions yielded the total Chi-square value ($\chi^2 = 2.467$). To assess statistical significance, we compared the calculated χ^2 value to the critical Chi-square value ($\chi^2_{\text{critical}} = 5.991$), determined for $df = 2$ and $\alpha = 0.05$. Alternatively, the associated p-value ($p = 0.291$) was considered. Since $\chi^2 < \chi^2_{\text{critical}}$ and $p > 0.05$, we cannot reject the null hypothesis and conclude that there is no statistically significant association between gender and the automatically assigned skill levels via ACCUPLACER.

The same procedure was applied using the human-assigned skill levels, with identical degrees of freedom and significance thresholds. Because human raters assigned different placement levels than ACCUPLACER, the distribution of texts by levels varies.

The expected counts, as included in Table 11, were calculated based on the same assumption that gender and placement level are independent, and the $(O-E)^2/E$ values represent each cell's contribution to the overall Chi-square statistic. The total sum of these contributions yielded the total Chi-square value ($\chi^2 = 0.415$). The critical Chi-square value was the same ($\chi^2_{\text{critical}} = 5.991$). Alternatively, the associated p-value ($p = 0.812$) was considered. Since $\chi^2 < \chi^2_{\text{critical}}$ and $p > 0.05$, we also conclude that there is no statistically significant association between gender and the human-assigned placement levels.

Table 11 Observed vs. Expected Counts for Human Placement by Gender with Chi-square Contributions.

Placement Level	Gender	Observed	Expected	$(O - E)^2 / E$
DevEd Level 1	Male	14	13.112	0.060
	Female	28	28.888	0.027
DevEd Level 2	Male	36	38.088	0.114
	Female	86	83.912	0.051
College Level	Male	14	12.8	0.112
	Female	27	28.2	0.051
Total	-	205	205	0.415

Assessment with Race

We also examined whether there was an association first between racial groups and the classification by skill level produced by ACCUPLACER, and, second, with the human ratings. The distribution of placement levels across racial groups, along with their corresponding observed and expected counts and Chi-square contributions, is presented in Table 12.

Table 12 Observed vs. Expected Counts for (ACCUPLACER) Placement by Race with Chi-square Contributions.

Placement Level	Race	Observed	Expected	$(O - E)^2 / E$
DevEd Level 1	American Indian	10	8.526	0.255
	Black or African American	14	13.177	0.051
	White	30	32.297	0.163
DevEd Level 2	American Indian	10	12.474	0.491
	Black or African American	25	19.278	1.699
	White	44	47.249	0.223
College Level	American Indian	13	12.000	0.083
	Black or African American	12	18.546	2.310
	White	51	45.455	0.677
Total	-	209	209	5.952

The degrees of freedom were calculated based on 3 three levels (DevEd Level 1, DevEd Level 2, College Level) and 3 racial groups (American Indian, Black or African American, and White). The computed Chi-square statistic yielded a value of $\chi^2 = 5.952$, which does not exceed the critical value of 9.488 for 4 degrees of freedom at the $\alpha = 0.05$ level. The associated p-value of $p = 0.203$ is above the conventional threshold for statistical significance. Therefore, we cannot reject the null hypothesis, indicating no statistically significant association between race and ACCUPLACER placement level within this sample.

Finally, the assessment was repeated but with the humanly-assigned skill level. Table 13 summarizes the observed and expected counts and Chi-square contributions.

The computed Chi-square statistic yielded a value of $\chi^2 = 9.588$, which slightly exceeds the critical value of 9.488 for 4 degrees of freedom at the $\alpha = 0.05$ level. The associated p-value of $p = 0.048$ falls just below the conventional threshold for statistical significance. While this suggests a possible association between race and human-assigned placement levels, even though raters did not have access to demographic data, the result is borderline and should be interpreted cautiously, warranting further investigation.

Upon analyzing each individual Chi-square contribution, we focused on values with the greatest discrepancies between the observed and expected counts within the race. This approach enabled us to identify, within the context of this sample, whether certain racial

Table 13 Observed vs. Expected Counts for (Human) Placement by Race with Chi-square Contributions.

Placement Level	Race	Observed	Expected	(O - E)² / E
DevEd Level 1	American Indian	4	6.914	1.228
	Black or African American	*17*	*10.686*	3.731
	White	22	26.191	0.671
DevEd Level 2	American Indian	20	19.486	0.014
	Black or African American	29	30.114	0.041
	White	*75*	*73.810*	0.019
College Level	American Indian	9	6.600	0.873
	Black or African American	*5*	*10.200*	2.651
	White	28	25.000	0.360
Total	-	209	209	9.588

groups were overrepresented or underrepresented at specific placement levels. A higher observed count relative to the expected suggests overrepresentation, with a lower observed count indicating underrepresentation (italicized values).

In the DevEd Level 1 category, students who self-identified as Black or African American appeared to be more frequently than expected placed within this level, which may help explain their comparatively lower representation at the College Level. In contrast, students who identified as White were more frequently placed in DevEd Level 2 and College Level, indicating a higher-than-expected concentration in those categories. The variations between the observed and expected contributions for American Indians, within each placement level, were not as pronounced as those of the other groups.

6 Conclusions

This study investigated the intersection of sociodemographic characteristics, linguistic features, and writing placement outcomes in a higher education context. We analyzed a corpus of 210 anonymized writing samples from native English speakers (L1), representative of the institution's demographic composition. Using 40 top-ranked linguistic features drawn from Coh-Metrix, CTAP, and Developmental Education-Specific (DES) feature sets, we assessed potential disparities across gender and race using three statistical tests: one-way ANOVA, Tukey's HSD, and Chi-square.

The results we obtained provided evidence that not all linguistic features are equally sensitive to demographic variation. Gender, in particular, does not appear to have an influence on the linguistic features measured. This finding supports the idea that, at least for the features examined, variation in writing may not be primarily driven by gender.

Among racial groups, in contrast, a total of 9 features, particularly those tied to syntax complexity, discourse markers, and lexical precision, revealed differences. These features were mostly found within the human-devised set (DES) and aligned with the focus of DevEd courses. While not designed to directly measure academic ability, these DES features offer valuable diagnostic insight into students' current writing performance at the onset of college and align with DevEd's mission of identifying support needs for student success.

Comparing automated and human-assigned placement classifications, no significant associations were found with gender. However, a borderline significant association between race and human-assigned placement levels ($p = 0.048$) raises important questions about raters' sensitivity to linguistic features that intersect with race. This finding highlights the need for continued efforts to ensure placement fairness and address potential bias. Because

human evaluation remains essential for detecting nuanced linguistic patterns that automated systems may overlook, our findings advocate for a combined approach that leverages the strengths of human judgment and automated analysis to potentially: (i) enhance placement accuracy and educational outcomes; (ii) guide targeted instruction; and (iii) support the development of more equitable and responsive placement tools.

References

1 Hervé Abdi and Lynne J Williams. Tukey's Honestly Significant Difference (HSD) test. *Encyclopedia of Research Design*, 3(1):1–5, 2010.

2 Lisa R Arnold, Lei Jiang, and Holly Hassel. After implementation: Assessing student self-placement in college writing programs. *Journal of Writing Assessment*, 17(1), 2024.

3 Elisabeth A Barnett, Elizabeth Kopko, Dan Cullinan, and Clive R Belfield. Who should take college-level courses? Impact findings from an evaluation of a multiple measures assessment strategy. *Center for the Analysis of Postsecondary Readiness*, 2020.

4 Jessie S Barrot and Joan Y Agdeppa. Complexity, accuracy, and fluency as indices of college-level L2 writers' proficiency. *Assessing Writing*, 47:100510, 2021.

5 Susan Bickerstaff, Katie Beal, Julia Raufman, Erika B Lewy, and Austin Slaughter. Five principles for reforming Developmental Education: A review of the evidence. *Center for the Analysis of Postsecondary Readiness*, pages 1–8, 2022.

6 Carolina Castillo and Natalia Ávila Reyes. Students' sociodemographic characteristics and writing performance: A systematic literature review. *Reading and Writing*, pages 1–37, 2025.

7 Xiaobin Chen and Detmar Meurers. CTAP: A Web-Based Tool Supporting Automatic Complexity Analysis. In *Proceedings of the Workshop on Computational Linguistics for Linguistic Complexity (CL4LC)*, pages 113–119, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee. URL: https://aclanthology.org/W16-4113.

8 Miguel Da Corte and Jorge Baptista. Charting the linguistic landscape of developing writers: An annotation scheme for enhancing native language proficiency. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue, editors, *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 3046–3056, Torino, Italia, May 2024. ELRA and ICCL. URL: https://aclanthology.org/2024.lrec-main.272/.

9 Miguel Da Corte and Jorge Baptista. Enhancing writing proficiency classification in Developmental Education: The quest for accuracy. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue, editors, *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 6134–6143, Torino, Italia, May 2024. ELRA and ICCL. URL: https://aclanthology.org/2024.lrec-main.542/.

10 Miguel Da Corte and Jorge Baptista. Leveraging NLP and machine learning for English (l1) writing assessment in developmental education. In *Proceedings of the 16th International Conference on Computer Supported Education (CSEDU 2024), 2-4 May, 2024, Angers, France*, volume 2, pages 128–140, 2024. doi:10.5220/0012740500003693.

11 Miguel Da Corte and Jorge Baptista. Refining English writing proficiency assessment and placement in Developmental Education using NLP tools and Machine Learning. In *Proceedings of the 17th International Conference on Computer Supported Education - Volume 2: CSEDU*, pages 288–303. INSTICC, SciTePress, 2025. doi:10.5220/0013351500003932.

12 Miguel Da Corte and Jorge Baptista. Toward consistency in writing proficiency assessment: Mitigating classification variability in Developmental Education. In *Proceedings of the 17th International Conference on Computer Supported Education - Volume 2: CSEDU*, pages 139–150. INSTICC, SciTePress, 2025. doi:10.5220/0013353900003932.

13 Jane Denison-Furness, Stacey Lee Donohue, Annemarie Hamlin, and Tony Russell. Welcome/Not Welcome: From Discouragement to Empowerment in the Writing Placement Process at Central Oregon Community College. In Jassica Nastal, Mya Poe, and Christie Toth, editors, *Writing Placement in Two-Year Colleges: The Pursuit of Equity in Postsceondary Education*, pages 107–127. The WAC Clearinghouse/University Press of Colorado, 2022. doi:10.37514/PRA-B.2022.1565.2.04.

14 Martin East and David Slomp. The ethical turn in writing assessment: How far have we come, and where do we still need to go? *Language Teaching*, 57(2):262–273, 2024.

15 Nikki Edgecombe and Michael Weiss. Promoting equity in Developmental Education reform: A conversation with Nikki Edgecombe and Michael Weiss. *Center for the Analysis of Postsecondary Readiness*, page 1, 2024.

16 Elizabeth Ganga and Amy Mazzariello. Modernzing college course placement by using multiple measures. *Education Commission of the States*, pages 1–9, 2019. URL: https://postsecondaryreadiness.org/wp-content/uploads/2019/03/Modernizing_College_Course_Placement_by_Using_Multiple_Measures_Final.pdf.

17 Sandra Götz and Sylviane Granger. Learner corpus research for pedagogical purposes: An overview and some research perspectives. *International Journal of Learner Corpus Research*, 10(1):1–38, 2024.

18 Sarah Hirsch, Kenny Smith, and Madeleine Sorapure. Collaborative writing placement: Partnering with students in the placement process. *Journal of Writing Assessment*, 17(2), 2024.

19 Darin L Jensen and Joanne Baird Giordano. Afterword. Placement, equity, and the promise of democratic open-access education. *Writing Placement in Two-Year Colleges: The Pursuit of Equity in Postsecondary Education*, pages 279–86, 2022.

20 Young-Suk Grace Kim, Christopher Schatschneider, Jeanne Wanzek, Brandy Gatlin, and Stephanie Al Otaiba. Writing evaluation: Rater and task effects on the reliability of writing scores for children in grades 3 and 4. *Reading and writing*, 30:1287–1310, 2017.

21 Holly Kosiewicz, Cristián Morales, and Kalena E. Cortes. The "missing English learner"' in higher education: How identification, assessment, and placement shape the educational outcomes of English learners in community colleges. In *Higher Education: Handbook of Theory and Research: Volume 39*, pages 1–55. Springer, 2023.

22 Josh Lederman. Validity and racial justice in educational assessment. *Applied Measurement in Education*, 36(3):242–254, 2023. doi:10.1080/08957347.2023.2214654.

23 Bruce W Lee and Jason Hyung-Jong Lee. Prompt-based learning for text readability assessment. In *In Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023)*, pages 1–19. Toronto, Canada. Association forComputational Linguistics., 2023.

24 Stephanie Link and Svetlana Koltovskaia. *Automated Scoring of Writing*, pages 333–345. Springer International Publishing, Cham, 2023. doi:10.1007/978-3-031-36033-6_21.

25 Susan Lyons, Maria Elena Oliveri, and Mya Poe. A framework for enacting equity aims in assessment use: A justice-oriented approach. In *Culturally Responsive Assessment in Classrooms and Large-Scale Contexts*, pages 88–105. Routledge, 2025.

26 Ross Markle. Redesigning course placement in service of guided pathways. *Educational Considerations*, 50(2):8, 2025.

27 Michael Matta and Narmene Hamsho and. Consequences of response formats on racial and ethnic bias and fairness in writing assessments. *School Psychology Review*, 0(0):1–14, 2025. doi:10.1080/2372966X.2025.2462984.

28 Danielle S McNamara, Yasuhiro Ozuru, Arthur C Graesser, and Max Louwerse. Validating CoH-Metrix. In *Proceedings of the 28th annual Conference of the Cognitive Science Society*, pages 573–578, 2006.

29 Neil Murray and Gerard Sharpling and. What traits do academics value in student writing? Insights from a psychometric approach. *Assessment & Evaluation in Higher Education*, 44(3):489–500, 2019. doi:10.1080/02602938.2018.1521372.

30 Jessica Nastal. Beyond tradition: Writing placement, fairness, and success at a two-year college. *Journal of Writing Assessment*, 12(1), 2019.

31 Jessica Nastal and Kris Messer. Afterword: Finding the right note in writing placement. *Journal of Writing Assessment*, 18(1), 2025.

32 Mari Nygård and Anne Kathrine Hundal. Features of grammatical writing competence among early writers in a Norwegian school context. *Languages*, 9(1):29, 2024.

33 María Elena Oliveri, René Lawless, and Robert J. Mislevy. Using evidence-centered design to support the development of culturally and linguistically sensitive collaborative problem-solving assessments. *International Journal of Testing*, 19(3):270–300, 2019. doi:10.1080/15305058.2018.1543308.

34 Ramona T Pittman, Lynette O'Neal, Kimberly Wright, and Brittany R White. Elevating students' oral and written language: Empowering African American students through language. *Education Sciences*, 14(11):1191, 2024.

35 Mya Poe, Jessica Nastal, and Norbert Elliot. Reflection. An admitted student is a qualified student: A roadmap for writing placement in the two-year college. *Journal of Writing Assessment*, 12(1), 2019.

36 Jennifer Reid, Mahshid Ahmadian, D. Jennings, Anathea Abad Pepperl, and et.al. Saying it aloud: Inclusive teaching statements impact on sense of belonging and engagement. *Journal of College Science Teaching*, 0(0):1–14, 2025. doi:10.1080/0047231X.2025.2487437.

37 Rachael Ruegg. Assessment of written assignments in first-year Humanities and Social Sciences courses: Textual features of academic writing. *Assessment in Education: Principles, Policy & Practice*, 32(1):60–76, 2025. doi:10.1080/0969594X.2025.2467672.

38 Padam Singh. P value, statistical significance and clinical significance. *Journal of Clinical and Preventive Cardiology*, 2(4):202–204, 2013.

39 J Suárez-Álvarez, M Oliveri, A Zenisky, and SG Sireci. Current assessment needs in adult education and workforce development: Summary report (center for educational assessment report no. 998). *Center for Educational Assessment*, 2023. URL: https://createadultskills.org/system/files/ASAP%20Brief%201_Nov%202023_Needs%20Assessment.pdf.

40 Meghan Sweeney and Crystal Colombini. (Re) placing personalis: A study of placement reform and self-construction in mission-driven contexts. *Journal of Writing Assessment*, 17(1), 2024.

41 The College Board. ACCUPLACER Program Manual. (online), 2022. URL: https://secure-media.collegeboard.org/digitalServices/pdf/accuplacer/accuplacer-program-manual.pdf.

42 Rodrigo Wilkens, David Alfter, Xiaoou Wang, Alice Pintard, Anaïs Tack, Kevin P Yancey, and Thomas François. Fabra: French aggregator-based readability assessment toolkit. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 1217–1233, 2022. URL: https://aclanthology.org/2022.lrec-1.130.

43 Sachiko Yasuda. What does it mean to construct an argument in academic writing? a synthesis of English for general academic purposes and English for specific academic purposes perspectives. *Journal of English for Academic Purposes*, 66:101307, 2023.

A Chatbot to Help Promoting Financial Literacy

Davi Silva Eleuterio
UNIAG, Instituto Politécnico de Bragança, Portugal

Pedro Filipe Oliveira
CeDRI, Instituto Politécnico de Bragança, Portugal

Paulo Matos
CeDRI, Instituto Politécnico de Bragança, Portugal

Jorge Manuel Afonso Alves
UNIAG, Instituto Politécnico de Bragança, Portugal

Abstract

Currently, governments and many other institutions have been making significant efforts to promote financial literacy. However, a considerable portion of the population still lacks basic financial knowledge, highlighting the need for updated strategies to enhance financial education. In today's digital world – where people often search for quick and convenient solutions – the development of a reliable and intelligent chatbot to answer questions related to financial concepts and decision-making can be very beneficial. This paper proposes the implementation of an automated web scraper to extract content from a trustworthy financial education website with plenty of useful concepts about finances, using this collected data to develop a chatbot which provides accurate and helpful responses to users. The solution is built using technologies such as Streamlit, Langchain, and OpenAI.

2012 ACM Subject Classification Applied computing → Document analysis; Computing methodologies → Information extraction; Computing methodologies → Machine learning

Keywords and phrases chatbot, financial literacy, web scraper, LLM, RAG

Digital Object Identifier 10.4230/OASIcs.SLATE.2025.7

Funding This work was supported by national funds through FCT/MCTES (PIDDAC): CeDRI, UIDB/05757/2020 (DOI: 10.54499/UIDB/05757/2020);
and SusTEC, LA/P/0007/2020 (DOI: 10.54499/LA/P/0007/2020).
This work was supported by national funds through FCT/MCTES (PIDDAC): UID/04752 – Applied Management Research Unit (UNIAG).

1 Introduction

Rehman and Mia [8] define financial literacy as the combination of understanding, abilities, and confidence that enables individuals to make effective financial decisions. It includes comprehending core financial concepts and utilizing them practically. But unfortunately, a significant portion of the population lacks this fundamental financial understanding, contributing to difficulties in managing debt and resulting in over-indebtedness.

In parallel with the growing digitalization of the financial world, the ways that people access information are evolving rapidly. This change highlights the importance of using new technologies to enhance financial literacy, particularly among younger generations. Traditional approaches involving complex texts or extensive research are often less utilized by individuals accustomed to getting instant information from sources such as artificial intelligence (AI) and social media short videos.

Chatbot technology has been used across many fields of knowledge and numerous articles address its successfulness in conducting quick research and providing trustworthy responses. The ability of chatbots to be integrated into popular social media platforms can also highly increase the use of the tool, making it convenient for the user [7].

We propose that a specialized financial chatbot can effectively solve the financial literacy gap by providing accessible and engaging financial education. By delivering accurate information through simple, conversational interactions, such a tool can empower users to make informed decisions, raise awareness about credit risks, and build essential financial knowledge.

This paper explores the design and potential of a financial chatbot specifically built to leverage a reliable data source. Our aim is to demonstrate how providing accurate and practical financial knowledge through user-friendly conversations can reduce the incidence of factual inaccuracies and misleading information often generated by AI, thereby significantly improving users' understanding of financial concepts.

2 State of the Art

Artificial intelligence has experienced much momentum in recent years due to the advancements in hardware and software technologies, specially after the release of ChatGPT in 2022. This paradigm of Generative AI, which leverages advancements in Natural Language Processing (NLP) and is frequently powered by massive Large Language Models (LLMs), has enabled the creation of highly interactive and versatile applications. Among these, advanced chatbots capable of understanding and generating human-like text represent a significant area of development and application [5].

(a) Definition of Chatbot

A chatbot is an AI system and a Human-computer Interaction (HCI) model, which uses NLP and sentiment analysis to communicate in human language by text or oral messages with humans or other chatbots. Interactive agents, artificial conversation entities, smart bots and digital assistants are examples of chatbots among the internet, acting as a powerful tool along many applications, such as education, business, e-commerce, healthcare and many others [1].

(b) Brief History of Chatbots

The development of chatbot agents draws upon foundational ideas from early computational and mathematical concepts. One early mathematical model relevant to sequence prediction, a key element in generating responses, was the Markov Chain, developed by Russian mathematician Andrey Markov in 1906. This statistical model for predicting random sequences has been utilized in machine learning fields for tasks such as autocomplete and next-word prediction for many years.

Another pioneer milestone regarding machine intelligence capable of human-like interaction was the Turing Test, proposed by Alan Turing in 1950. Turing, often regarded as a father of theoretical computer science and artificial intelligence, introduced this test to assess a machine's ability to exhibit intelligent behaviour indistinguishable from that of a human. The test involves a human interrogator conversing separately with a hidden machine and a hidden human. The machine passes if the interrogator cannot reliably determine which is the machine. The Turing Test significantly influenced subsequent artificial intelligence research and spurred efforts towards creating machines capable of natural language conversation [12].

In recent times, the most impactful development in the history of chatbots has been the emergence of systems like ChatGPT, powerfully demonstrating the capabilities of Large Language Models (LLMs), such as GPT-3.5 and GPT-4 [4]. Trained on massive datasets, these models achieve robust language skills, enhanced reasoning, and strong contextual understanding, enabling coherent multi-turn dialogues. ChatGPT, in particular, made this cutting-edge technology widely accessible for public interaction and evaluation.

Underpinning these capabilities is typically the Transformer neural network architecture [11]. Their effectiveness stems from a two-stage training process: initial unsupervised pre-training on vast text to build general knowledge, followed by supervised fine-tuning on dialogue data to hone conversational performance. This methodological approach, leveraging the Transformer's design, is fundamental to their sophisticated natural language generation, with research actively analysing current advancements and trends [9].

(c) Use of Chatbots in Education

AI has been used in the domain of education for over 40 years in different shapes and forms, supporting school administration, teachers and students in different applications [2]. Educators can use them for developing instructional materials and assessments, although responsible application is essential to encourage critical thinking among students. For learners, these tools can promote equity through more accessible information and aid in providing effective learning strategies adapted to diverse preferences [2]. They can also assist with assessing student submissions and suggesting pedagogical improvements. However, it is important to recognize that the current iteration of tools like ChatGPT still has functional limitations, may contain factual inaccuracies, and cannot replicate humans' ability to provide truly differentiated, specific instruction for every student [5].

Research indicates that effective student learning outcomes are strongly linked to personalized support, while insufficient individual attention can hinder academic progress. Methods like micro-learning have been shown to alleviate student fatigue [10] and improve material retention, contributing positively to comprehension, skill development, and overall academic performance. Within this context, chatbots are suggested as valuable tools for e-learning environments. They can serve as interactive tutors, managing student inquiries and providing feedback, and potentially facilitate communication with families regarding a child's learning. By instantly responding to common questions, chatbots enhance the accessibility and comfort for students seeking assistance, making learning more engaging [6].

However, the widespread adoption of these tools also introduces challenges, notably concerns about student misuse leading to academic integrity issues [3]. In response, educational institutions are implementing diverse strategies to prevent such misconduct. Approaches range from simple prohibition of certain tools to revising and updating existing policies. For example, several traditional universities, including some within the UK's Russell Group, classify the unauthorized use of AI bots as academic misconduct. Furthermore, adapting assessment methods is recommended, such as designing tasks less susceptible to AI assistance by incorporating unique content, or returning to traditional formats like written exams instead of only computer-based testing [3].

3 Materials and Methods

3.1 Data Sources and Tools

The study used financial articles obtained from the website *todoscontam.pt*, which is an initiative from the government of Portugal to promote financial literacy through the National Plan for Financial Education. Since these articles are normally unstructured and contain both textual and visual information, a method to automatically extract the useful data was required. The main tools used in this paper include:

- The pages and subpages from the website *todoscontam.pt*
- Python-based programming for web scraping and chatbot development
- OpenAI's ChatGPT for interpreting and analysing financial information content.
- LangChain and Pinecone (vector database) for Retrieval-Augmented Generation (RAG).
- Streamlit framework for building an interactive web interface.

3.2 Methodology

This project was structured to achieve two main objectives:

- Extract and storage data from a reliable website containing foundational financial concepts.
- Implement a chatbot which answer questions about finances using the data extracted as basis.

To address these objectives, we implemented a web scraper using python programming and the chatbot itself:

(a) Web Scraper

One of the primary objectives of this chatbot is to ensure reliability. Because the information provided by large language models (LLMs) is not always fully trustworthy, it is crucial to source data from reliable and verifiable sources. To implement that, we manually collected all the URLs of useful articles regarding financial literacy from the website *todoscontam.pt*. Then, the web scraper is structured as follows:

- Selection of articles: the pages selected from *Todos Contam* website are only the ones which provide financial concepts or decision-making information, ignoring other pages containing non-relevant content, for example pages regarding contact, about the website and others. The URLs collected were inserted in a tuple containing the URL and the destination path. This path and sub paths are respectively the theme and sub themes of the article.
- Downloading and filtering content: for each URL iterated, the system downloads the Hyper-text Markup Language (HTML) content of the article. This HTML code contains many unwanted information, such as navigation bar, tab bar, footer content and its own HTML tags. At this step, every useless information is removed, and the final article text is finally converted to a markdown file.
- File and folder management: in this process, the title of the article is determined as the name of the file, and the folder where the file is stored is named with the theme of the article. A total of 222 documents were stored, with an average size of 2.75 kilobytes.

The diagram presented at figure 1 illustrates all the functioning of the web scraper and summarize each step.

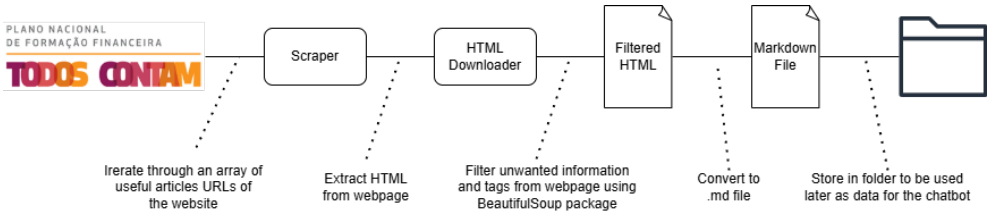


Figure 1 Web scraper.

(b) Chatbot Development

Following the extraction of financial literacy articles in Markdown format, a specialized chatbot was developed to provide users with information derived solely from this corpus. The chatbot leverages a Retrieval-Augmented Generation (RAG) architecture, combining the retrieval capabilities of a vector database with the generative power of a Large Language Model (LLM). The implementation utilizes Python, primarily employing the Langchain framework for orchestrating the RAG pipeline, Streamlit for the user interface, and Pinecone as the vector database.

a. Core components and configuration: the system is built upon several key libraries. Langchain-openai provides interfaces for OpenAI's models, specifically using ChatOpenAI with the gpt-4.1-nano model for response generation and OpenAIEmbeddings with the text-embedding-3-small model (1536 dimensions) for creating text embeddings. Langchain-pinecone facilitates interaction with the Pinecone vector database. Streamlit's caching mechanism (@st.cache_resource) is employed to efficiently manage resource-intensive objects like the LLM, embedding models, and the Pinecone connection. Figure 2 illustrates the chatbot functioning.

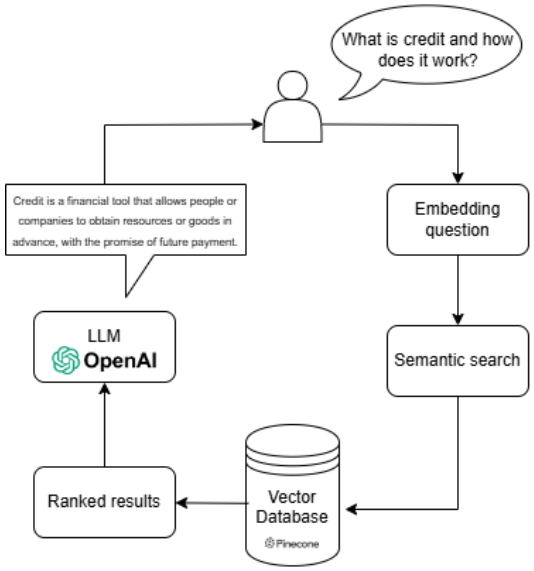


Figure 2 Chatbot operating architecture.

b. Data preparation and indexing: the initial step involves loading the Markdown documents from the specified directory, including subdirectories. The DirectoryLoader from Langchain, configured with UnstructuredMarkdownLoader, handles this process. A crucial preprocessing step adds metadata to each document, identifying the financial theme based on the name of the subfolder from which the document originated.

c. Retrieval-augmented generation pipeline: the core question-answering functionality is implemented using Langchain's RetrievalQA chain. This chain integrates the LLM with a retriever built upon the Pinecone vector store. The retriever is configured to perform similarity searches and retrieve the top 5 most relevant document chunks based on the vectorized user query.

   A custom PromptTemplate is employed to guide the LLM's response generation. This template incorporates:

   - A static context defining the chatbot's persona as a helpful financial assistant based on *todoscontam.pt* content, instructing it to use only retrieved information, state when information is unavailable, and avoid fabrication.
   - A placeholder for the context retrieved from the vector database.
   - A placeholder for the user's question.

   The RetrievalQA chain is configured to insert all retrieved text chunks directly into the context placeholder of the prompt. The chain is also set to return the source documents used for generation.

d. User interface and interaction: a web-based user interface is provided using Streamlit. It displays a chat interface where users can interact with the chatbot. Chat history is maintained using Streamlit's session state. When a user submits a query, the query is processed by the system. The response generated by the LLM is displayed, along with an expandable section listing the source documents (filename and theme) that the retriever identified as relevant for answering the query.

4 Results

To assess the chatbot's performance in an initial phase, a set of 48 questions spanning six key financial literacy themes was curated, reflecting common user queries in this domain. These themes, derived from the structure of the *Todos Contam* portal, include:

1. Personal and Family Budgeting
2. Savings and Basic Investment
3. Credit and Debt
4. Essential Banking Products
5. Basic Insurance
6. Consumer Rights and Duties in Finance

Each question was presented to the developed chatbot, and the corresponding answer was recorded and analysed. The evaluation focused on the chatbot's ability to adhere to its core instruction: provide accurate and relevant answers based solely on the retrieved document context from the scraped articles, and explicitly state when the requested information was not available within that knowledge base. Answer quality was assessed based on correctness relative to the source material, relevance to the question, and appropriate handling of knowledge gaps.

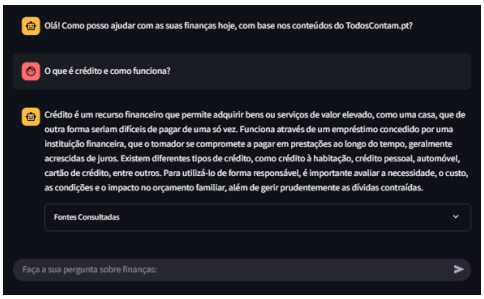


Figure 3 Example of interaction 1.

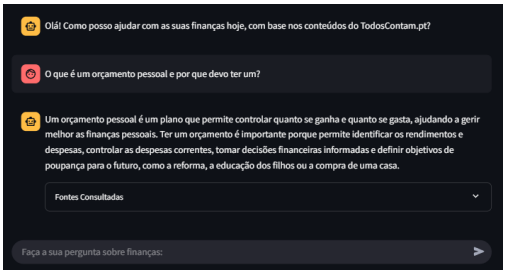


Figure 4 Example of interaction 2.

4.1 Findings

The analysis of the 48 question-answer pairs revealed a high degree of adherence to the provided knowledge base and instructions. The chatbot consistently utilized the information present in the scraped articles to formulate responses.

Table 1 summarizes the performance across the different themes. An answer was considered successfully addressed if it either provided a correct response based on the source material or correctly identified that the information was not present in the knowledge base.

Table 1 Summary of Chatbot Performance by Theme.

Theme	Correctly Answered	Identified as Missing
1. Budgeting	8	0
2. Savings & Investment	7	1
3. Credit & Debt	6	2
4. Banking Products	7	1
5. Insurance	8	0
6. Consumer Rights	6	2
Total	**42**	**6**

Overall, the chatbot successfully addressed all 48 questions according to the evaluation criteria. In 42 cases, it provided answers directly derived from the scraped content. In the remaining 6 cases, it correctly identified that the specific information requested was either entirely absent from the source documents or that the source documents lacked the requested level of detail, thereby demonstrating its ability to recognize the boundaries of its knowledge base. This corresponds to a 87.5% success rate in information retrieving and 100% success rate in terms of adhering to the operational instructions and leveraging the provided corpus.

4.2 Discussion

The results indicate that the RAG-based chatbot effectively functions as an information retrieval and summarization tool for the specific knowledge base derived from *Todos Contam*. Its main strength lies in its fidelity to the reliable source material, ensuring that users receive information aligned with the content curated by the Bank of Portugal and the National Plan for Financial Education. The explicit prompting to rely solely on retrieved documents and acknowledge gaps proved successful. It was also observed that the chat provides responses with consistent structure and content when asked the same question multiple times, varying only in wording.

However, the chatbot's primary limitation is its dependence on the scraped content. It cannot answers questions outside the scope of the articles available on *Todos Contam* or provide broader financial context, comparisons with products not mentioned, or real-time market data. While it correctly identifies knowledge gaps, it cannot bridge them without additional information sources. Furthermore, the quality and comprehensiveness of the answers are inherently limited by the quality and depth of the original articles.

4.3 Proposed Solutions and Future Work

To enhance the chatbot's utility and address its limitations, several avenues for future work are proposed:

- **Expand Knowledge Base:** Incorporate content from other authoritative Portuguese financial sources, such as the websites of the CMVM (Securities Market Commission), ASF (Authority for the Supervision of Insurance and Pension Funds), the Bank of Portugal's main site, and relevant government portals (e.g., related to taxes or social security). This would broaden the range of topics the chatbot can address.
- **Incorporate Structured Data:** Explore adding structured data (e.g., current interest rates for specific products, calculators) via APIs or separate databases to provide more dynamic and tool-like functionalities.
- **User Evaluation:** Conduct user studies with the target audience to gather qualitative feedback on the chatbot's usability, understandability, and perceived usefulness in improving financial literacy. This feedback can guide further development iterations.

By expanding the knowledge base and potentially refining the RAG pipeline, the chatbot's potential as a comprehensive financial literacy tool can be significantly enhanced.

5 Conclusions

Financial literacy remains a critical challenge, with many individuals lacking the necessary knowledge to make informed financial decisions in an increasingly complex digital world. Traditional educational methods often struggle to engage audiences accustomed to readily available, interactive digital content. This paper addressed this gap by proposing and implementing a specialized chatbot designed to enhance financial education in Portugal.

We presented the development process, which involved automated web scraping of reliable content from the *Todos Contam* website, followed by the construction of a chatbot using a Retrieval-Augmented Generation (RAG) architecture. Leveraging tools like Langchain, OpenAI models, Pinecone, and Streamlit, we created an interactive system capable of answering user queries based on the curated financial knowledge base.

The evaluation demonstrated the chatbot's effectiveness within its defined scope. It successfully addressed a diverse set of 48 questions across key financial themes, either by providing accurate information derived strictly from the source material or by correctly identifying when the requested information was unavailable in its knowledge base. This highlights the system's fidelity to the reliable source and its adherence to operational instructions, crucial aspects for building trust in AI-driven financial tools.

In conclusion, this work demonstrates the viability of using a RAG-based chatbot, grounded in verified information sources, as a tool to promote financial literacy. By providing accessible, reliable, and interactive financial education, such systems hold significant potential to empower individuals and contribute to better financial well-being. The emphasis on using curated, trustworthy data sources is paramount when developing AI applications for sensitive domains like personal finance.

References

1 Eleni Adamopoulou and Lefteris Moussiades. Chatbots: History, technology, and applications. *Machine Learning with Applications 2*, 2021.

2 Ilaha Ashrafova. Education and chatbots: New opportunities for teachers and students. *Journal of Azerbaijan Language and Education Studies*, 2(2), 2025.

3 Ali Ateeq, Mohammed Alzoraiki, Marwan Milhem, and Ranyia Ali Ateeq. Artificial intelligence in education: implications for academic integrity and the shift toward holistic assessment. *Frontiers in Education*, 9, 2024.

4 Chiranjib Chakraborty, Soumen Pal, Manojit Bhattacharya, Snehasish Dash, and Sang-Soo Lee. Overview of chatbots with special emphasis on artificial intelligence-enabled ChatGPT in medical science. *Sec. Medicine and Public Health*, 2023.

5 Faisal Kalota. A primer on generative artificial intelligence. *Education Sciences*, 2024.

6 Chee Ken Nee, Mohd Hishamuddin Abdul Rahman, Noraffandy Yahaya, Nor Hasniza Ibrahim, Rafiza Abdul Razak, and Chie Sugino. Exploring the trend and potential distribution of chatbot in education: A systematic review. *International Journal of Information and Education Technology*, 13(3), 2023.

7 Reshawn Ramjattan, Patrick Hosein, and Nigel Henry. Using chatbot technologies to help individuals make sound personalized financial decisions. *2021 IEEE International Humanitarian Technology Conference (IHTC)*, 2021.

8 Khurram Rehman and Md Aslam Mia. Determinants of financial literacy: a systematic review and future research directions. *Future Business Journal*, 10, 2024.

9 Chen Renzhang and Zhao Haixia. AI chatbot research: A bibliometric analysis of advancements and trends. *Journal of Macau University of Science and Technology (Humanities and Social Sciences)*, 19, 2025.

10 M. S. Shail. Using micro-learning on mobile applications to increase knowledge retention and work performance: A review of literature. *Cureus*, 11(8):e5307, 2019. doi:10.7759/cureus.5307.

11 Emma Yann Zhang, Adrian David Cheok, Zhigeng Pan, Jun Cai, and Ying Yan. From turing to transformers: A comprehensive review and tutorial on the evolution and applications of generative transformer models. *Sci*, 2023.

12 Michal Černý. The history of chatbots: the journey from psychological experiment to educational object. *Journal of Applied Technical and Educational Sciences*, 12(3), 2022.

An Architecture for Composite Combinatorial Optimization Solvers

Khalil Chrit
NOVA LINCS, University of Évora, Portugal

Jean-François Baffier
IIJ Research, Tokyo, Japan

Pedro Patinho
NOVA LINCS, University of Évora, Portugal

Salvador Abreu
NOVA LINCS, University of Évora, Portugal

Abstract

In this paper, we introduce elements for `MoSCO`, a framework for building hybrid metaheuristic-based solvers from a collection of reusable base components. The framework is implemented in Julia and provides a modular architecture for composing solvers through a pipeline-based approach. The modular design of `MoSCO` supports the creation of reusable components and adaptable solver strategies for various Constraint Satisfaction Problems (CSPs) and Constraint Optimization Problems (COPs). We validate `MoSCO`'s utility through practical examples, demonstrating its effectiveness in reconstructing established metaheuristics and enabling the creation of novel solver configurations. This work lays the foundation for future developments in automated solver construction and parameter optimization.

2012 ACM Subject Classification Software and its engineering → Constraint and logic languages; Software and its engineering → Formal language definitions; Theory of computation → Discrete optimization

Keywords and phrases Hybrid Metaheuristics, DSL

Digital Object Identifier 10.4230/OASIcs.SLATE.2025.8

Funding This work was partially supported by the Portuguese Foundation for Science and Technology under PhD fellowship UI/BD/153047/2022 and strategic project UID/04516 (NOVA LINCS).

1 Introduction

Metaheuristics are a class of problem-independent algorithms known for their ability to efficiently find approximate solutions to hard combinatorial optimization problems, e.g. traveling salesman problem (TSP) [1], quadratic assignment problem (QAP) [8] and vehicle routing problem (VRP) [23]. Metaheuristics are search techniques that navigate the search space adeptly, often producing near-optimal solutions to problems that are otherwise intractable. Despite their success, developing and fine-tuning metaheuristic solvers is a challenging problem in itself, with difficulties in guiding the stochastic process and the high sensitivity to parameter changes which impact intensification and diversification of the search process. Designing and implementing effective metaheuristic solvers requires domain expertise and much trial-and-error, making the whole process time-consuming and burdensome [3, 22]. Frequently, a particular metaheuristic will be well suited to a particular problem or problem instance, but poorly so for other problems or even just instances. This situation led to the development of *hybrid metaheuristics*, which combine multiple approaches in order to more efficiently solve a wider class of problems and instances [18].

We take a step back and propose a modular framework in which to construct hybrid metaheuristic-based solvers called MoSCO. The goal is mainly to allow for an abstracted description of the structure of a *hybrid* metaheuristic solver, which is used to synthesize a functioning problem-specific solver, by combining parts of existing solvers which are connected in a network which conducts the process of deriving a stream of solutions for the original problem, meaning to reach the optimum.

We validate the proposed architecture through a Julia-based implementation, showcasing MoSCO's potential to reconstruct well-known reference metaheuristics by applying it to a well-known problem domain.

The remainder of this paper is structured as follows: in section 2 we discuss previous approaches to the problem of describing metaheuristic solvers and proceed, in section 3, to introduce the general architecture of MoSCO. Section 4 is used to describe a textual representation for the language and provide examples. In section 5, we outline the structure of our prototype developed in Julia and finally, in section 7 we assess our work and set possible lines for future developments.

2 Related Work

2.1 Parallel-Oriented Solver Language (POSL)

Reyes-Amaro et al. [21] proposed the Parallel-Oriented Solver Language (POSL), a framework for building interconnected meta-heuristic based solvers that work in parallel. POSL focuses on sharing not only information but also behaviors among solvers, allowing for solver modifications during runtime. While POSL shares similarities with our proposed solver architecture, our work emphasizes the use of a Domain Specific Language (DSL) and incorporates meta-learning algorithms for optimizing solver parameters and architecture.

2.2 ParadisEO

Cahon et al. [7] introduced ParadisEO, a white-box object-oriented software framework dedicated to the flexible design of metaheuristic algorithms for combinatorial optimization. ParadisEO provides a broad range of metaheuristic components and encourages code reusability, allowing users to design their own solvers by assembling and customizing existing components. Our proposed architecture shares the idea of modularity and code reusability with ParadisEO but also emphasizes the use of a DSL to connect components and incorporates meta-learning algorithms for optimization.

2.3 jMetal

Durillo and Nebro [10] developed jMetal, a Java-based framework for multi-objective optimization with metaheuristics. jMetal provides a rich set of components and tools for designing, experimenting with, and benchmarking metaheuristic algorithms. While jMetal focuses on multi-objective optimization, our proposed architecture is designed for general combinatorial optimization problems and emphasizes the use of a DSL and meta-learning algorithms for optimizing solver parameters and architecture.

In summary, several approaches have been proposed to facilitate the development of metaheuristic-based solvers for combinatorial optimization problems. Our proposed modular solver architecture builds upon existing work by providing a problem-independent architecture design, incorporating a Domain Specific Language for connecting components, with plans to incorporate meta-learning algorithms to automatically optimize solver parameters and architecture.

3 Modular Solver Architecture

Here, we provide a detailed description of the various elements that make up the proposed architecture. Figure 1 shows the input/output and parameters of the operator, which forms the basic building block of our architecture.

3.1 Configuration Streams

Within the solver, the processing of solution candidates follows defined pathways, formally represented as a directed graph $G = (V, E)$, where V is the set of operators and E represents the connections between them, which we call *Configuration Streams*. Each edge $e \in E$ corresponds to a Configuration Stream that transmits configurations (solution candidates) from one operator to another.

▶ **Definition 1.** *A* Configuration Stream S *is a communication channel between operators that manages the transmission of configurations, defined as:*

$$S : \mathcal{O}_{source} \to \mathcal{O}_{destination} \tag{1}$$

where $\mathcal{O}_{source}$ *and* $\mathcal{O}_{destination}$ *are operators, and* S *transmits a set of configurations* $C = \{c_1, c_2, \ldots, c_n\}$*, where each* $c_i \in \Omega$ *represents a potential solution within the solution space* Ω*.*

- *When* $|C| = 1$*, the stream transmits exactly one configuration at a time. Typically used in trajectory-based metaheuristics such as Simulated Annealing [15] or Tabu Search [11].*
- *When* $|C| > 1$*, the stream manages a set of configurations sequentially. This supports population-based metaheuristics like Genetic Algorithms [12] or Particle Swarm Optimization [14], where multiple solution candidates evolve concurrently.*

3.2 Operators

Operators are the functional components that transform configurations as they flow through the solver pipeline. Each operator implements a specific algorithmic behavior that forms part of the overall search process.

▶ **Definition 2.** *An* operator $\mathcal{O}$ *is a transformation function defined as:*

$$\mathcal{O} : \mathcal{I}_{\mathcal{O}} \to \mathcal{O}_{\mathcal{O}}$$

Where:

- $\mathcal{I}_{\mathcal{O}}$ *represents the specific* "input domain" *for operator* $\mathcal{O}$*. This domain could be drawn from various sets depending on the operator, including the problem specification space* Θ *(for the* init *operator), the solver representation space* $\mathcal{S}$ *(for the* gen *operator specifically), the configuration space* $\mathcal{A}$*, the power set* $\mathcal{P}(\mathcal{A})$*, or Cartesian products like* $\mathcal{A} \times \mathcal{A}$*.*
- $\mathcal{O}_{\mathcal{O}}$ *represents the specific* "output range" *for operator* $\mathcal{O}$*, typically being* $\mathcal{A}$*, or* $\mathcal{P}(\mathcal{A})$*, and* $\mathcal{S}$ *for the* init *operator.*
- $\mathcal{A}$ *represents the* "set of possible configurations" *(i.e., the type of a single solution candidate) within the solution space* Ω*.*

Operators manipulate configurations, often taking elements of type $\mathcal{A}$ *or sets* $\mathcal{P}(\mathcal{A})$ *as the input and/or producing them as output.*

Next, we present the core operators within the architecture:

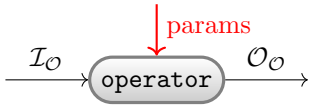


Figure 1 Graphical representation of an operator with input $\mathcal{I}_\mathcal{O}$, output $\mathcal{O}_\mathcal{O}$, and configurable parameters.

Init: Responsible for setting up the problem domain, encoding the CSP model into a solver-compatible representation (which is then used by other components to generate and evaluate solution candidates), and defining the search space and the domain of the decision variables, the objective function, constraints, and other relevant information (meta-data). Formally, the Init operator can be defined as:

$$\mathcal{O}_{\text{Init}} : \Theta \to \mathcal{S} \tag{2}$$

It encodes the problem instance Θ into a solver-compatible representation $\mathcal{S}$, which includes definitions for V (Variables), D (Domains), $f : \mathcal{A} \to \mathbb{R}$ (objective function), and the set of constraints $\mathcal{C}$ that implicitly define the solution space Ω. The Init operator can be parameterized with r (random) or h (heuristic) initialization strategies (illustrated in Figure 2).

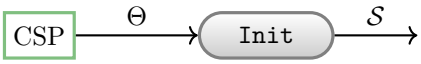


Figure 2 The Init operator transforms a problem instance into a solver-compatible representation.

Generate: Responsible for instantiating the initial set of solution candidates within the defined search space. This operator transforms the abstract problem definition into concrete configuration instances that serve as the starting point for the optimization process. Formally, the generate operator can be defined as:

$$\mathcal{O}_{\text{Generate}} : \mathcal{S} \to \mathcal{P}(\mathcal{A}) \tag{3}$$

Where $\mathcal{P}(\mathcal{A})$ denotes the set of configurations, allowing for the generation of either a singleton set (single configuration) or a population of configurations (multiple candidates). The operator's behavior can be parameterized with:

- $n = |\mathcal{P}(\mathcal{A})|$: number of generated configurations.
- σ: initialization strategy.
  - σ_r: Random, assigns values to decision variables according to a uniform probability distribution over their respective domains.
  - σ_h: Heuristic, employs problem-specific knowledge to generate configurations with potentially higher initial quality.

▶ Note. The parameter n doesn't determine whether the subsequent search process follows a trajectory-based or a population-based approach (this is determined by the compilator, which we'll describe in section 4.2). It only determines the number of configurations to generate.

Neighbourhood: Performs moves in the search space to generate neighboring solutions. It supports various move types including swap, shift, block moves or any other move that can be defined for the problem at hand.

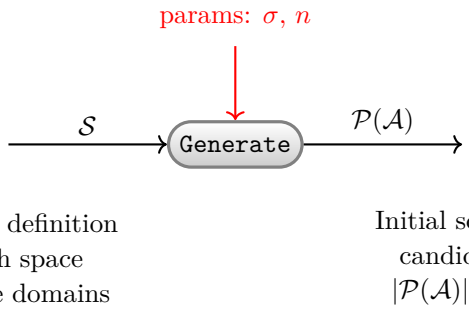


Figure 3 The Generate operator creates initial solution candidates from the problem representation.

A Neighbourhood operator, defined as: $\mathcal{O}_{\text{Neighbourhood}} : \mathcal{A} \times \mathcal{M} \to \mathcal{P}(\mathcal{A})$
is a function that, given a configuration $c \in \mathcal{A}$ and a move type $m \in \mathcal{M}$, computes the corresponding set of neighbours, denoted by $N(c, m) \in \mathcal{P}(\mathcal{A})$.
Where $N(c, m)$ is the set of neighbours of configuration c generated using move type $m \in \mathcal{M} = \{\text{swap}, \text{insert}, \text{delete}, \text{replace}, \cdots\}$. The specific neighbours generated depend on the implementation associated with the move type m.

When a stream delivers a set of configurations $C = \{c_1, c_2, \ldots, c_k\}$ to this operator, the intended architectural behavior is that the operator is implicitly applied to each configuration in the set. This would conceptually produce a collection of neighbour sets $\{N(c_1, m), N(c_2, m), \ldots, N(c_k, m)\}$.

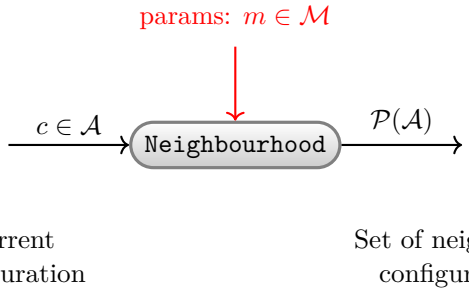


Figure 4 The Neighbourhood operator generates a set of configurations in the vicinity of the current solution.

Accept: This operator determines whether to accept or reject a candidate configuration $c_{\text{candidate}}$ based on the current configuration c_{current} and a chosen acceptance criterion a. Formally, it is a function $\mathcal{O}_{\text{Accept}} : \mathcal{A} \times \mathcal{A} \times \mathcal{Q} \to \mathcal{A}$ defined as:

$$\mathcal{O}_{\text{Accept}}(c_{\text{current}}, c_{\text{candidate}}, a) = \begin{cases} c_{\text{candidate}} & \text{if criterion } a \text{ evaluates to true} \\ c_{\text{current}} & \text{otherwise} \end{cases} \tag{4}$$

where $a \in \mathcal{Q}$ represents the selected acceptance criterion from a set of possible strategies. The acceptance criteria in $\mathcal{Q}$ include (but not limited to):

Always: The candidate is always accepted: $a_{\text{always}}(c_{\text{current}}, c_{\text{candidate}}) = \text{true}$.

Improvement: Accepted only if it improves the current solution: $a_{\text{improv}}(c_{\text{current}}, c_{\text{candidate}}) = f(c_{\text{candidate}}) < f(c_{\text{current}})$.

Probabilistic: Acceptance with a fixed probability p: Let $u \sim \text{Uniform}(0,1)$, then $a_{\text{probability}}(c_{\text{current}}, c_{\text{candidate}}, p) = u < p$.

Temperature-based (Simulated Annealing): Accepts improving solutions, or worsening ones with a probability dependent on a temperature parameter T. Let $\Delta f = f(c_{\text{candidate}}) - f(c_{\text{current}})$: $a_{\text{temperature}}(c_{\text{current}}, c_{\text{candidate}}, T) = (\Delta f < 0) \vee (u < \exp(-\frac{\Delta f}{T}))$

▶ Note.

- The described acceptance criteria assume a minimization problem where lower objective function values are preferred.
- Selection strategies like those used in *Tabu Search*, which involve evaluating multiple non-forbidden neighbors and selecting the best, are typically implemented in this framework through a sequence of operators (e.g., Neighbourhood, filtering based on the Tabu list, and selection using a dedicated operator or a `MUX` connector configured with a 'best' strategy) rather than as a single criterion within the Accept operator.

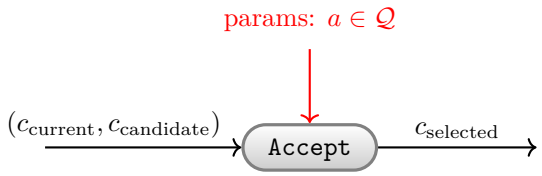


Figure 5 The Accept operator decides whether to accept a candidate solution based on the defined strategy.

Loop: This operator repeats the execution of an internal operator or sequence of operators until a specific termination condition is met.
Formally, it is represented as $\mathcal{O}_{\text{Loop}} : (\mathcal{A} \to \mathcal{A}) \times \mathcal{T} \times \mathcal{A} \to \mathcal{A}$.
Where $c^{(0)} = c_{\text{initial}}$, and $c^{(i)} = \text{Op}(c^{(i-1)})$ for $i \geq 1$.
The loop terminates after k iterations, where k is the smallest index for which the termination condition $\tau \in \mathcal{T}$ is true (the condition's evaluation may depend on the current iteration k, the current configuration $c^{(k)}$, elapsed time, solution history, etc.).
The output of the Loop operator is the final configuration $c_{\text{final}} = c^{(k)}$.
The termination conditions τ supported are: iteration count (i.e. stopping after a fixed number of iterations ($i_{\max}$)), exceeding a specific time limit ($t_{\max}$), convergence (reaching a point where the solution quality has not significantly improved over a certain period, defined by a threshold ϵ), or achieving a desired target objective function value (f_{target}).

▶ Note. Unlike operators that perform a direct transformation on a configuration in a single step, the Loop operator acts as a "control-flow construct", it manages the iterative execution process and maintains states related to termination conditions (e.g., iteration count, solution history).

3.3 Connectors

The connectors are the links between components that allow for a custom flow of data within the solver. They can be used to merge or split configuration streams (enabling parallelization of the search process [16]). Just like the loop operator (although fundamentally different), the

connectors are not operators in the traditional sense, as they are not responsible for performing any transformation on configurations, but rather, only for controlling the execution flow within the solver.

There are two types of connectors:

- **DEMUX**: Split a single configuration stream into multiple streams, each of which can be processed separately. $\mathcal{O}_{\text{DEMUX}} : \mathcal{A} \times \mathbb{N} \times \mathcal{B} \to \mathcal{P}(\mathcal{A})$ is a function that's defined as:

$$\mathcal{O}_{\text{DEMUX}}(c, n, p) = \{c_1, c_2, \ldots, c_n\} \tag{5}$$

  Where:
  - $c \in \mathcal{A}$ is the input configuration
  - $n \in \mathbb{N}$ is the number of output streams
  - $p \in \mathcal{B}$ is a flag indicating whether the streams should be processed in parallel
  - $c_i = c$ for all $1 \leq i \leq n$ (the same configuration is sent to all output streams)

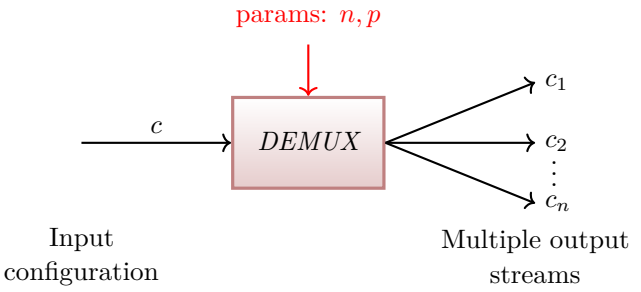


Figure 6 The DEMUX connector splits a single input configuration into multiple output streams.

- **MUX (Multiplexer/Selector)**: This component acts as a selection point, merging multiple candidate configuration streams and potentially comparing them to select a single output configuration. Formally, it can be represented as a function:

$$\mathcal{O}_{\text{MUX}} : \mathcal{P}(\mathcal{A}) \times \mathbb{N} \times \mathcal{S} \to \mathcal{A} \tag{6}$$

  Given a set of incoming candidate configurations $C_{\text{candidates}} = \{c'_1, c'_2, \ldots, c'_n\} \in \mathcal{P}(\mathcal{A})$, and a selection strategy $s \in \mathcal{S}$, and an index $i \in \mathbb{N}$ the MUX outputs a single selected configuration $c_{\text{selected}} \in C_{\text{candidates}}$.

  Indexed Candidate (i): Select the candidate c'_i arriving from the i-th conceptual input stream (this refers to the stream source, not an ordering within the set $C_{\text{candidates}}$).

  Predicate based selection: Select the candidate c'_i where $s(c'_i) = \text{true}$ for $1 \leq i \leq n$. The set $\mathcal{S}$ of selection strategies includes, but is not limited to:
  - **Best**: $s_{\text{best}}(c_i) = f(c_i) = \min_{1 \leq j \leq n} f(c_j)$ for minimization
  - **Random**: $s_{\text{random}}(c_i) = \text{true}$ with probability $\frac{1}{n}$

3.4 Leashed Solvers

Leashed solvers are composite entities within the framework that encapsulate a set of operators to act as a singular unit. Think of a leashed solver as a complete, potentially complex, solver (i.e. a local search algorithm [13], or even another metaheuristic) that is embedded within a larger solver architecture but operates under strict external control - it's kept "on a leash" by the primary solver framework. The key aspects of this concept are:

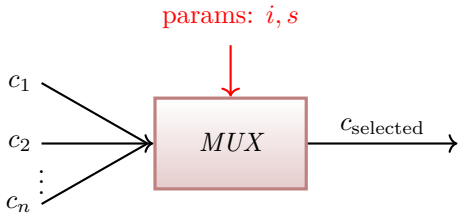


Figure 7 The MUX connector merges multiple input streams into a single output configuration.

- **Encapsulation**: A leashed solver bundles a complete metaheuristic solver, either external or a composite of of `MoSCO` operators (might also include its own loops, state, and logic) into a single reusable entity.
- **Externally Controlled execution**: This is the crucial "leash" part. The enclosing solver dictates when the leashed solver is executed, what input it receives (e.g., a specific configuration to improve), and how long or under what condition it runs.
- **Limited autonomy & resources**: It doesn't run indefinitely or consume unbounded resources. Its execution is typically bounded by parameters like iteration count, time limit, or convergence criteria.
- **Stateful interaction**: Given that the operators are stateless transformations, a leashed solver is expected to maintain its own internal state during execution, but this state might be reset or initialized each time it's called by the main solver.
- **Dictated purpose**: The primary solver uses the leashed solver strategically as a sub-routine for a specific purpose, and not just a generic step (e.g. intensification, diversification, neighbourhood exploration). A leashed solver acts as a stateful and controllable "sub-routine" within the pipeline, offering more complexity than a simple operator, but less independence than running a completely separate solver process.

In essence, leashed solvers allow for arbitrarily complex operations to be bundled and utilized as if they were a singular component. The driving idea is to use existing solver strategies but cap their ability to work towards a solution (hence the "leash" analogy.) Moreover, this design provides a mechanism for modularity and encourages the integration of existing specialized solvers.

4 Domain Specific Language

The elements introduced in the previous sections combine to form hybrid solvers. This may be expressed using a textual representation, which we now expound on. This section introduces the domain-specific language (DSL) meant to describe entire solvers and their structure, and gives a few examples.

4.1 Syntax

The DSL grammar can be formally defined as a 4-tuple $G = (N, T, P, S)$ where:

- N is the set of non-terminal symbols:

$$N = \{\text{Solver}, \text{Expression}, \text{Operator}, \text{Sequence}, \text{Loop}, \text{Parallel}\} \tag{7}$$

- T is the set of terminal symbols, consisting of operator names, parameters, and syntactic delimiters:

- Operator names: `init`, `gen`, `neighbourhood`, `accept`, ...
- Syntactic delimiters: `|`, `*`, `(`, `)`, `[`, `]`, `=`, `→`, ...

- P is the set of production rules, with Solver as the start symbol:

$$\text{Solver} \rightarrow \text{Expression} \tag{8}$$
$$\text{Expression} \rightarrow \text{Operator} \mid \text{Sequence} \mid \text{Loop} \mid \text{Parallel} \tag{9}$$
$$\text{Operator} \rightarrow \text{OperatorName} \mid \text{OperatorName(Parameters)} \tag{10}$$
$$\text{Sequence} \rightarrow \text{Expression} \mid \text{Expression} \tag{11}$$
$$\text{Loop} \rightarrow \text{(Expression)*} \mid \text{(Expression)*(Condition)} \tag{12}$$
$$\text{Parallel} \rightarrow \text{[Expression, Expression, ...]} \tag{13}$$

- S is the start symbol:

$$S = \text{Solver} \tag{14}$$

For future versions, we plan to implement *Conditional Processing*, selecting one of multiple paths based on a condition

$$\text{Conditional} \rightarrow \{\text{Expression, Expression, ...}\} \tag{15}$$

This feature will enable more dynamic solver behavior based on runtime conditions but is not yet implemented in the current version.

Variable Binding

The DSL also supports variable binding to create more complex data flows:

```
solver = init | gen > current_state;
current_state > loop (
neighbourhood | accept )
```

Where "`op > variable_name`" binds the output of an operator "op" to a variable, which can be referenced by subsequent operators. This enables feedback loops and more complex control structures.
An alternative, more explicit syntax would be:

```
solver =
init > v1;
gen < v1 > v2;
loop < v2 (
neighbourhood < current > candidates;
(current, candidates) > accept > current;
)
```

In this notation:

- Each statement ends with a semicolon.
- `op < v1` provides input to an operator.
- `op < v1 > v2` combines both input and output binding.
- `v1 > op > v2` is equivalent to `op < v1 > v2`.
- Multiple inputs can be specified as a tuple: `< (v1, v2) >`

This notation makes the data flow more explicit and enables complex connections between operators.

4.2 DSL Examples

Here we illustrate how common metaheuristic algorithms can be expressed using the DSL.

- **Parallel Solver Branches**

```
solver = init | gen | DEMUX | [
  neighbourhood | accept(criterion=improvement),
  neighbourhood(move_type=swap) | accept
  ] | MUX(strategy=best)
```

  This solver creates two parallel search paths with different neighborhood and acceptance strategies and combines results by selecting the best solution found in either branch.

- **Stochastic Local Search**

```
solver = init | gen(n=1) | loop(
  neighbourhood(move_type=swap) |
  accept(criterion=probability, alpha=0.5)
  )*(it=1000)
```

  This representation captures the essence of Stochastic Local Search:
  1. Initialize the problem.
  2. Generate a single starting solution.
  3. Generate a neighbouring solution using the swap move.
  4. Accept the neighbour if it is better than the current solution, otherwise accept with a certain probability (alpha).
  5. repeat for 1000 iterations.

- **Simulated Annealing**

```
solver = init | gen(n=1) | loop(
  neighbourhood(move_type=swap) |
  accept(criterion=temperature, T0=100, alpha=0.95)
  )*(it=1000)
```

  This specifies:
  1. Initialize the problem.
  2. Generate a single starting solution.
  3. Generate a neighbouring solution using the swap move.
  4. Accept the neighbour based on a temperature schedule (with intial temperature t0 and cooling factor alpha).
  5. repeat for 1000 iterations.

- **Tabu Search**

```
solver = init | gen(n=1) | loop(
  neighbourhood(move_type=swap) |
  accept(criterion=tabu)
  )*(it=1000)
```

  The structure of this solver is the same as the previous examples, with the difference that the accept operator uses the tabu criterion, which translates to a tabu search solver.

▶ Note. When operators do not have parameters explicitly specified, they use their default values. In this case, the `accept(criterion=tabu)` operator uses a default tabu list size of 7.

- **Iterated Local Search**[17]

```
solver = init | gen(n=1) > current_best;
loop (
    current_best >
        perturbation_operator >
        perturbed_sol;
    perturbed_sol >
        local_search_procedure >
        new_solution;
    new_solution >
        accept(criterion=ils) >
        current_best
) * (iterations=1000)
```

In this example, we used the variable binding notation, as well as a leashed solver to encapsulate the local search procedure. `perturbation_operator` and `local_search_procedure` are `leashed solvers`, created from a sequence of basic operators.

4.3 Graphical Representation

The DSL can be visually represented as a directed graph $G_V = (V, E, \lambda)$ where:

- V represents the set of nodes corresponding to operators and connectors
- $E \subseteq V \times V$ represents the set of directed edges corresponding to configuration streams
- $\lambda : V \to \Omega$ is a labeling function that maps nodes to their operational semantics

The mapping between DSL constructs and graphical elements follows the principles expressed in Table 1.

Table 1 Mapping between DSL constructs and their graphical representations.

DSL Construct	Graph Representation
Operator (Op)	Node with label Op
Sequence (Op1 | Op2)	Directed edge from Op1 to Op2
Loop ((Expr)*)	Directed edge from Expr to Loop
Parallel ([Expr1, Expr2, ...])	DEMUX node with multiple outgoing edges, followed by parallel paths, ending with MUX node

The simplest solver follows a linear pipeline from initialization to solution output, as shown in figure 8.

Figure 8 Simple sequential solver.

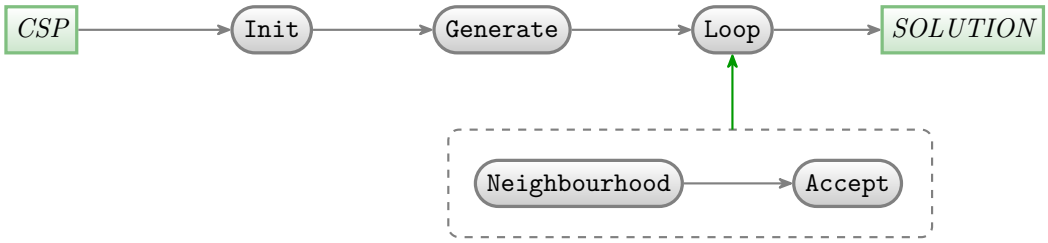


Figure 9 A simple iterative solver using the Loop operator representation.

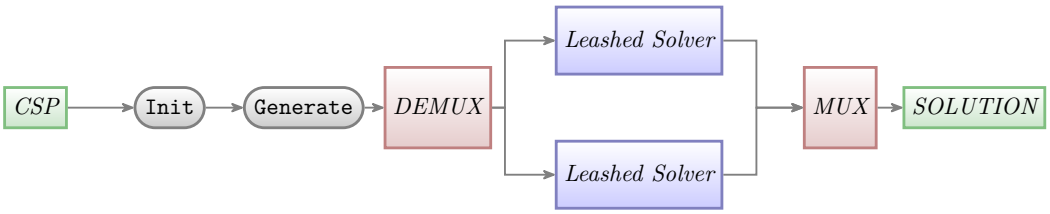


Figure 10 Simple solver with parallel processing paths using leashed solvers.

The graphical representation provides an intuitive visualization of flow of configurations and execution within the solver, making it easier to understand complex solver architectures. This complements the textual DSL by providing a clear mapping between syntactic constructs and their semantic interpretation in terms of computation and data flow. Figures 8, 9, and 10 provide examples of different solver configurations using the graphical notation.

5 Prototype Implementation

To validate the design choices made in the previous sections, we are currently working on a prototype, which should serve as a testbed for experimenting with the DSL, the composition of operators, and the overall workflow for constructing hybrid metaheuristic solvers.

The prototype is being implemented in Julia, chosen for its high performance for scientific computing and its metaprogramming capabilities, which are advantageous for implementing the DSL and the solver components efficiently. The overall implementation is composed of the following key modules:

1. **DSL Parser**: The module responsible for parsing solver descriptions written in the DSL. It transforms the textual representation into an internal Abstract Syntax Tree (AST) that represents the structure and components of the solver pipeline.
2. **Solver Engine**: The core execution engine interprets the AST generated by the parser. It manages the flow of configurations through the specified sequence of operators and connectors and executes them based on the defined pipeline structure. It handles the instantiation of operators, the management of configuration streams, and the control flow.
3. **Modular Component Library**: A library containing implementations of the core operators (e.g., `Init`, `Generate`, `Neighbourhood`, `Accept`) and connectors (`MUX`, `DEMUX`) described in Section 3. It should be easily extensible, to allow new operators or variations of existing ones to be added. Each component follows the defined input/output interfaces to ensure modularity and composability.
4. **Problem Interface**: An abstraction layer designed to decouple the solver engine and components from specific problem model descriptions. It defines structures for representing problem instances (variables, domains, constraints) and configurations.

The interaction between these modules goes as follows: the parser reads the DSL input, the engine uses the resulting AST to configure and run the pipeline, invoking operators from the component library, which in turn operate on problem-specific data structures defined via the problem interface. Listing 1 shows an example of how the DSL (tabu search example in Section 4.2) is translated and parsed into Julia code, resulting in the construction of a solver pipeline using the proposed architecture.

Listing 1 Tabu search solver pipeline in Julia.

```
  Pipeline([
    Loop(
        Pipeline([
            Neighbor(SwapMove()),
            Accept(:tabu, Dict{Symbol,Any}(:tabu_size => tabu_size))
        ]),
        max_iterations,
        0.0,  # Target fitness for early stopping
    )
])
```

While still under active development, we aim to demonstrate the expressiveness of the framework in representing known metaheuristics and the flexibility of the proposed architecture in creating novel hybrid solvers. Initial experiments focus on reconstructing standard algorithms like Simulated Annealing, Tabu Search and basic parallel search strategies to verify the interaction between the core components.

6 Experimental Results for Magic Square

We constructed five distinct solvers: Tabu Search[11], two Simulated Annealing variants, GRASP (Greedy Randomized Adaptive Search Procedure), and a hybrid of Tabu Search and Simulated Annealing. Each solver employed a local search strategy with appropriate neighborhood operators, acceptance criteria, and control flow defined within the DSL.

Table 2 shows the performance of six solver configurations on the 3x3 magic square problem, including a comparison with a standard implementation from the `Metaheuristics.jl` package. The column "Parameters Used" shows how the core parameters for each metaheuristic (such as tabu list size, SA's initial temperature and cooling rate, or GRASP's greediness factor) are declaratively specified within the DSL. The results indicate that both the moderate Simulated Annealing configuration (t0=10) and the Hybrid (TS+SA) solver achieved a 100% success rate. The Hybrid solver was efficient, finding a solution in an average of 301ms and 4514 iterations. The aggressive SA variant (t0=20) also performed well with an 87% success rate, while Tabu Search proved robust, achieving a 74% success rate. GRASP, in its current setup, found solutions remarkably quickly in its successful runs (averaging only 3ms and 71 iterations) but exhibited a lower overall success rate of 27%. For comparison, the `Metaheuristics.jl` SA solver achieved a 62% success rate. It is notably faster, which is expected since our framework is currently developed with a focus on features and architectural flexibility first; performance optimizations will be addressed once the implementation is finalized.

Table 2 Comparison of `MoSCO`-specified Solvers on 3x3 Magic Square (100 runs, max 50k iterations/run).

Solver	Parameters Used	Avg Time (ms, successful)	Avg Iterations (successful)	Success Rate (%)
Tabu Search	`tabu_list_size=7`	383	16779	74.0
Simulated Annealing (Moderate)	`t0=10, cooling_rate=0.95`	397	11757	100.0
Simulated Annealing (Aggressive)	`t0=20, cooling_rate=0.95`	577	17249	87.0
GRASP	`greediness=0.3`	3	71	27.0
Hybrid (TS+SA)	`tabu_list_size=7,t0=20, cooling_rate=0.95`	301	4514	100.0
Metaheuristics.jl (SA)	`Default`	119	N/A	62.0

▶ Note. The `Metaheuristics.jl` package uses an internal, automatic annealing schedule that is linear with respect to the fraction of evaluations completed (T = nevals/max_evals) unlike the classic geometric cooling schedule implemented in our SA variants. This means its temperature and cooling rate are not directly tunable by the user, hence the 'Default' parameter listing.

This comparative overview highlights the framework's potential for easy and rapid experimentation, as well as the analysis of solver designs based on their core components.

Further benchmarking will involve more complex problems, larger instances, and a wider array of solver configurations and performance metrics to rigorously assess the framework's capabilities and the performance characteristics of the generated solvers.

7 Future Directions

We presented a modular architecture for constructing meta-heuristic based solvers. This approach simplifies the process of creating high-performance solvers for different problem domains, by providing a DSL for connecting reusable components into complete architectures. We also presented an analysis framework that can be used to measure the performance of the generated architectures in terms of the quality of solutions and speedups obtained. In the future, we plan to further develop the DSL to include additional elements, such as logical expressions and control flow constructs.

Additionally, we plan to explore the use of meta-learning algorithms to optimize both the parameters and overall architecture of our proposed solver [5] [20] [9] [6], incorporating techniques from the reinforcement learning world to guide the design and configuration of solver components.

We also want to widen the range of problem domains, in order to demonstrate the solvers' adaptability. This will involve testing the architecture on problems with varying levels of complexity and characteristics.

Another line of work which we plan to develop is the design of an XCSP3 [4] or MiniZinc [19] back-end, which would allow us to plug in existing complex model descriptions. A related aspect would be the inclusion of specialized components for global constraints [2].

Overall, the proposed architecture has the potential to improve the performance of existing optimization algorithms and generate new, high-performance solvers for a wide range of problem domains, by simplifying the design process and incorporating meta-learning algorithms for optimization.

References

1 David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. The traveling salesman problem: a computational study. *Princeton university press*, 2006.

2 Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Thierry Petit. Global constraint catalogue: Past, present and future. *Constraints An Int. J.*, 12(1):21–62, 2007. doi:10.1007/S10601-006-9010-8.

3 Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308, 2003. doi:10.1145/937503.937505.

4 Frédéric Boussemart, Christophe Lecoutre, Gilles Audemard, and Cédric Piette. Xcsp3: an integrated format for benchmarking combinatorial constrained problems. *arXiv preprint arXiv:1611.03398*, 2016.

5 Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of metaheuristics*, pages 457–474. Springer, 2003. doi:10.1007/0-306-48056-5_16.

6 Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pages 449–468. Springer, 2010.

7 Sébastien Cahon, Nordine Melab, and El-Ghazali Talbi. Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380, 2004. doi:10.1023/B:HEUR.0000026900.92269.EC.

8 Eranda Çela. *The quadratic assignment problem: theory and algorithms*. Springer Science & Business Media, 1998.

9 Konstantin Chakhlevitch and Peter Cowling. Hyperheuristics: recent developments. In *Adaptive and multilevel metaheuristics*, pages 3–29. Springer, 2008. doi:10.1007/978-3-540-79438-7_1.

10 Juan J Durillo and Antonio J Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011. doi:10.1016/J.ADVENGSOFT.2011.05.014.

11 Fred Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

12 John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

13 Holger H Hoos and Thomas Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004.

14 James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of International Conference on Neural Networks (ICNN'95), Perth, WA, Australia, November 27 - December 1, 1995*, pages 1942–1948. IEEE, 1995. doi:10.1109/ICNN.1995.488968.

15 Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983. doi:10.1126/SCIENCE.220.4598.671.

16 Manuel López-Ibáñez, Danny Múnera, Daniel Díaz, and Salvador Abreu. Weaving of metaheuristics with cooperative parallelism. In *Parallel problem solving from nature—PPSN XV: 15th international conference, coimbra, portugal, september 8–12, 2018, proceedings, part I 15*, pages 436–448. Springer, 2018.

17 Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. *Iterated Local Search*, pages 320–353. Springer US, Boston, MA, 2003. doi:10.1007/0-306-48056-5_11.

18 Danny Munera, Daniel Diaz, and Salvador Abreu. Hybridization as cooperative parallelism for the quadratic assignment problem. In *Hybrid Metaheuristics: 10th International Workshop, HM 2016, Plymouth, UK, June 8-10, 2016, Proceedings 10*, pages 47–61. Springer, 2016. doi:10.1007/978-3-319-39636-1_4.

19 Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007. doi:10.1007/978-3-540-74970-7_38.

20 Ender Özcan, Burak Bilgin, and Emin Erkan Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, 2008. URL: http://content.iospress.com/articles/intelligent-data-analysis/ida00313.

21 Alejandro Reyes-Amaro, Eric Monfroy, and Florian Richoux. Posl: A parallel-oriented metaheuristic-based solver language. *International Journal of Metaheuristics*, 6(1-2):6–29, 2017.

22 El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

23 Paolo Toth and Daniele Vigo. *The vehicle routing problem*. SIAM, 2002.

Semantic Representation of Adverbs in the Lexicalized Meaning Representation (LMR) Framework

Jorge Baptista
University of Algarve, Campus de Gambelas, Faro, Portugal
INESC-ID Lisboa, Portugal

Izabela Müller
University of Algarve, Faro, Portugal
INESC-ID Lisboa, Portugal

Sónia Reis
University of Algarve, Faro, Portugal
INESC-ID Lisboa, Portugal

Abstract

Semantic parsing serves as a crucial interface between natural language and formal meaning representations, enabling computational systems to capture the underlying semantic structure of linguistic expressions. This paper addresses a relatively understudied area in both linguistic theory and natural language processing: the semantic representation of *adverbs*. We conduct a comparative analysis of annotation guidelines and practices across two semantic representation frameworks: Lexicalized Meaning Representation (LMR), applied to the European Portuguese edition of the novella "*O Principezinho*" by Antoine de Saint-Exupéry (1943); and Abstract Meaning Representation (AMR), applied to the Brazilian Portuguese edition, "*O Pequeno Príncipe*". The study reveals significant limitations in AMR's handling of adverbial constructions, particularly when assessed against contemporary syntactic-semantic advances in linguistic theory. Furthermore, it highlights the theoretical and practical challenges that LMR continues to face in this domain.

2012 ACM Subject Classification Computing methodologies → Semantic networks

Keywords and phrases Semantic representation, Adverbs, Lexicalized Meaning Representation (LMR), Abstract Meaning Representation (AMR), Annotation guidelines, European Portuguese, Brazilian Portuguese, Comparative analysis, The Little Prince, Corpus linguistics, Natural Language Processing (NLP), Multi-word expressions, Syntactic-semantic interface, Linguistic theory

Digital Object Identifier 10.4230/OASIcs.SLATE.2025.9

Funding This work was partially supported by Portuguese national funds through Fundação para a Ciência e a Tecnologia, DOI: 10.54499/UIDB/50021/2020).
Izabela Müller: Universidade do Algarve, Faculdade de Ciências Humanas e Sociais, Programa de Doutoramento em Ciências da Linguagem

1 Introduction

Within symbolic approaches to language, semantic parsing acts as a crucial interface between natural language (NLPT) and formal meaning representations, enabling computational systems to capture the underlying semantic structures of linguistic expressions. Among the various linguistic categories, adverbs remain relatively underrepresented in both linguistic theory and natural language processing. This study offers a comparative analysis of annotation guidelines and practices across two semantic representation frameworks as applied to a comparable text corpus in both European (PT-PT) and Brazilian Portuguese (PT-BR).

On the one hand, Abstract Meaning Representation (AMR), a popular framework for semantic parsing [9, 10], was applied to the Brazilian Portuguese edition of Antoine de Saint-Exupéry's novella *O Pequeno Príncipe* [4]. On the other hand, Lexicalized Meaning

Representation (LMR) [16, 15], an AMR-inspired semantic annotation framework, specifically designed to address challenges particular to European Portuguese and to overcome certain limitations observed in the practical use of AMR for semantic parsing, has been applied to the European Portuguese edition, *O Principezinho* (1943).

The analysis reveals significant limitations in AMR's treatment of adverbial constructions, especially when evaluated against recent developments in syntactic-semantic theory. At the same time, it underscores the ongoing theoretical and practical challenges faced by LMR in this domain.

Given that adverbs and adverbial constructions in general constitute a substantial portion of content words in texts – and fulfill various functions *within* sentences (as determiners, modifiers, and quantifiers) as well as *across* sentences of a discourse (as conjunctive, evaluative, or enunciative elements) – their accurate semantic representation is essential for understanding both the informational content of texts and the organizational structure of meaning in discourse. This paper aims to bring these issues to the forefront and to encourage a discussion that goes beyond mere formalism differences between distinct frameworks, advocating for more robust and descriptively adequate solutions in the semantic representation of adverbs.

The paper is structured as follows: Section 2 provides a succinct literature review on AMR and LMR, including existing lexical resources that support semantic annotation, main applications, and tools associated with semantically annotated corpora, with a focus on current efforts in Portuguese. Section 3 presents, in some detail, the two semantically annotated versions of Saint-Exupéry's novella *The Little Prince / Le Petit Prince*, along with the NLP processing applied to both versions to facilitate information retrieval from the corpora. Sections 4 and 5 summarize the key features of each annotation framework particularly with regard to the semantic parsing of adverbs. Next, Section 6 discusses the main solutions adopted and the challenges encountered in the AMR representation of *O Pequeno Príncipe* [4], and compares these with the strategies proposed by LMR [16] for identical or similar cases. The paper concludes with a summary of the findings and suggestions for future research.

2 Literature review

Since its introduction by Banarescu *et al.* [9], Abstract Meaning Representation (AMR) has emerged as a widely adopted graph-based framework for semantic representation. In AMR, the meaning of a sentence is encoded as a directed, acyclic, single-rooted graph, where nodes correspond to concepts and edges denote semantic relations. Beyond the foundational proposal, supplementary resources – most notably the tutorial by Schneider *et al.* [52] – have played a pivotal role in facilitating the adoption of the formalism. These materials have been essential for training annotators and for standardizing the use of PENMAN notation in the representation of AMR graphs.

The initial motivation behind the AMR formalism was to address the scarcity of corpora with suitable semantic annotations for core natural language processing tasks such as named entity recognition, semantic role labeling, word sense disambiguation, and co-reference resolution [9]. The framework also sought to streamline the annotation process by abstracting away from morphosyntactic details and word order, while simultaneously drawing on existing linguistic resources [43].

Abstract Meaning Representation (AMR) has emerged as a widely adopted semantic representation framework across a broad range of Natural Language Processing (NLP) tasks. Its increasing relevance is supported by recent advances in semantic graph-based text analysis and generation. The following paragraphs highlight key application areas documented in the literature.

AMR has been incorporated into multi-step pipelines involving syntactic parsing, semantic content extraction, and summary generation. [27] propose a three-stage approach: converting narrative text into AMR graphs, extracting summary subgraphs, and generating corresponding sentences. Similarly, AMR2Text [57, 60] addresses the challenge of generating fluent natural language from semantic graphs.

AMR-enhanced Transformer architectures [38] integrate AMR representations into both the encoder and decoder, demonstrating performance improvements in high- and low-resource settings. These models also help mitigate issues such as unwanted repetition and hallucination.

Techniques such as AMR-Data Augmentation [55] generate semantically equivalent paraphrases by transforming AMR graphs and re-generating text, thus enhancing models for textual similarity and classification. The ABEX model[1] [30] further exploits AMR-based abstract descriptions for few-shot semantic understanding tasks. AMR representations, often combined with textual embeddings, have been applied to diverse classification tasks, including sentiment analysis (APARN[2]; [41]), paraphrase detection, AI-generated text detection (DART; [48]), fake news identification (FakEDAMR; [31]), and toxic content detection (AMR-CNN; [28]).

AMR also provides a structured framework for identifying events, relations, and arguments. [39] and [62] demonstrate its effectiveness in event detection and classification. In the biomedical domain, AMR-based approaches have also proven useful for extracting specialized events and relations [8, 51].

AMR has been applied to both Question Answering and Knowledge(-based) Graph Construction tasks, enabling SPARQL query generation[3], decomposition of complex questions into sub-questions (QDAMR [26]), and semantic graph-based answer validation.

Due to its abstract and modular nature, AMR can be customized for specialized contexts such as legal reasoning [53], human–robot interaction (e.g., Dialogue-AMR [19] and Gesture-AMR [37]), and the representation of technical instructions.

AMR has been employed in tasks such as scene graph analysis and image captioning (SGRAM[23]), as well as in speech modeling for interrupted discourse, with implications for the development of accessible voice assistants [1].

AMR-based semantic metrics have been proposed for evaluating coherence in dialogue systems (DEAM; [38]) and factual consistency in summarization (AMRFact; [50]), showing improved correlation with human judgments.

These applications underscore the versatility of AMR as an interpretable and robust semantic representation framework, suitable for a wide range of NLP tasks, including low-resource and multimodal scenarios.

With an initial focus on the English language, AMR-annotated corpora were developed across several domains, including news articles, online forums, and blogs, although only a limited number of these resources are publicly available. Among the accessible corpora are the AMR version of *The Little Prince* and the Bio AMR corpus [6]. The availability of such resources has spurred the development of a variety of AMR parsers, encompassing graph-based approaches [29], dependency-based methods [58, 59], transition systems [25], and deep learning architectures, including sequence-to-sequence models with BiLSTM [40, 49] and transformer-based models [17, 21].

[1] `https://github.com/Sreyan88/ABEX`
[2] `https://github.com/THU-BPM/APARN`
[3] SPARQL is the standard query language and protocol for querying Resource Description Framework (RDF) data, as defined by the World Wide Web Consortium (W3C).

Recognizing the scarcity of AMR resources in languages other than English, several studies have explored adaptation strategies. [18] and [24] applied transfer learning techniques to train multilingual AMR parsers for languages such as Italian, Spanish, German, and Chinese [61, 63].

For Portuguese, parallel developments have taken place for both Brazilian Portuguese (PT-BR) and, later, European Portuguese (PT-PT). In the PT-BR context, the first AMR corpus was created by [2] based on *O Pequeno Príncipe*, the Brazilian translation of the Saint-Exupéry novel, based on the French edition (1943). This initiative was followed by the construction of corpora from the journalistic domain and from the opinion domain (OpiSums-PT-AMR) [20]. In parallel, various parsing approaches were investigated, including a rule-based system [5], adaptations of English parsers – such as NeuralAMR [36], augmented with Portuguese embeddings – and interlingual strategies like XPTA [54], which achieved a Smatch score [22] of 66%.

For European Portuguese (PT-PT), [16] proposed the Lexicalized Meaning Representation (LMR), an adaptation of the AMR framework designed to account for grammatical specificities of PT-PT and to address several issues in the original AMR annotation guidelines. LMR introduces both structural and methodological modifications aimed at preserving the grammatical category and lexical identity of tokens, avoiding abstraction through verb lemmas, and maintaining a direct (one-to-one) link between content words of the surface text and their semantic representation. Unlike AMR, which typically omits auxiliaries, copulas, and prepositions, LMR explicitly represents these elements, encoding corresponding semantic relations such as `:VAUX` and `:VSUP`, in the case of auxiliary verbs, or the semantic nexus conveyed by prepositions, for example `:place`. Annotation in LMR is guided by a catalogue of lexical senses, drawing on resources such as the *Dicionário Gramatical de Verbos do Português* [13] and lexicon-grammar of predicative nouns [14], which provide extensive coverage of verbal and nominal predicates, respectively. Furthermore, the semantic predicates of both these resources are mapped onto their respective adjectivalizations.

In a second study, [15], 50 sentences from the *O Principezinho* were analysed across four language versions (EN, ES, PT-BR, and PT-PT), comparing LMR with traditional AMR. The analysis highlighted inconsistencies in existing annotations and showed that LMR enables greater alignment between syntactic structure and semantic representation, particularly in the treatment of null subjects, sentential adverbials, and auxiliary verb constructions. The proposal is accompanied by a set of public annotation guidelines [11] and aims at greater linguistic adequacy and computational formalization.

Parser evaluation is traditionally based on the SMATCH metric [22], which measures the overlap between generated and gold-standard graphs. However, alternative metrics such as SEMA [7] have emerged, which are more sensitive to structural properties.

Concerning future work on AMR-related issues, [7] propose expanding Brazilian Portuguese AMR corpora, employing techniques such as ensemble learning and back-translation, and explicitly handling linguistic phenomena such as the so-called *null subjects*, multiword expressions (MWE; e.g *pessoas crescidas* "grown-ups"), and synthetic diminutive suffixes (e.g. the suffix -(*z*)*inho* in *Principezinho* "The little Prince").

According to the LMR guidelines [11], certain linguistic phenomena are addressed with specific strategies to streamline the annotation process. Notably, the resolution of zero pronouns – or more broadly, any omitted arguments within a sentence – is deferred to a post-processing phase, unless the antecedent is present within the same sentence. This approach aims to reduce the cognitive load on annotators and enhance consistency across annotations. Conversely, multiword expressions (MWEs) are identified during a pre-processing stage, as

they function as semantic units and should be treated as indivisible entities prior to annotation. This methodology ensures that annotators can focus on delineating semantic relations within sentences. Similarly, the identification, delimitation, and potential normalization of named entities, including temporal expressions [33], are managed through automated processes, thereby relieving annotators of these tasks [16]. Regarding diminutive suffixes – a complex aspect in Portuguese due to their nuanced semantic implications, and, apparently, without any English or French equivalent – the LMR framework currently does not yet prescribe a specific annotation strategy, acknowledging the need for further research in this area.

To sum up, since its introduction [9], Abstract Meaning Representation (AMR) has evolved [10], from a theoretical framework with initial annotations in English to the development of specialized parsers [25, 29] and, more recently, to multisentential [47] and multilingual adaptations [61]. In the context of Portuguese, both Brazilian Portuguese and European Portuguese have witnessed significant advancements. For PT-BR, initiatives include the creation of annotated corpora such as AMRNews and OpiSums-PT-AMR, which address domain-specific challenges in journalistic and opinion texts, respectively [35]. Additionally, strategies like back-translation have been explored to mitigate the scarcity of annotated data for natural language generation tasks [56]. In the case of PT-PT, adaptations have focused on aligning AMR with the linguistic characteristics of European Portuguese, leading to the development of resources like the Lexicalized Meaning Representation (LMR) framework [16]. These collective efforts underscore the commitment to tailoring AMR or AMR-inspired frameworks to the specific features of Portuguese.

3 Methods

This study is based on two parallel corpora derived from different Portuguese translations of *The Little Prince/Le Petit Prince*, de Antoine de Saint-Exupéry (1943): the Brazilian Portuguese version, *O Pequeno Príncipe*, annotated according to the Abstract Meaning Representation (AMR) [9] framework, and the European Portuguese version *O Principezinho*, annotated within the Lexicalized Meaning Representation (LMR) framework [16]. Our analysis pays particular attention to the representation of adverbs, both single-word (e.g. *muito* "very", *atentamente* "attentively") and multiword expressions (e.g. *de vez em quando* "from time to time", *pouco a pouco* "little by little").

The Brazilian AMR corpus follows the AMR annotation guidelines [9] and was developed by aligning it with the English version of *The Little Prince*, and the corresponding AMR annotations [2, p. 975]. Following the methodology adopted in similar AMR projects in other languages, the English AMR structures were imported and then adapted to accommodate the Brazilian Portuguese version. The novel, organized into twenty-seven chapters, contains 1,562 sentences in English and 1,527 in Brazilian Portuguese.

To support the analysis of both the AMR corpus, we processed the Brazilian Portuguese version of the novel through the STRING system [42]. This Natural Language Processing (NLP) system, specifically developed integrates a comprehensive lexicon of Portuguese adverbs, both single-word (e.g. *apenas* "only", *lentamente* "slowly") and multiword expressions (*no entanto* "however", *de preferência* "preferably"), including the Brazilian Portuguese multiword adverb lexicon, currently in development [46]. The corpus was automatically tagged for parts of speech and syntactic dependencies, providing detailed structural information to cross-reference with AMR annotations. The STRING system's output was then used to help retrieve (most of) the instances of adverbs and adverbial structures in the Brazilian text that eventually had been represented in the AMR annotation. In our analysis of *O*

Pequeno Príncipe (PT-BR), we identified 90 distinct single-word adverbs, which appeared 1,023 times throughout the text. Among these, 42 unique adverbs ending in *-mente* were observed, occurring 76 times. Additionally, we found 159 different compound adverbs, with a combined total of 241 occurrences. It is important to note that temporal named entities, despite functioning adverbially, were excluded from this count.

In the analysis of *O Principezinho* (PT-PT), we identified 1,107 adverbs in 1,503 sentences. Of these, 739 were single-word adverbs, corresponding to 51 distinct forms; 297 were compound adverbs, with 174 unique forms; and 71 were *-mente* adverbs, of which 42 were different.

These data establish a solid empirical foundation for the contrastive analysis that follows, enabling a systematic evaluation of how the two semantic annotation frameworks represent adverbs in the context of equivalent translations. The following sections present the core principles and strategies of each of the frameworks under analysis.

4 AMR Guidelines

This section focuses specifically on the relevant components of the AMR guidelines that govern the annotation of adverbial content words. These can be summarized as follows.

Adverbs ending in *-mente* suffix (ing., *-ly*) are for the most part **manner** adverbs. The AMR guidelines stipulate[4] that, for representation purposes, adverbs ending in *-ly* are typically stemmed from their base adjectival form and are introduced by the dependency relation `:manner`.

For example, the adverb *quickly* would be represented as *quick*, so that the lemma form can be easily mapped across related constructions; for example:

(1) *The army moved quickly* vs. *The army moved in a quick way/manner*
→ `(m / move :arg0 (a / army) :manner (q / quick)).`

Naturally, in the case an adverb ending in *-ly* conveys a different sense other than **manner**, it is annotated as it is in the text and it is not stemmed. This is the case of some quantifying adverbs. They involve intensifiers (*very*, *extremely*) and downtoners (*somewhat*, *relatively*), which are annotated with `:degree`[5]. For example, in (2) the quantitative sense of the adverb implies a different representation and the word is not stemmed:

(2) *I hardly know her.*
→ `(k / know-01 :ARG0 (i / i) :ARG1 (s / she) :degree (h / hardly))`

The *frequency* role is used to annotate "how often an action or event occurs"[6]. Analytical frequency temporal expressions, which are usually parsed as temporal named entities, and involve the noun *times* are simply parsed as in:

(3) *We met three times.*
→ `(m / meet-03 :frequency 3 :ARG0 (w / we))`

while, for those that involve several variables, the AMR resources to the special frame `rate-entity-91`, an abstract construct rendering the representation much more complex. For example:

[4] https://github.com/amrisi/amr-guidelines/blob/master/amr.md#adverbs-with--ly
[5] https://github.com/amrisi/amr-guidelines/blob/master/amr.md#degree
[6] https://github.com/amrisi/amr-guidelines/blob/master/amr.md#frequency

(4) *We play bridge every Wednesday afternoon.*
→ `(p / play-01 :frequency (r / rate-entity-91 :ARG4 (d / date-entity :weekday (w2 / wednesday) :dayperiod (a / afternoon))))`.

Still, one can arguably consider that this category also covers simple adverbs such as *often*, or *sometimes*.

In a similar way, **duration**[7] adverbs are annotated with the `:duration` relation, plus the abstract construct `:temporal-quantity`, which takes two arguments, a `:quant` and the `:unit`:

(5) *He worked for two hours.*
→ `(w / work-01 :duration (t / temporal-quantity :quant 2 :unit (h2 / hour)))`

As a final note on the AMR guidelines, note that in this framework, the PENMAN graph has no explicit `root` node; instead, the topmost semantic predicate serves this role. As a consequence, adverbial sentence-external modifiers [44] are inadequately represented when they are forced to depend on the main verb of the sentence, as though they were merely internal verbal modifiers, thereby obscuring their true scope and semantic independence. This is particularly obvious with conjunctive adverbs (*therefore*, *nevertheless*, *however*) evaluative adverbs (*unfortunately*, *hopefully*), and enunciation-oriented adverbs (*sincerely*, *honestly*).

5 LMR Guidelines

In this section, we present the contributions of LMR [16] to the annotation of the various aspects of AMR discussed previously. Our focus is on highlighting a number of fundamental differences between the two frameworks, rather than providing an exhaustive comparison.

The main distinction between LMR and AMR lies in the fact that LMR annotates the text *directly*, constructing the sentence graph on the actual textual elements. In contrast, AMR generates a separate graph that is then *appended* to the sentence as a whole. Secondly, LMR operates directly on the textual elements, without lemmatizing them beforehand; lemmatization, as well as word-sense disambiguation, are considered postprocessing steps.

LMR also avoids introducing abstract constructs to represent meaning. Any abstraction or semantic generalization, such as translating the actual meaning of words into more abstract representations – if should it prove to be necessary, is deferred to a later stage. This choice reflects the understanding that abstract constructs are often difficult for human annotators to manipulate consistently.

Furthermore, LMR does not replace meaningful lexical items, such as conjunctions (e.g., *porque* "because"), with abstract semantic relations like `:cause`. Instead, it retains the original textual elements and repurposes the dependency relation `:op2` to link these so-called grammatical words to their respective arguments.

Finally, the reconstitution of zeroed anaphoric elements is handled in LMR similarly to AMR, provided that the antecedent is present within the same sentence. Anaphora resolution is considered a postprocessing task, so as not to overburden the human annotation process. For this reason, omitted arguments – such as the subject, which is typically zeroed in null-subject languages like Portuguese – are not reconstituted during the annotation phase.

[7] `https://github.com/amrisi/amrguidelines/blob/master/amr.md#duration`

However, LMR does allow for the reconstitution of lexical elements that have been omitted due to contextual redundancy. This corresponds to what [34] referred to as *appropriate reduction*, in which an omitted element can be safely restored because its zeroing results from being the most likely (often the sole) candidate in the given context, and is therefore considered redundant. For example, in:

(6) *O Pedro gosta de* (*ler*) *romances policiais* "Pedro likes (to read) detective novels"
→ `(g / gosta :arg0 (r / Rui) :arg1 (l / ler :arg0 r :arg1 (rp / romances policiais)))`

one can safely reconstitute *ler* (and its repeated zeroed subject, correferent to the main subject) as this representation more adequately describes the meaning of the sentence. By reintroducing such elements, the overt semantic structure of the sentence is fully recovered.

We now turn to the specific features of LMR concerning the representation of adverbs and adverbial constructions. With regard to adverbs ending in *-mente* (and their English equivalents with the *-ly* suffix), LMR adopts an approach distinct from that of standard AMR. Instead of replacing the adverb with its morphologically related adjective (e.g., *quickly* → *quick*), LMR retains the adverb in its textual form as a node in the graph, applying the `:manner` relation directly to the adverb and linking it to the predicate it modifies. Similarly, in the case of analytical phrases (e.g., *in a quick way* / *in a quick manner*), LMR preserves the manner operator-noun (*way*, *manner*) and represents the surface adjective in its original form, while maintaining the relevant syntactic and semantic relations. Hence, for the examples in (1), renumbered below as (7), LMR produces two distinct but semantically equivalent representations: one for the synthetic form of the adverb with the suffix, and another for the analytical construction with the manner operator-nouns:

(7) *The army moved quickly* vs. *The army moved in a quick way/manner*
→ `(m / move :arg0 (a / army) :manner (q / quickly))`
vs. → `(m1 / move :arg0 (a / army) :manner (i / in :op2 (m2 / manner :mod (q / quick)))`.

In addition, *subject-oriented manner adverbs* ([44], class MS), such as *cuidadosamente* "carefully", which are also annotated with `:manner`, an added `:arg0` dependency that links the adverb to the subject, reflecting its dual scope (8):

(8) *O Pedro trabalhava* cuidadosamente cp. *O Pedro era* cuidadoso
"Pedro worked carefully cp. Pedro was careful"
→ `(root :main (t / trabalhava :arg0 (p / Pedro) :manner (c / cuidadosamente :arg0 p)))`

The `:manner` relation is also used in prepositional expressions, such as *com sabedoria* lit.: "with wisdom" "wisely", where the preposition *com* "with" is linked to the upper predicate via `:manner` and to the predicative noun by `:op2`:

(9) <id=222>*Mas, logo a seguir, observou* com sabedoria ... "But, shortly afterwards, he observed wisely."
→ `(o / observou :manner (c / com :op2 (s / sabedoria)))`

The `:degree` relation, used to link adverbial expressions of intensity or degree to the predicative elements they modify, is used in the same way as in AMR:

(10) <id=43> ... *fui despertado por uma vozinha* muito *curiosa* "I was awakened by a very curious little voice."
→ `(v / vozinha :arg0-of (c / curiosa :degree (m2 / muito)))`

This dependency is also to be related to comparative constructions such as *mais* "more" or *menos difícil* "less difficult". This involving a (compound) comparative subordinate conjunction, *do que* "than". In this case, the conjunction is linked by `:compared-to` to the first term of comparison, and the `:op2` relation is linked to the comparative term:

(11) <id=578> *É muito mais difícil julgar-se a si mesmo* do que *julgar os outros.*
"It is much more difficult to judge oneself than to judge others."

```
→ (root :main (d / difícil :vaux (e / é)
:degree (m / mais :degree (mt / muito))
:arg0 (j1 / julgar :arg1 (si / si :mod (msm / mesmo)))
:compared-to (dq / do_que :op2 (j2 / julgar :arg1 (o /
outros)))))
```

Frequency, **duration**, and other **time**-denoting adverbs are represented in LMR in much the same way as in AMR, using the operators `:frequency`, `:duration`, and `:time`, respectively. However, LMR does not distinguish between simple adverbs (e.g., *anteontem* "the day before yesterday") and their multiword equivalents (e.g., *antes de ontem* "idem"). It also does not treat differently any temporal expressions that are typically parsed as *named entities* (e.g., *na passada sexta-feira* "last Friday") [33, 32]. The identification, delimitation, and normalization of these named entities' expressions are expected to take place during the parsing stage, thereby reducing the cognitive load on the human annotator. This allows the annotator to focus primarily on distinguishing among the three major temporal categories.

The `:focus` relation is used in LMR to represent **focus** adverbs whose function is to highlight a constituent or a word within the sentence, following the definiton of [44] (class MF). These include simple forms such as *só* and *apenas* "only, just", *justamente* "just/precisely", and compound expressions such as *em especial* "especially" or *acima de tudo* "above all".

(12) <id=170> *Só então acham que o conhecem.* "Only then do they think they know him."
→ `(a / acham :time (e / então :focus (s / só))`

The specific syntactic-semantic properties of **focus** adverbs are detailed below, in Section 6, but suffice to say, for now, that they show strong resistance to clefting when this operation separates it from the element or constituent they focus on, but they can undergo clefting along with that element or constituent:

(13) a. **É* só *que então acham que o conhecem.*
b. **É* então *que só acham que o conhecem.*
c. **É* só então *que acham que o conhecem.*

In addition to adverbs like *só* "only", the cleft construction *ser... que* "is...that" is also treated as a focusing device. Focus adverbs are thus distinct from other adverbial types. Nevertheless, they exhibit diverse and complex syntactic behavior, as well as different specific meanings, and their scope, whether over the entire sentence or a specific constituent can sometimes be difficult to determine [45].

In the case of **polarity**-denoting adverbs, such as *não* "not", *nunca* "never", and *jamais* "never", these are explicitly encoded in the graph and linked to the element they modify using the `:neg` relation. The negative value conveyed by the subordinating conjunction *sem* "without" is not marked, however, as it is assumed to be incorporated into its lexical entry.

LMR also introduces certain semantic relations not present in AMR. For instance, **conformative** or **proportional** adverbials – such as those introduced by *conforme* "as, according to" or *à medida que* "as, while" – are represented using the relation `:conformative`.

Finally, leveraging the fact that all LMR graphs have a `root` node, sentence-modifying adverbs can be linked to this node to signal their scope on the whole sentence. This is the case of *então* "then", which is a connective discursive device, that is, a conjunctive adverb [44] (class PC) and is to be interpreted as an adverbial modifier on the entire sentence (either as a **time** or a **consequence**), as shown in (14):

(14) <id=15> *Então, desenhei* ... "Then, [I] draw ..."
→ `(root :mod (e / então) :main (d / desenhei ...`

Notice, however, that the AMR guidelines make no reference to sentence-external adverbial modifiers. Most of the time, as we will see below, these elements are either made to depend on the main predicate – which is arguably inadequate – or are omitted altogether from the graph, a rather unsatisfactory solution.

6 Corpora analysis

In this section, we present key findings from our analysis of the AMR corpus of *O Pequeno Príncipe* in Brazilian Portuguese (PT-BR) [3]. Where appropriate, we contrast the annotation solutions observed in this corpus with those found in the LMR annotation of *O Principezinho* in European Portuguese (PT-PT) [12] taking into account the differences between the two language varieties.

In many cases, adverbs ending in *-mente* "-ly", which were supposedly to be represented by the *lemma* of their adjectival base (e.g., *quickly* represented as *quick*, see (1)), appear to have been annotated inconsistently, and were retained in their derived form, i.e. with the suffix *-mente*. Of 76 instances of adverbs ending in *-mente*, 60 have not been stemmed. For example, in (15), we found the derived forms of two adverbs (*lentamente* "slowly", *regularmente* "regularly"), ending in -mente and not the corresponding base forms (*lento* "slow" and *regular* "regular"):

(15) <id=440> ... *os vulcões queimam* lentamente, regularmente*, sem erupções.*
"the volcanoes burn slowly, regularly, without eruptions."
→ `c / queimar :manner (v / lentamente) :manner (v / regularmente)`
`:manner ((e / erupção :polarity -))`

The AMR representation of the corresponding English sentence, although structurally very similar, differs significantly:

(16) <id=440> *If they are well cleaned out , volcanoes burn slowly and steadily , without any eruptions .*
→ `(b / burn-01 :ARG1 (v / volcano :ARG1-of (e2 / erupt-01`
`:polarity -)) :manner (s3 / steady) :ARG1-of (s / slow-05))`

Apparently, the prepositional phrase *without any eruptions* was parsed as a modifier (`:arg1-of`) of the noun *volcanoes*, while the stemmed adverb *steadily* is correctly parsed as a `:manner` of the main verb *burn*. However, despite being coordinated with the latter, the adverb *slowly* – also stemmed to *slow* – was parsed as an `:arg1-of` of the verb, which is likely an annotation error.

The LMR solution to the corresponding sentence in PT-PT addresses all these issues, even if a manner operator-noun *forma* "way/manner" is involved:

(17) <id=440> *Quando são bem limpos, os vulcões ardem de forma lenta e regular, sem erupções.* "When they are well cleaned, the volcanoes burn slowly and regularly, without eruptions."
→
```
(root :main (a / ardem :arg0 (v / vulcões)
:manner (d / de :op2 (f / forma :mod (e / e
:coord1 (l / lenta) :coord2 (r / regular))))
:manner (s / sem :op2 (e / erupções))
```

Still, the (assumed) asyndetic coordination between the **manner** adverbial formed with the operator-noun *forma* and the adverbial introduced by the preposition *sem* is not represented in the graph.

On the other hand, and according to AMR guidelines, *subject-oriented manner* adverbs are parsed in the same way as other, more generic manner adverbs – typically verb- or object-oriented – by using the `:manner` relation.

(18) *Olhou atentamente, e disse: – Não!* "He looked attentively and said, 'No!' "
→ `(o / olhar-01 :ARG0 (e1 / ele) :manner (a / atentamente))`

While this solution is consistent with the AMR guidelines, it fails to capture the dual scope of such adverbs, which modify both the verb and the subject. In this respect, LMR proposes to represent this secondary scope explicitly, by introducing an additional `:arg0` relation linking the adverb to the subject of the sentence.

(19) *Ele analisou com toda a atenção e disse: – Não!* "He examined it very carefully and said: "No!' "
→
```
(root :main (e / e :coord1 (a / analisou :arg0 (e / ele)
:manner (c / com :op2 (a / atenção :arg0 e :quant (t / toda))))
:coord2 (d / disse :arg0 e :arg1 (n / não :mode-exclamative))))
```

In the English AMR-annotated text, 21 instances of the `:duration` relation were identified, most of which make use of the `temporal-quantity` construct, as previously described. However, several examples – some arguably partially idiomatic – are annotated using only `:duration`, without further specification. These include expressions such as *for a time* (id=1559), *for a moment* (id=598), and *for an instant* (id=1522):

(20) <id=1522> *He remained motionless for an instant.*
`(r / remain-01 :duration (i / instant)`

Simple-word **duration** adverbs have been inconsistently represented in *The Little Prince* AMR corpus. Although relatively few in number, these cases provide insightful examples that help highlight inconsistencies in annotation practices. For example, the adverb *forever* appears only three times and is annotated differently in each instance: as `:duration` (id=1040), as `:mod` (id=20), and, in one case, as `:extent` (id=1179) – as in the sentence *You become*

responsible, forever*, for what you have tamed.* This latter annotation corresponds precisely to the recommendation provided in the official AMR guidelines[8], where the example given is *The road goes on* forever. On the other hand, the adverb *never* is systematically represented as *ever*, introduced by the `:time` relation and accompanied by the `polarity -` qualifier on the main verb (e.g., id=20). This is observed in 33 instances). This uniform treatment of *never* as a **temporal** locative rather than a **duration** adverb may be debatable. However, the distinction between these categories is not always clear-cut. In certain cases, the adverb is entirely omitted from the representation (e.g., id=302, 438, 748, 1144). Other uses of `:duration` seem more "creative" and are sometimes difficult to interpret (see, for instance, id=797, 905, or 1119).

We have seen in Section4 the proposed representation for **frequency** adverbials. In practice, AMR annotation of *The Little Prince* for frequency adverbs varies, especially for simple-word adverbs. For *sometimes* we find the adverb directly attached by `:frequency` to the main predicate:

(21) <id=509> *" Then I - - I order you* sometimes *to yawn and sometimes to - - "*
→ `(y2 / yawn-01 :frequency (s / sometimes))`

Other times, the (generic?) `:time` relation is used:

(22) <id=1459> *And you will sometimes open your window ...*
→ `(o / open-01 :time (s / sometimes)`

As for the adverb *often*, besides the `:time` relation, one also finds the abstract construct `have-degree-91` takes the adverb as an argument `:ARG2` and an inverted relation `:frequency-of` operating on the main verb:

(23) <id=1069> *" It is an act too often neglected , " said the fox .*
→ `(n / neglect-01 :ARG1 (a / act-02) ARG1-of (h / have-degree-91 :ARG2 (o / often :frequency-of n)))`

The number of **frequency** adverbials explicitly marked as such in the AMR representation of the PT-BR corpus is smaller (16 instances) than in the English text. Still, the Brazilian annotators did not strictly follow the AMR guidelines. In most cases, instead of using the *temporal-quantity* construct – which appears only three times:

(24) <id=972> ... *ele dá uma volta por minuto ...* "it makes one turn per minute"
→ `(d / dar-03 :ARG0 (p / planeta) :ARG1 (v / volta) :frequency (t2 / temporal-quantity :quant 1 :unit (m1 / minuto)))`

they opted for alternative representations, namely, the annotators just used the `:frequency` relation and omitted the time-denoting noun *vez(es)* "time(s)":

(25) <id=673> ... *só fui incomodado três vezes.* "[I] was only disturbed three times."
→ `(i / incomodar-01 :ARG0 (e / eu) :frequency 3)`

or just added the `:unit` for the *modulo* element:

(26) <id=939> ... *trabalhavam duas vezes por ano.* "[they] worked twice a year"
→ `(t / trabalhar :frequency 2 :unit (a / ano))`

[8] https://github.com/amrisi/amr-guidelines/blob/master/amr.md#extent

For *algumas vezes* "a few times" (id=1168) the determiner is used directly under `:frequency`, like the numbers in (25).

The most salient aspect is the representation of several multiword frequency adverbs in much the same way as proposed by LMR, with the graph nodes formed by hyphenating the individual elements of the multiword expression. Thus, we find compound adverbs such as: *de vez em quando* "once in a while" (id=587), represented as `de-vez-em-quando`; *cada vez* "every time" (id=589), as `cada-vez`; *às vezes* "sometimes" (id=683), as `às-vezes`; *todos os dias* "every day" (id=747), as `todo-dia`; *toda semana* "every week" (id=748), as `toda-semana`. Finally, the multiword expression *de ano em ano* "from year to year" (id=790) is represented simply by *ano*, which is likely an annotation oversight.

The concept of **focus** exists in the AMR directives but deals with a completely different phenomenon and refers to what the sentence is primarily about.[9] In this paper, by the concept of **focus** as applied to determinative adverbs, we consider the use of adverbs as defined in [44] (class MF), namely, an element that is a focus-modifier on a constituent or on another phrase element, including the main verb. We can see examples of these focusing adverbs in the case of *only* in (27) focusing on a quantifier (*one*), *exactly* in (28) focusing on the prepositional phrase *under the star*, or *just* in (29), focusing on the main verb:

(27) <id=370> *They had only one ring of petals ...*

→ `(r / ring :quant 1 :mod (o / only)`

(28) <id=1559> *Wait for a time ,* exactly *under the star.*

→ `(w / wait-01 :mode imperative :location (u / under :op1 (s / star) :manner (e / exact)))`

(29) <id=1395> *I was* just *coming to tell him ...*

→ `(c / come-01 :mod (j / just)`

In nominal and prepositional phrases, focus adverbs resist to be isolated from the constituent they focus on by the clefting operation, for example, (30)-(31) and (33)-(34); but can undergo that operation together with that constituent, as in (32) and (35):

(30) **It was only that they had one ring*

(31) ?/?**It was one ring that they had only*

(32) *It was only one ring that they had*

(33) *He waited exactly under the star*

(34) **It was exactly that he waited under the star*

(35) ?/?**It was under the star that he waited exactly*

While a specific representation of this phenomenon is apparently absent from the AMR directives, in most cases, the focus adverb is captured by a generic `:mod` relation. Notice, however, that *exactly* in (28) is parsed as a `:manner`.

To conclude this section, we offer a brief commentary on *sentence-external adverbial modifiers*, as defined by [44]. These adverbs modify the sentence as a whole and, thus, can be fronted to the beginning of the sentence, even in negative constructions, as they fall

[9] `https://github.com/amrisi/amr-guidelines/blob/master/amr.md#focus`

outside the scope of the negation adverb – an internal sentence modifier (36). Moreover, they do not undergo clefting, since this syntactic operation applies only to sentence-internal constituents (37):

(36) *Portanto, ele (não) fez perguntas.* "So, he (did_not) made some questions."

(37) **Foi portanto que ele (não) fez perguntas.* "It was so that he (did_not) made questions."

Most of these sentence-external adverbs are conjunctive adverbs, whose function is to connect the sentence in which they appear to the preceding sentence (or to another sentence within the same discourse). Therefore, such adverbs cannot occur at the absolute beginning of a discourse.

Since in LMR there is a `root` node for every sentence, it is possible to link sentence-external adverbial modifiers to this *root* node. The lexically-defined, specific semantic nexus conveyed by the adverb can then be used, in a postprocessing stage, to semantically connect different utterances within the same discourse.

For the most part, sentence-external adverbs are omitted from the PT-BR corpus representations. Consider, for instance, *portanto* "so/then/thus/therefore". It appear 5 times in the corpus, and is never represented in the graphs. The multiword *no entanto* "however" appears nine times in *O Pequeno Príncipe* (id=714, 759, 760, 819, 1236, 1255, 1310, 1344, and 1369), and none of these occurrences is represented in the corresponding graphs.

Things become more complex when an adverb can present two or more senses, and thus may correspond to different syntactic-semantic constructions. The adverb *assim* "so" can function either as a sentence-internal **manner** adverb with an anaphoric value (e.g. *Ele fez isto assim = dessa maneira* "He did that in that way"); in that construction, it can also be an anaphor for an adjective or another predicate in a post-copula sentence:

(38) <id=157> *As pessoas grandes são assim.* "Grown-ups are like_that"

or it can function as a sentence-external modifier:

(39) <id=107> *Assim, ... perguntou-me bruscamente ...* "Then/Thus, [he] asked me abruptly"

This adverb appears 25 times in the corpus but is omitted from the graphs in most cases (e.g., in sentences with IDs 18, 22, 25, 174, 211, 264, 286, 397, 417, 466, ...). It is generally retained in the graph only when it occurs in a post-copular, predicative position, functioning anaphorically to replace a previously expressed adjective or predicate, as illustrated in (38). In such cases, it is parsed as an adjectival predicate, and that occurs seven times. There is only one instance where it functions as a **manner** anaphor, as in (id=1257): *E, caminhando assim, eu descobri o poço.* "And, walking like_that, I discovered the well." Finally, we observe the concessive multiword expression *mesmo assim* "even so", where its representation as a **manner** adjunct is clearly inadequate:

(40) <id=630> *Admira-me mesmo assim!* "Admire me, even so!"
→ `(a / admirar :manner (a / assim :mod (m / mesmo)))`

Such examples could be multiplied *ad libitum*.

7 Conclusion

This paper examined the semantic representation of adverbs through a comparative analysis of Abstract Meaning Representation (AMR), as applied to the Brazilian Portuguese edition of *O Pequeno Príncipe* [2], and the corresponding treatment proposed within the Lexicalized Meaning Representation (LMR) framework [16, 11], focusing on its application to the European Portuguese edition of the same novel, *O Principezinho* [12]. As a reference for the original AMR framework, we also considered the English AMR-annotated version of *The Little Prince* [9], in conjunction with the official AMR guidelines [10].

The study has revealed the challenges involved in capturing the semantic contributions of adverbial constructions across both frameworks.

The analysis highlights the underrepresentation and sometimes inadequate semantic parsing of adverbs in AMR, particularly with regard to syntactic-semantic alignment and their interaction with other sentence constituents. Conversely, LMR offers a more linguistically-informed approach grounded in syntactic theory, enabling a richer and more fine-grained encoding of adverbial meaning. However, the paper also notes that LMR faces its own set of theoretical and practical challenges, such as the need for consistent annotation practices and comprehensive guidelines to account for the semantic variety and syntactic flexibility of adverbs.

Overall, the findings suggest that adverbs – despite their frequency and functional importance in natural language – remain a relatively neglected category in semantic parsing. The paper advocates for a more robust theoretical and computational treatment of adverbs in semantic representation frameworks. Future work should aim to extend and refine the annotation guidelines for LMR, improve tool support for automatic parsing, and explore the integration of such enhancements in multilingual and cross-framework scenarios.

References

1 Angus Addlesee and Marco Damonte. Understanding disrupted sentences using underspecified abstract meaning representation. *Amazon Science*, 2023. URL: https://tinyurl.com/4ts6eb7t.

2 Rafael Anchiêta and Thiago Pardo. Towards AMR-BR: A SemBank for Brazilian Portuguese Language. In *LREC 2018*. ELRA, 2018. URL: https://www.aclweb.org/anthology/L18-1157.

3 Rafael Anchiêta. *Abstract Meaning Representation Parsing for the Brazilian Portuguese Language*. PhD thesis, Universidade de São Paulo, 2020.

4 Rafael Anchiêta and Thiago Pardo. An AMR-based Representation for Brazilian Portuguese. In *LREC 2018*. ELRA, 2018.

5 Rafael Anchiêta and Thiago Pardo. Rule-Based AMR Parsing for Brazilian Portuguese. In *7th Brazilian Conf. Intelligent Systems (BRACIS)*, pages 490–495. IEEE, 2018.

6 Rafael Anchiêta and Thiago Pardo. Bio AMR: A Biomedical Abstract Meaning Representation Corpus. In *LREC 2022*, pages 1234–1240, 2022.

7 Rafael T. Anchiêta, Marco A. S. Cabezudo, and Thiago A. S. Pardo. SEMA: An Extended Semantic Evaluation Metric for AMR. In *ACL 2019*. Springer Int. Publishing, 2019. arXiv:1905.12069.

8 Xuefeng Bai, Yulong Chen, Linfeng Song, and Yue Zhang. Semantic Representation for Dialogue Modeling. In *ACL 2021)*, pages 4430–4445. ACL, 2021. doi:10.18653/v1/2021.acl-long.342.

9 Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract Meaning Representation for Sembanking. In *ACL 2013*, pages 178–186, 2013. URL: https://aclanthology.org/W13-2322/.

10 Laura Banarescu et al. Abstract Meaning Representation (AMR) 1.2.6 Specification. https://github.com/amrisi/amr-guidelines/blob/master/amr.md, 2019.

11 Jorge Baptista. Lexical Meaning Representation – Guidelines. Technical report, University of Algarve/INESC-ID Lisboa, 2024. URL: https://gitlab.hlt.inesc-id.pt/u000803/lmr4pt/.

12 Jorge Baptista. LMR4PT – Principezinho. Technical report, University of Algarve/INESC-ID Lisboa, 2024. URL: https://gitlab.hlt.inesc-id.pt/u000803/lmr4pt/.

13 Jorge Baptista and Nuno Mamede. *Dicionário Gramatical de Verbos do Português*. Universidade do Algarve, 2020.

14 Jorge Baptista and Nuno Mamede. Syntactic Transformations in Rule-Based Parsing of Support Verb Constructions: Examples from European Portuguese. In *9th Symposium on Languages, Applications and Technologies (SLATE 2020)*, volume 83 of *OASIcs*, pages 11:1–11:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/OASIcs.SLATE.2020.11.

15 Jorge Baptista, Sónia Reis, Pedro A Santos, and João Dias. Comparação de anotações AMR entre línguas: problemas e soluções. *Revista da Associação Portuguesa de Linguística*, 1(11):27–54, 2024.

16 Jorge Baptista, Sónia Reis, João Dias, and Pedro Santos. Lexicalized Meaning Representation (LMR). In *5th Int. Workshop on Designing Meaning Representations @ LREC-COLING 2024*, pages 101–111, Torino, Italia, 2024. ELRA and ICCL. URL: https://aclanthology.org/2024.dmr-1.11/.

17 Michele Bevilacqua, Rexhina Blloshmi, and Roberto Navigli. One Spring to Rule Them Both: Semantic and Syntactic Parsing with Spring. In *ACL 2021*, pages 1545–1557. ACL, 2021.

18 Rexhina Blloshmi, Michele Bevilacqua, and Roberto Navigli. XL-AMR: Enabling Cross-Lingual AMR Parsing with Transfer Learning Techniques. In *2020 EMNLP 2020*, pages 2487–2500. ACL, 2020.

19 Claire Bonial, Mitchell Abrams, David Traum, and Clare Voss. Builder, we have done it: Evaluating & Extending Dialogue-AMR NLU Pipeline for Two Collaborative Domains. In *14th Int. Conf. Computational Semantics (IWCS)*, pages 173–183. ACL, 2021. URL: https://aclanthology.org/2021.iwcs-1.17/.

20 Matheus Cabezudo and Thiago Pardo. Towards Building an AMR Corpus for Brazilian Portuguese. In *LREC 2020*, pages 236–244. ELRA, 2020.

21 Deng Cai and Wai Lam. AMR Parsing via Graph-sequence Iterative Inference. In *ACL 2020*, pages 1290–1301. ACL, 2020. doi:10.18653/V1/2020.ACL-MAIN.119.

22 Shu Cai and Kevin Knight. Smatch: An Evaluation Metric for Semantic Feature Structures. In *ACL 2013*, pages 748–752. ACL, 2013. URL: https://aclanthology.org/P13-2131/.

23 Woo Suk Choi, Yu-Jung Heo, and Byoung-Tak Zhang. SGRAM: Improving Scene Graph Parsing via Abstract Meaning Representation, 2022. doi:10.48550/arXiv.2210.08675.

24 Marco Damonte and Shay B. Cohen. Cross-lingual Abstract Meaning Representation Parsing. In *ACL 2018 Conf. of the North American Chapter of the ACL: Human Language Technologies, Volume 1 (Long Papers)*, pages 1146–1155. ACL, 2018. doi:10.18653/V1/N18-1104.

25 Marco Damonte, Shay B. Cohen, and Alessandro Sordoni. An Incremental Parser for Abstract Meaning Representation. In *ACL 2017*, pages 536–546. ACL, 2017.

26 Yang Deng, Wenxuan Zhang, Weiwen Xu, Ying Shen, and Wai Lam. Nonfactoid Question Answering as Query-Focused Summarization With Graph-Enhanced Multihop Inference. *IEEE Transactions on Neural Networks and Learning Systems*, 35(8):11231–11245, 2024. doi:10.1109/TNNLS.2023.3258413.

27 Shibhansh Dohare, Harish Karnick, and Vivek Gupta. Text Summarization using Abstract Meaning Representation, 2017. arXiv:1706.01678.

28 Ermal Elbasani and Jeong-Dong Kim. AMR-CNN: Abstract Meaning Representation with Convolution Neural Network for Toxic Content Detection. *J. Web Engineering*, 21(3):677–692, 2022. doi:10.13052/jwe1540-9589.2135.

29 Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A. Smith. A Discriminative Graph-Based Parser for the Abstract Meaning Representation. In *ACL 2014 (Volume 1: Long Papers)*, pages 1426–1436. ACL, 2014. doi:10.3115/V1/P14-1134.

30 Sreyan Ghosh, Utkarsh Tyagi, Sonal Kumar, C. K. Evuru, S Ramaneswaran, S Sakshi, and Dinesh Manocha. ABEX: Data Augmentation for Low-Resource NLU via Expanding Abstract Descriptions, 2024. doi:10.48550/arXiv.2406.04286.

31 Shubham Gupta, Narendra Yadav, Suman Kundu, and Sainathreddy Sankepally. FakEDAMR: Fake News Detection Using Abstract Meaning Representation Network. In *Complex Networks & Their Applications XII*, pages 308–319. Springer Nature Switzerland, 2024.

32 Caroline; Hagège, Jorge; Baptista, and Nuno Mamede. Identificação, classificação e normalização de expressões temporais do português: A experiência do Segundo HAREM e o futuro. In Cristina; Mota and Diana Santos, editors, *Desafios na avaliação conjunta do reconhecimento de entidades mencionadas: Actas do Encontro do Segundo HAREM (Aveiro, 11 de Setembro de 2008)*, chapter 2, pages 33–54. Linguateca, 2009.

33 Caroline Hagège, Jorge Baptista, and Nuno Mamede. Caracterização e processamento de expressões temporais em português. *Linguamática*, 2(1):63–76, 2010. URL: http://www.linguamatica.com/index.php/linguamatica/article/view/47.

34 Zellig Harris. *Notes du Cours de Syntaxe*. Seuil, Paris, 1976. (edited by Maurice Gross).

35 Márcio Inácio, Marco Antonio Sobrevilla Cabezudo, Renata Ramisch, Ariani Di Felippo, and Thiago Pardo. The AMR-PT Corpus and the Semantic Annotation of Challenging Sentences from Journalistic and Opinion Texts. *D.E.L.T.A.*, 39(3):1–31, 2023. doi:10.1590/1678-460X202339355159.

36 Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. Neural AMR: Sequence-to-Sequence Models for Parsing and Generation. In *ACL 2017*, pages 146–157. ACL, 2017. doi:10.18653/V1/P17-1014.

37 Kenneth Lai, Richard Brutti, Lucia Donatelli, and James Pustejovsky. Encoding Gesture in Multimodal Dialogue: Creating a Corpus of Multimodal AMR. In *LREC-COLING 2024*, pages 5806–5818. ELRA and ICCL, 2024. URL: https://aclanthology.org/2024.lrec-main.515/.

38 Changmao Li and Jeffrey Flanigan. Improving Neural Machine Translation with the Abstract Meaning Representation by Combining Graph and Sequence Transformers. In *2nd Workshop on Deep Learning on Graphs for Natural Language Processing (DLG4NLP 2022)*, pages 12–21. ACL, 2022. doi:10.18653/v1/2022.dlg4nlp-1.2.

39 Xiang Li, Thien Huu Nguyen, Kai Cao, and Ralph Grishman. Improving Event Detection with Abstract Meaning Representation. In *ACL 1st Workshop on Computing News Storylines*, pages 11–15, 2015.

40 Chunchuan Lyu and Ivan Titov. AMR Parsing as Graph Prediction with Latent Alignment. In *ACL 2018*, pages 397–407. ACL, 2018. doi:10.18653/V1/P18-1037.

41 Fukun Ma, Xuming Hu, Aiwei Liu, Yawen Yang, Shuang Li, Philip S. Yu, and Lijie Wen. AMR-based Network for Aspect-based Sentiment Analysis. In *ACL 2023*, pages 322–337. ACL, 2023. doi:10.18653/v1/2023.acl-long.19.

42 Nuno Mamede, Jorge Baptista, Cláudio Diniz, and Vera Cabarrão. STRING: A Hybrid Statistical and Rule-Based Natural Language Processing Chain for Portuguese. In *ACL 10th Int. Conf. Computational Processing of Portuguese (PROPOR 2012)*, 2012. Demo session.

43 Behrooz Mansouri. Survey of Abstract Meaning Representation: Then, Now, Future, 2025. arXiv:2505.03229.

44 Christian Molinier and Françoise Levrier. *Grammaire des adverbes: description des formes en* -ment. Droz, Genève, 2000.

45 Izabela Müller and Jorge Baptista. Focus Adverbs in Portuguese. In *II Cong Int. Humanidades UAb. Desafios da transformação digital*, page [to appear], Lisboa, Portugal, 2025. Universidade Aberta.

46 Izabela Müller, Nuno Mamede, and Jorge Baptista. Advérbios Compostos do Português do Brasil. *Revista da Associação Portuguesa de Linguística*, 1(10):230–250, 2023. doi:10.26334/2183-9077/rapln10ano2023a13.

47 Tim O'Gorman, Michael Regan, Kira Griffitt, Ulf Hermjakob, Kevin Knight, and Martha Palmer. AMR Beyond the Sentence: the Multi-sentence AMR corpus. In *ACL 27th Int. Conf. Computational Linguistics*, pages 3693–3702. ACL, 2018. URL: https://aclanthology.org/C18-1313.

48 Hyeonchu Park, Byungjun Kim, and Bugeun Kim. DART: An AIGT Detector using AMR of Rephrased Text. *arXiv preprint arXiv:2412.11517*, 2024. doi:10.48550/arXiv.2412.11517.

49 Hao Peng, Sam Thomson, and Noah A. Smith. Deep Semantic Parsing with Syntactic Clues. In *ACL 2017*, pages 1831–1841. ACL, 2017.

50 Haoyi Qiu, Kung-Hsiang Huang, Jingnong Qu, and Nanyun Peng. AMRFact: Enhancing summarization factuality evaluation with AMR-driven negative samples generation. *arXiv preprint arXiv:2311.09521*, 2023.

51 Sudha Rao, Daniel Marcu, Kevin Knight, and Hal Daumé III. Biomedical Event Extraction using Abstract Meaning Representation. In *BioNLP 2017*, pages 126–135. ACL, 2017. doi:10.18653/v1/W17-2315.

52 Nathan Schneider, Jeffrey Flanigan, and Tim O'Gorman. The Logic of AMR: Practical, Unified, Graph-Based Sentence Semantics for NLP. In *ACL 2015*, pages 4–5. ACL, 2015. doi:10.3115/v1/N15-4003.

53 Nikolaus Schrack, Ruixiang Cui, Hugo López, and Daniel Hershcovich. Can AMR Assist Legal and Logical Reasoning? In *Findings of the ACL: EMNLP 2022*, pages 1555–1568. ACL, 2022. doi:10.18653/v1/2022.findings-emnlp.112.

54 Luana Seno, Rafael Anchiêta, and Thiago Pardo. Cross-lingual AMR Parsing for Brazilian Portuguese. In *14th Int. Conf. Computational Processing of Portuguese (PROPOR)*, pages 64–74. Springer, 2021.

55 Ziyi Shou, Yuxin Jiang, and Fangzhen Lin. AMR-DA: Data Augmentation by Abstract Meaning Representation. In *Findings of the ACL: ACL 2022*, pages 3082–3098. ACL, 2022. doi:10.18653/v1/2022.findings-acl.244.

56 Marco Antonio Sobrevilla Cabezudo, Rafael Anchieta, and Thiago Pardo. Comparison of Cross-lingual Strategies for AMR-to-Brazilian Portuguese Generation. *Research Square*, 2022. doi:10.21203/rs.3.rs-2221954/v1.

57 Andreas Vlachos and H. Hardi. Guided Neural Language Generation for Abstractive Summarization Using Abstract Meaning Representation. *arXiv preprint arXiv:1808.09160*, 2018.

58 Chuan Wang and Nianwen Xue. A Transition-based Algorithm for AMR Parsing. In *ACL 2015 Conf. of the North American Chapter of the ACL: Human Language Technologies*, pages 366–375. ACL, 2015.

59 Chuan Wang and Nianwen Xue. Exploring Transition-based AMR Parsing with Sentence Rewriting. In *EMNLP 2015*, pages 868–877. ACL, 2015.

60 Tianming Wang, Xiaojun Wan, and Hanqi Jin. AMR-To-Text Generation with Graph Transformer. *Transactions of the ACL*, 8:19–33, 2020. doi:10.1162/tacl_a_00297.

61 Shira Wein and Julia Bonn. Comparing UMR and Cross-lingual Adaptations of AMR. In *4th Int. Workshop on Designing Meaning Representations*, pages 23–33. ACL, 2023. URL: https://aclanthology.org/2023.dmr-1.3.

62 Runxin Xu, Peiyi Wang, Tianyu Liu, Shuang Zeng, Baobao Chang, and Zhifang Sui. A two-stream AMR-enhanced model for document-level event argument extraction. *arXiv preprint arXiv:2205.00241*, 2022. doi:10.48550/arXiv.2205.00241.

63 Nianwen Xue, Ondřej Bojar, Jan Hajič, Martha Palmer, Zdeňka Urešová, and Xiuhong Zhang. Not an Interlingua, But Close: Comparison of English AMRs to Chinese and Czech. In *LREC 20414*, pages 1765–1772. ELRA, 2014. URL: http://www.lrec-conf.org/proceedings/lrec2014/pdf/384_Paper.pdf.

Stepwise Source, a Supporting Tool for Source Code Demonstration

João Santos
ALGORITMI Research Centre/LASI - DI, University of Minho, Braga, Portugal

Alvaro Costa Neto
ALGORITMI Research Centre/LASI - DI, University of Minho, Braga, Portugal
Research Centre in Digitalization and Intelligent Robotics (CeDRI),
Laboratório Associado para a Sustentabilidade e Tecnologia em Regiões de Montanha (SusTEC),
Instituto Politécnico de Bragança, Portugal
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, Barretos, Brazil

Pedro Rangel Henriques
ALGORITMI Research Centre/LASI - DI, University of Minho, Braga, Portugal

Abstract

The difficulties in teaching and learning computer programming remain a pressing issue to this day. Several studies and tools have been developed over the years to tackle this challenge from many different points-of-view. One of the biggest tools an educator has to support him in a classroom is the progressive explanation of how a source code is constructed and what effects each of its parts has on the overall result. Attempts to translate this live-directed tool to an on-line experience is usually time-consuming and lacking in features. In order to tackle this concern, a tool to create piecewise source code writing demonstrations was developed – Stepwise Source. The main idea behind this application is to allow step-by-step explanation of a source code construction, along with any relevant annotations and automatically assessed challenges that an educator may add. By providing a dynamic platform for both students and lecturers, this software aims to improve the teaching and learning of computer programming, while trying to imitate the information flow of a live lecture, with the added benefit of student-directed pace of explanation. Through interactive guidance and automated assessment, this tool has the potential to foster a deeper understanding of computational principles and promote proficiency in programming skills.

2012 ACM Subject Classification Applied computing → Interactive learning environments

Keywords and phrases Computer Programming Education, Source Code Demonstration, Education Technology

Digital Object Identifier 10.4230/OASIcs.SLATE.2025.10

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UID/00319/2023. The work of Alvaro was supported by national funds: UID/05757 - Research Centre in Digitalization and Intelligent Robotics (CeDRI); and SusTEC, LA/P/0007/2020 (DOI: 10.54499/LA/P/0007/2020)

1 Introduction

Teaching computer programming presents several challenges. On the educator's side, issues can arise from choosing an inappropriate programming language, focusing too much on the language itself rather than fostering computational thinking, or failing to tailor their teaching methods to meet the diverse needs of learners. [12, 3, 4, 9] From the student's perspective, difficulties may stem from a lack of foundational skills, such as weak mathematical or abstract thinking abilities, or trouble interpreting and breaking problems down into smaller, manageable tasks. [1, 9, 4, 16] Additionally, students often face challenges related to motivation and persistence, including giving up early when encountering initial obstacles or struggling to develop autonomy in their learning process. [12, 2, 17, 1]

In order to address some of the identified challenges, various programming education tools have been developed:

- *Automatic assessment* tools, such as Codeboard [13] and ACEGrader [6], for instance, focus on enhancing student autonomy, allowing learners to track their own progress, and, often, enabling educators to monitor their development as well;
- *Source code animation* tools, as PythonTutor [10], help students visualize how code executes, improving their ability to grasp abstract concepts and increasing independence in learning;

Additionally, there are tools that fall into the category of *source code demonstration*. These are designed to support students in writing code by providing partially completed templates that they must finish. Khan Academy [11] features a solution along these lines in its exercises section. This approach not only fosters autonomy and offers a more hands-on learning experience, but it may also aid students in learning the process of computational thinking by helping them understand how to break down problems into smaller parts, thus boosting motivation.

This paper is divided in six sections: after this introduction, Section 2 presents the main ideas and concepts behind Stepwise Source. Section 3 introduces Stepwise Source's Domain-Specific Language (DSL) that supports the functioning of the entire application. Section 4 refers to the main technologies and techniques that were employed to build Stepwise Source. Section 5 shows how the application can be used and presents the feedback obtained from future programming educators. Finally, Section 6 finishes the paper with an overview and proposals for future improvements and projects.

2 Stepwise Source, Proposal

Stepwise Source, accessible at `http://stepwisesource.epl.di.uminho.pt`, finds a new path among the existing tools, by allowing educators to create piece-wise demonstrations of how a source code is written. By adding comments and alternate paths of explanation, teachers are able to construct complex explanations in an accessible form, that students can navigate on their own pace. Overall, it aims to simulate the explanation and construction of source code in a progressive manner, similar – albeit not identical – to a live demonstration. The main difference relies on the fact that a student itself will determine its rhythm by controlling the progression or regression of steps according to his or her preferences.

In order to do so, the system requires input from the educator in the form of a presentation. This presentation consists in a sequence of slides that contains the *source code*, whose construction will be demonstrated and explained to the student, and *annotations* to this source code that will offer guidance to the student on how to carry out said construction. This source code can be annotated with specific commands that define how the presentation will unfold, detailing annotated explanations, possible variations, and even questions for the students.

The overall system architecture is depicted in Figure 1. As can be seen, the source code whose construction is demonstrated is not parsed *per se*, only Stepwise Source's commands are processed. The results generated by the parser are compiled into an internal representation that, in turn, is stored in a database for quick construction of the navigation web page. The student interacts with the system both by navigating through the presentation, and by answering exercises that might be added to the slides.

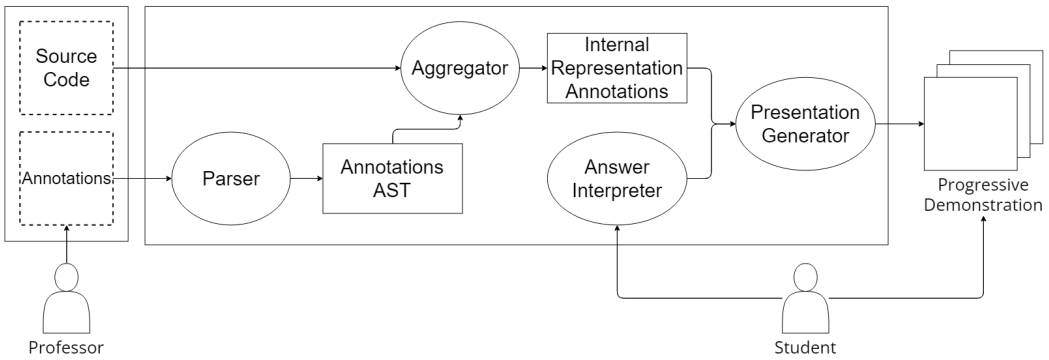


Figure 1 Stepwise Source architecture.

Listing 1 Syntax for a slide with an optional description.

```
\slide{<ID>}{<TITLE>}
\description{<DESCRIPTION>}
\begin
    <CONTENT>
\end
```

3 Stepwise Source, DSL

A DSL with a syntax similar to that of LaTeX, named *Stepwise Language*, was designed specifically for Stepwise Source. The syntactical similarity to LaTeX was chosen for its wide use among computer programming teachers for document preparation. It aims to facilitate teachers' presentation input, allowing for all features to be accessible through simple text.

3.1 Overview and Use

Stepwise Language was designed to empower educators in teaching computational thinking and programming through the creation of interactive and visually appealing presentations. Central to this DSL are several commands that provide educators with the tools to structure their lessons, style what is covered, integrate interactive components, and regulate the information flow. This section provides a detailed exploration of these commands, including their syntax, components, and practical applications.

3.1.1 Defining a Slide

At the core of any presentation crafted using this DSL lies the concept of a slide, an essential element that encapsulates both structure and content (see Listing 1 for its declaration syntax). Its creation begins with the `\slide` command to define a new slide. Each slide is designated by a unique identifier (`ID`) that facilitates reference throughout the presentation, ensuring that content is organized and coherent.

The title of the slide, represented by the `TITLE` parameter, provides a succinct overview of the slide's focus. It sets the stage for the audience, offering a glimpse into the topic being addressed. Complementing this, an optional description can be included through the `\description` command, adding an extra layer of context and allowing for detailed explanation of the slide. This description elaborates on the content, helping the audience to better understand the significance of what they are about to learn.

Listing 2 Example of a slide on variable declaration.

```
% This is a comment that spans to the end of the line.
\slide{Intro}{Introduction to Variables}
\description{Variables are used to store values that can be accessed
    ↪ later on.}
\begin
int main(int argc, char *argv[]) \{
    int x = 5;
    int y = x + 10;
\}
\end
```

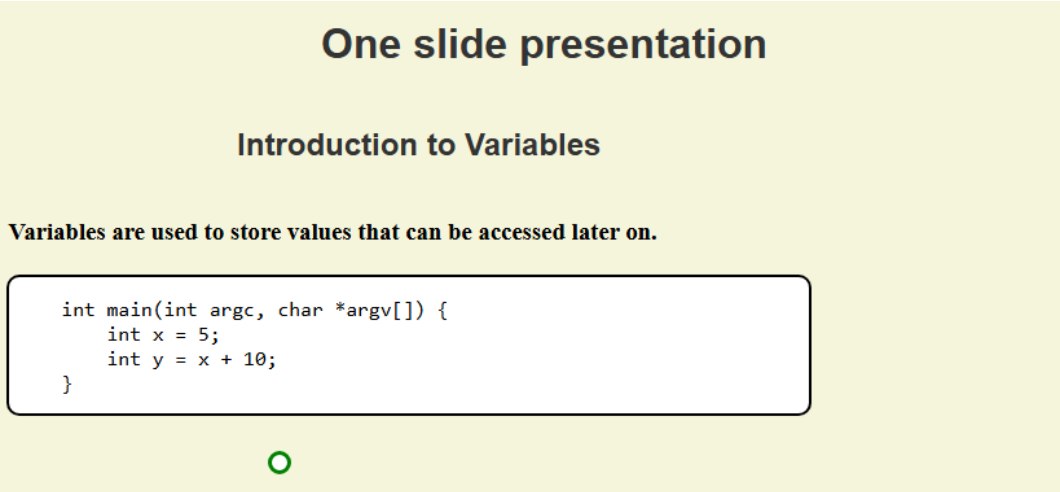


Figure 2 Simple slide with text content (from Listing 2).

In order to encapsulate the main content of the slide, the `\begin` and `\end` commands create a block where the primary material resides. This content can consist of various elements, including code snippets, explanatory annotation text, and exercises that can be evaluated through regular expressions. The ability to insert diverse types of content within this structure allows educators to tailor their presentations to suit different teaching styles and student needs.

As a running example for the rest of this section, a slide designed to introduce the concept of variable declaration in computer programming using the C language is presented in Listing 2. In this example, the slide (whose `ID` is `Intro`) begins with a clear title that indicates its purpose (`Introduction to Variables`), while the description succinctly explains the function of variables in programming. The block of content then demonstrates the concept with concrete examples, thereby supporting student comprehension. The rendering of this slide can be seen in Figure 2.

3.1.2 Styling

In addition to the foundational structure of slides, the visual presentation of content plays a vital role in effective communication. To enhance the clarity and appeal of presentations, the DSL offers two distinct commands that specify visual styles for annotations: `\newreactive` and `\newfixed`. Both commands have the essential function of defining custom styles which can be applied to the content's annotations, giving educators a versatile way of enhancing

Listing 3 Syntax for the declaration of new styles.

```
\newreactive{<ID>}{<KEY>; <ATTRIBUTE>=<VALUE>; ...}
\newfixed{<ID>}{<KEY>; <ATTRIBUTE>=<VALUE>; ...}
```

Listing 4 Example of use for the `\newreactive` and `\newfixed` commands.

```
\newreactive{highlight}{color: green; bold}
\newfixed{removed}{color: red; strikethrough}

\slide{Intro}{Introduction to Variables}
\description{Variables are used to...}
\begin
int main(int argc, char *argv[]) \{
    \highlight{int x = 5;}{Initial value of x.}
    \removed{int x = x + 10;}{This is wrong!}
\}
\end
```

the visual appearance of their presentations. They ensure that particular aspects of the presentation stand out and call attention to important information by establishing specific styles. These instructions contrast in how and when they deliver the styles to the audience, even though they both aim to improve content presentation.

The `\newfixed` command ensures that the applied styles are visible at all times, providing immediate and constant context to the audience. This approach is useful in scenarios where frequent reference to important information is necessary, offering a stable and predictable presentation experience.

On the other hand, the `\newreactive` command introduces a dynamic element by revealing styles only upon user interaction, such as hovering or clicking on an element. This feature reduces initial visual clutter, creating a more simplistic experience. It is particularly advantageous when a gradual reveal of information is desired, allowing educators to control the pacing of content delivery.

The syntax for defining these styles is presented in Listing 3. In both declarations, `ID` serves as the identifier for each styling command definition, allowing it to be referenced throughout the presentation for consistent application. The `KEY` and `ATTRIBUTE` components consist of a set of CSS-like styling rules that can encompass various attributes such as text colour, font weight, and other visual properties that contribute to the overall aesthetics of the content[1].

To illustrate the application of these commands, Listing 4 adds them to the running example of Listing 2. The `highlight` style (applied on line 8 via the `\highlight` command) emphasizes key information using green and bold text, while the `removed` style (line 9, via `\removed`) can effectively signal to students which elements have been removed from prior discussions with a red colour and strike-through effect. It is important to note that the explanations for each styled annotation is given as a second parameter for each of the style commands. The slide for this example can be seen in Figure 3. Teachers can also define macros that use a style and a fixed message to avoid repetition when creating slides, such as demonstrated by Listing 5. The Error command is an example of a macro that can be used throughout the presentation, with the message already defined. Whereas, with the Remove Style command, the message needs to be defined each time it is used.

[1] The list of keys and attributes is presented in Subsection 3.1.3, Table 1.

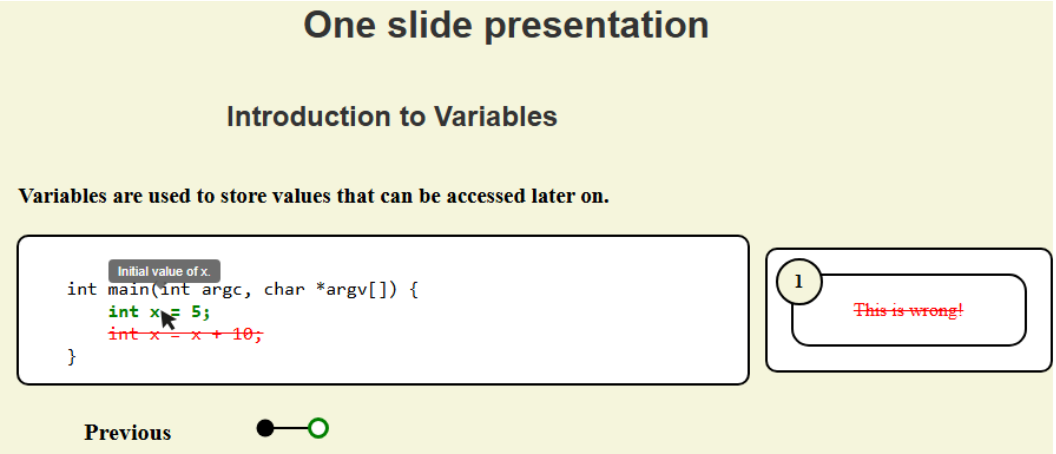


Figure 3 Slide with both fixed and reactive annotations (from Listing 4).

Listing 5 Example of use for macros.

```
\newfixed{removed}{text-color: red; strikethrough;}

% Defines the macro "error" with the message "This is wrong!".
\removed{error}{This is wrong!}

\slide{Intro}{Introduction to Variables}
\description{Variables are used to...}
\begin
int main(int argc, char *argv[]) \{
    \removed{int 2x = 5;}{Variable names cannot start with digits.}
    % This macro adds the message "This is wrong!" automatically.
    \error{int x = x + 10;}
\}
\end
```

Such visual distinctions serve to guide students' attention to significant aspects of the material, akin to what a teacher would do while explaining a snippet of code in class.

3.1.3 Styling Keys

Stepwise Source's DSL employs a CSS-like syntax for defining styles. This familiar structure assists educators who may have prior experience with web development, allowing them to focus on content creation rather than navigating complex formatting rules. Educators can apply colours through specific attributes such as `text-color`, `background-color`, and `border-color`. The colours themselves can be specified either by their common names (e.g. `red`, `green`, and `blue`) or by hexadecimal codes in the standard format `#RRGGBB`.

Table 1 provides a comprehensive overview of the DSL's styling keys and attributes, illustrating how they correspond to standard CSS properties. The syntax chosen was not CSS but a similar one because the DSL could not allow for all of CSS symbols.

Table 1 Translation of DSL styling keys and attributes to CSS.

DSL	CSS Equivalent	Description
`text-color: red`	`color: red`	Changes the text color to red, drawing attention to highlighted text.
`background-color: blue`	`background-color: blue`	Sets the background color to blue, useful for emphasizing sections.
`border-color: #FA0C25`	`border-color: #FA0C25`	Defines the color of an element's border as a specific hex code, highlighting areas effectively.
`bold`	`font-weight: bold`	Highlights key terms or code for emphasis.
`strikethrough`	`text-decoration: line-through`	Visually indicates removed or outdated content.
`underline`	`text-decoration: underline`	Can be used to underscore important points or warnings.
`text-style`	`font-style`	Applies styling to text, with options for `italic` and `oblique` for emphasis or differentiation.
`text-size: 16`	`font-size: 16px`	Sets the font size of the text, allowing for adjustments in readability.
`text-align: center`	`text-align: center`	Aligns the text to the centre of the element, useful for headings or important messages.

Listing 6 Syntax for exercises creation.

```
1 \exercise{/<REGEX>/}{<PROMPT>}
```

3.1.4 Exercises

A distinctive feature of this DSL is its ability to stimulate student engagement through the `\exercise` command, whose syntax is shown in Listing 6. This command creates interactive elements within the presentation, fostering active participation and self-assessment among students.

In this syntax, `REGEX` is a regular expression[2], to be defined by the tutor, that automatically validates the input provided by the student, ensuring the accuracy of their responses. The `PROMPT` serves as the exercise's statement that is presented to the student, guiding their interaction and prompting them to reflect on the content of the presentation.

In order to illustrate the application of interactive exercises, consider the example in Listing 7, which allows for immediate student feedback. In this instance, the command:
`\exercise{/[0-9]+/}{Complete the code...}`
prompts students to input a valid number, checking their response against the specified regular expression. This interactivity not only enhances student involvement but also enables them to self-assess their understanding in real-time.

Interactive exercises foster an engaging learning environment, allowing students to test their comprehension and receive immediate feedback based on their inputs. Such features not only promote active learning but also encourage students to engage more deeply with the material, ultimately enriching the overall educational experience. An example of an exercise correctly answered (green background indicates a correct answer, red a wrong one) can be seen in Figure 4.

[2] JavaScript standard `https://www.w3schools.com/jsref/jsref_obj_regexp.asp`

Listing 7 Example of an interactive exercise.

```
\slide{Intro}{Introduction to Variables}
\description{Variables are used to...}
\begin
int main(int argc, char *argv[]) \{
    int x = \exercise{/[0-9]+/}{Complete the code by assigning an integer
    ↪  value to the variable "x".};
    int y = x + 10;
\}
\end
```

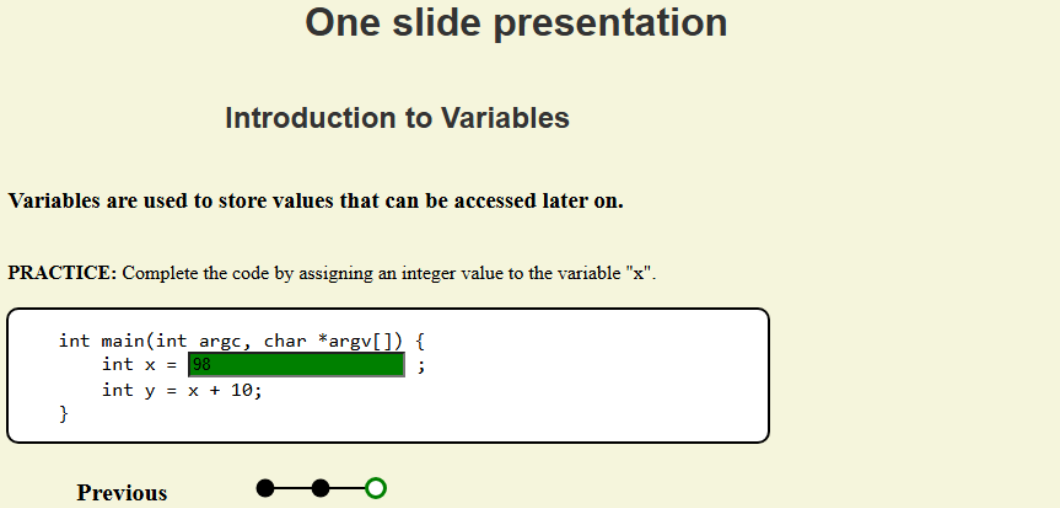


Figure 4 Slide with an exercise correctly answered (from Listing 7).

3.1.5 Presentation Flow

Controlling the flow of the presentation is essential to effectively convey information. The mandatory `\order` command enables this by allowing educators to dictate the sequence of slides, thereby creating dynamic, ans possibly branching paths that are tailored to their audience's needs.

Listing 8 shows the syntax of the `\order` command. `SLIDE_LIST` represents the list of identifiers that dictates the order in which the slides will be presented. It is specified by a comma-separated list of slide IDs. A *fork feature* allows educators to create branches in the presentation using the *pipe operator* (`|`), enabling alternative paths for the flow of explanation. This would be used in cases which a topic demands not only demonstration of correct implementation, but also incorrect ones. Another possible use for the fork feature is the explanation of alternatives to one topic, such as the different repetition primitives in a language that can be used to implement loops.

Listing 9 illustrates the functionality of the `\order` command. It defines a presentation flow that starts with the slide `Intro`, followed by `Variables`, where the presentation can diverge to either provide examples (`Examples`) or conduct a quiz (`Quiz`). Finally, the paths merge and concludes the presentation with the slide `Conclusion`. The actual navigation graph can be seen in Figure 5. By utilizing the `\order` command, educators can effectively manage the presentation's flow, ensuring that content is delivered in a manner that is both engaging and responsive to the students' dynamics.

Listing 8 Syntax for the `\order` command.

```
1 \order{<SLIDE_LIST>}
```

Listing 9 Example of a presentation containing five slides.

```
1 \slide{Intro}{Introduction to Variables} ...
2 \slide{Variables}{How to Declare Variables} ...
3 \slide{Examples}{Examples: Declaring Variables} ...
4 \slide{Quiz}{Quiz: Variable Declaration} ...
5 \slide{Conclusion}{The End} ...
6 \order{Intro, Variables, (Examples | Quiz), Conclusion}
```

3.2 Formal Definition

An Extended Backus-Naur Form (EBNF) grammar was written[3] in order to capture the syntax and structure required for Stepwise Language's specialized commands[4]. This section provides a detailed analysis of each production rule within the EBNF, starting with the overall structure of the language and proceeding through the components of each command, including slides, style definitions, macro declarations and presentation order.

The EBNF definition for Stepwise Language begins with the required `start` production rule, as seen in Listing 10. This initial rule is defined as a list of commands, including slide definitions (line 3), style (lines 4 and 5) and macro (line 6) declarations. After all the commands are set, the presentation order is defined.

The slide command has three main components, represented by non-terminal symbols: `slide`, `description` (optional), and `code_block`. The slide non-terminal rule, shown in Listing 11, maps to the slide command with an identifier and a title. The slide keyword is defined in both Portuguese and English[5] (line 2), while the slide identifier follows typical identifier syntax: a letter followed by alphanumeric characters (line 3). Titles use the `TEXT` terminal symbol (line 4), which matches either escaped curly brackets (`\}`) or any character with the exception of a closing curly bracket.

The `description` command, shown in Listing 11, uses the same `TEXT` terminal for specifying text content. The `code_block` command, meanwhile, encapsulates `content` within `BEGIN` and `END` keywords, as shown in Listing 11.

Inside a slide's content, it is possible to include both source code (in any programming language language) and Stepwise Language commands. The EBNF rule for `content`, shown in Listing 13, includes three commands. The first is the `EXERCISE` command, which accepts a regular expression (`REGEX`, defined on line 6) and a text value (line 3) as arguments. The `NOT_END` commands (lines 4 and 5) are declared to ensure that commands do not accidentally signal the end of the slide, as would happen if their identifiers were `end`. Instead, by using the terminal `NOT_END`, these command definitions operate within the slide's content without triggering its conclusion.

[3] The source code for the whole project including the language can be found here: `https://github.com/Yakari144/StepwiseSource`

[4] More specifically, the grammar was written in Lark's dialect for EBNF in order to both *design and implement* the parser.

[5] All keywords are defined in both languages. Identifier, by their own nature, can be written in any language that uses the English alphabet.

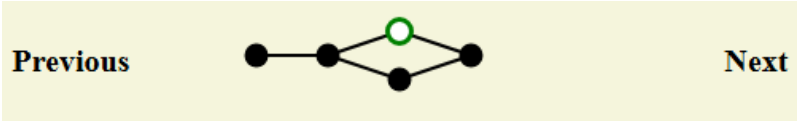


Figure 5 Navigation graph for the presentation of Listing 9.

Listing 10 Initial production rule.

```
start: commands
commands: command+ order
command: slide description? code_block
    | nreactive
    | nfixed
    | macro
```

Source code, defined by the `CODE` terminal, supports any kind of character and escaped symbols that are preceded with backslash (\). It is designed to allow for the use of any programming language, even those that require characters that are reserved for Stepwise Language, such as open and close curly brackets, and backslash itself. The listings in Section 3 that contain a running example of slides with a source code written in C show that, in order to use curly brackets to open and close C's blocks of code, it is necessary to escape them with backslash (`\{` and `\}`). This strategy, very common in other languages, even allows the use of Stepwise Language source code inside itself, as the content of a slide[6].

The `nreactive` and `nfixed` rules, detailed in Listing 14, share a similar syntax and only differ by their keyword. They define a unique identifier followed by a list of CSS-like styling rules, each separated by a semicolon.

The `macro` rule (Listing 15) is defined similarly to the other command rules, but without a specific keyword. The `MACRO` terminal captures identifiers that *do not conflict with the other command keywords using a negative lookahead to avoid them.* The remaining syntax aligns closely with that of the other commands.

The final command rule defined in the Stepwise Language's EBNF, `order`, is shown in Listing 16. This command uses a hierarchy of productions to handle command ordering and prioritization. The recursive structure self-manages precedence: parentheses and identifiers are evaluated first, followed by the pipe operator (`|`) and then commas, as the lowest priority.

4 Stepwise Source, Development

The selection of programming languages and tools for building Stepwise Source was crucial to lay a strong foundation for the system. Python [15] was chosen due to its flexibility, ease of use, and the wide range of libraries available. Among these libraries, Lark was selected for its ability to easily and rapidly construct a parser suited for Stepwise Language.

The implementation of Lark's Interpreter module enables a customized parsing process tailored to the needs of the DSL. Beyond basic command interpretation, the Interpreter class traverses the Abstract Syntax Tree (AST), handling each node to build the core data structures required by the parser, while ensuring efficient communication with the

[6] In a similar fashion to programming languages *bootstrapping*.

Listing 11 Slide declaration production rule.

```
slide: "\\" SLIDE "{" ID "}" "{" TEXT "}"
SLIDE: "diapositivo" | "slide"
ID: ("a".."z"|"A".."Z")("a".."z"|"A".."Z"|"_"|"0".."9")*
TEXT: /(((\\\})|([^}]))+/
```

Listing 12 Description production rule.

```
description: "\\" DESCRIPTION "{" TEXT "}"
```

database. The Interpreter class includes key properties for managing data that is essential to Stepwise Source's functionality, including commands and slides declarations, the current slide's identifier and content, and the order structure.

Beyond its internal properties, the Interpreter class also implements methods for each of the non-terminal symbols in the grammar. Beginning with the annotation style declarations, in order to convert the CSS-like syntax of the Stepwise Language into CSS by iteratively processing each token and adding the appropriate CSS formatting.

The presentation order command is handled by the functions responsible for storing the order of the presentation and parsing the slide order expression declared inside the `\order` command parameter. Since the operator precedences were structurally defined in the EBNF[7], the interpreter only has to collect and store the data. It checks each token, determining whether it represents a stopping point or a recursion case, structuring the data accordingly.

Every time a slide declaration is parsed, a new slide object is created and set as the current one. The Interpreter then iterates through the slide's contents, populating the slide object with properties. The annotated source code inside the slide's content is translated into HTML in order to style it accordingly, with each exercise command converted to an input text box.

4.1 Database

Given the importance of managing and retrieving complex data, choosing a suitable database was essential. *MongoDB* [14], a document-based database, was selected for its flexibility in handling the structured data generated by the parser.

In order to effectively organize the parsed data structures within MongoDB, the database system is divided into three collections: *presentations*, *annotations*, and *orders*. Each collection plays a specific role in managing the structured information resulting from Stepwise Language's parsing.

The *presentations* table stores presentations that contain an identifier, a name, the original text input used to generate the presentation, and a list of processed slides. Each slide within a presentation includes its identifier, title, description, generated HTML code, a list of annotation identifiers, and a list of exercises, where each exercise entry specifies its identifier, assessment regular expression, and question.

In the *annotations* table, each entry represents the annotations linked to a specific presentation. This table includes the presentation identifier it refers to, alongside a list of annotations. Each annotation entry has an identifier and a category attribute, classifying it

[7] These precedence rules are almost identical to usual arithmetic precedence definitions in regular programming languages.

Listing 13 Production rule that corresponds to the content of a slide.

```
code_block: "\\" BEGIN content "\\" END
content: (entry | code)+
entry: "\\" EXERCISE "{" REGEX "}" "{" TEXT "}"
    | "\\" NOT_END "{" content "}" "{" TEXT "}"
    | "\\" NOT_END "{" content "}"
REGEX: /\/(\\\/|[^\/])+\//
code: CODE
CODE: /(\\\\|\\\}|\\\{|[^\\\}\{])+/
```

Listing 14 Production rules to declare new styling commands.

```
nreactive: "\\" NEWREACTIVE "{" ID "}" "{" mycss (";" mycss)* "}"
nfixed: "\\" NEWFIXED "{" ID "}" "{" mycss (";" mycss)* "}"
mycss: TXT_COLOR ":" COLOR | BG_COLOR ":" COLOR | BORDER_COLOR ":" COLOR
    | BOLD | ITALIC | UNDERLINE | STRIKETHROUGH
    | TXT_SIZE ":" NUM | TXT_ALIGN ":" ALIGN
```

as either a "command" or "macro". Entries in the "command" category also include a type and a list of styling attributes, while those in the "macro" category contain the command that created it and the associated text.

The final table, *orders*, holds the structural order of each presentation. Each order entry includes the presentation identifier and an order string that defines the slide arrangement, such as:

```
[Beginning, [[Tips], [Exercise]], Syntax, Ending]
```

which, organizes slides by identifier. Nested lists in the order string represent forks, with each path in a fork following the same syntax as the main order. This organization allows each path to be processed as an independent presentation structure.

4.2 Backend

The backend serves as the intermediary layer, managing communication between the frontend, the parser, and the database. Built with *JavaScript* [7] and *Express.js* [8], the backend efficiently handles requests and processes data, whether it involves retrieving stored content from MongoDB or sending commands to be processed by the Python-based parser. This architecture enables the frontend to dynamically interact with the system, providing the necessary content to users without delays or inconsistencies.

4.3 Web Application

The frontend was developed with *Create React App* [5] due to its simplicity and efficiency. React's component-based architecture made it easier to manage dynamic content and consistency throughout the application. This framework allows the Web App to render and

Listing 15 Macro production rule.

```
macro: "\\" MACRO "{" ID "}" "{" TEXT "}"
MACRO: /(?!"newreactive"|"newfixed"|"order"|"novoreativo"|"novofixo"|"
    ↪ ordem")[a-z_A-Z][a-zA-Z_0-9]*/
```

Listing 16 Presentation order production rule.

```
order: "\\" ORDER "{" exp "}"
exp: term
    | exp ("," term)
term: factor
    | term ("|" factor)
factor: ID
    | "(" exp ")"
```

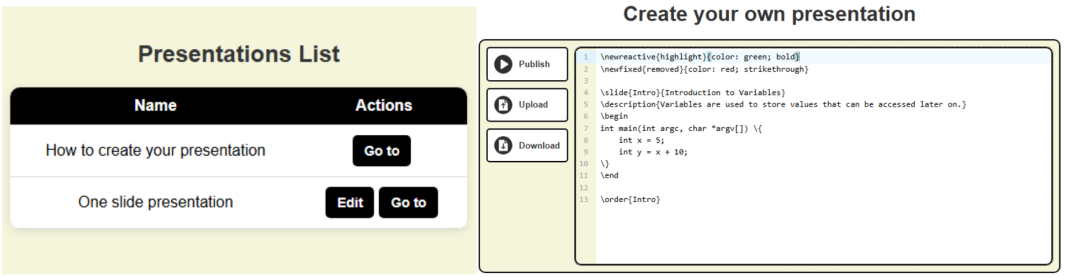


Figure 6 Presentations page (on the left) and Editor page (on the right).

update the learning materials in real time, providing an interactive experience for users. The web application is divided into six pages: Home, About, Documentation, Presentation List, Presentation Editor, and Presentation Navigator, which will be presented in the next section.

5 Stepwise Source, Use and Feedback

Stepwise Source is comprised of different pages that implement several aspects of the application. The main operating pages are the presentations list, editor and navigator.

5.1 Presentations List Page

The presentations page (left side of Figure 6) was designed to showcase all presentations created on the platform. The first row is reserved for the tutorial presentation. Since this tutorial is not intended for user modification, only a "Go To" button is available, preventing any edits. This "Go To" button serves as a redirect, guiding users to the respective presentation page. For all other presentations, besides "Go To", an additional "Edit" button is available, allowing users to modify the presentation content.

5.2 Presentation Editor Page

A large text box serves as the editor, allowing users to directly input the code needed to build their presentations (right side of Figure 6).

An "Upload" button opens a file explorer, enabling the user to select and load a file with pre-existing code. To support users who may wish to save their progress locally, a "Download" button is also available. This function instantly downloads the current content of the text editor as a file named "presentation.txt".

Once the presentation code is finalized, users can click the "Publish" button to publish their presentation. This action opens a panel for naming the presentation, making it easier to find later. The panel prompts for a name and displays any input or naming errors, guiding the user to make necessary corrections before publishing. If the page was accessed to edit an existing presentation, clicking "Publish" will update it instead of creating a new one.

5.3 Presentation Navigator Page

The central feature of Stepwise Source is the Presentation Page, where each slide in a presentation is displayed along with its source code, descriptions, annotations, styling, and unique navigation options. Figures 2 to 5 already showed the presentation navigator in use for the running example of Section 3.

Besides what has already been demonstrated in those figures, the navigation between slides requires more explanation. The navigation section below the content are enables users to move through the presentation slides. The user can click buttons “Previous” on the left and “Next” on the right for simple linear navigation, moving backwards and forwards respectively. In the case of branches (forks) in the presentation flow, more than one option will be available. A graph positioned in the centre of the navigation section shows the entire presentation flow, with each circle representing a slide. While the current slide appears in white with a green border, all other slides are represented in black circles. Users can click on any node to jump directly to that slide, allowing flexible navigation.

5.4 Feedback

To evaluate its effectiveness, a lecture was conducted with a Master’s Degree class in Informatics Education at Minho University, Braga. The session, with the participation of three professors and nine students, covered the need for educational tools that enhance programming instruction, highlighted the specific challenges Stepwise Source aims to address, and included a demonstration of the tool.

Students and teachers explored both perspectives by using the application directly while commenting on its use and usefulness. In addition to pointing out areas where the current version needed improvement, the feedback from this session, more importantly, affirmed Stepwise Source’s value as a relevant contribution for teaching and learning computer programming. Both learners and educators recognized its usefulness, suggesting it would be a meaningful addition to their educational toolkit. This positive reception highlights Stepwise Source’s relevance in supporting computer programming education.

This preliminary assessment session allowed for a confirmation on the positive aspects of Stepwise Source’s usability and user experience.

6 Conclusion

Stepwise Source positions itself as a flexible, useful and free tool to aid educators in creating on-line source code demonstrations that imitate the free-flowing cadence of live lectures. It implements a DSL that supports the creation of slides, styled annotations, and branches in explanation. It was reviewed by future informatics teachers, more specifically, computer programming, that reinforced the notion of its usefulness and applicability.

While a fully functional version of the tool has been developed, several enhancement ideas emerged during the development process and the assessment session that are not yet implemented. One potential improvement is to clarify the relationship between the styled annotation in the source code and its comment, possibly by incorporating note identification numbers. Additionally, offering varying levels of detail in the annotations could further personalize the learning experience, allowing students to engage at their preferred pace. At last, a Context-Free Grammar could be used to validate the learner’s input instead of a regular expression, which would further empower this tool.

Another suggested feature is the integration of a user management system, enabling educators to create dedicated classrooms where they could organize presentations by specific themes. This would enable them to share a curated list of related presentations through a single link, enhancing accessibility and coherence for their students. The last component of future improvement would be the possibility of embedding presentations into other web pages, including Learning Management Systems (LMS) such as Moodle.

Finally, a more robust and structured testing would analyse both the use on the educator's part and also the students' interaction with the tool. This procedure would result in a better overview of Stepwise Source's usability and user experience, but also its value as a computer programming educational tool.

References

1 Cristiana Esteves Araujo. *Training Computational Thinking: exploring approaches supported by Neuroscience*. PhD thesis, University of Minho, March 2022. PDInf - prethesis.

2 Albert Bandura, W. H. Freeman, and Richard Lightsey. Self-efficacy: The exercise of control. *Journal of Cognitive Psychotherapy*, 13(2):158–166, 1999. doi:10.1891/0889-8391.13.2.158.

3 Yorah Bosse, David Redmiles, and Marco A. Gerosa. Pedagogical content for professors of introductory programming courses. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '19, pages 429–435, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3304221.3319776.

4 Chin Soon Cheah. Factors contributing to the difficulties in teaching and learning of computer programming: A literature review. *Contemp. Educ. Technol.*, 12(2):ep272, May 2020.

5 Create React App. Create react app. https://create-react-app.dev/, 2024.

6 Sofia Guilherme Rodrigues dos Santos. Automatic grading of programming exercises. University of Minho, August 2023.

7 Brendan Eich. The javascript programming language. *Netscape Communications*, 1995.

8 Express. Express. https://expressjs.com/, 2024.

9 Anabela Gomes and Antonio Mendes. Learning to program - difficulties and solutions. In *International Conference on Engineering Education – ICEE 2007*, pages 283–287, January 2007.

10 Philip J. Guo. Online python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 579–584, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2445196.2445368.

11 Khan Academy. Khan academy. https://www.khanacademy.org/, 2024.

12 Rodrigo Pessoa Medeiros, Geber Lisboa Ramalho, and Taciana Pontual Falcão. A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education*, 62(2):77–90, 2019. doi:10.1109/TE.2018.2864133.

13 Bertrand Meyer. Codeboard. https://codeboard.io/, 2024.

14 MongoDB. Mongodb. https://www.mongodb.com/, 2024.

15 Python. Python. https://www.python.org/, 2024.

16 Yizhou Qian and James Lehman. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Trans. Comput. Educ.*, 18(1), October 2017. doi:10.1145/3077618.

17 Paula Correia Tavares, Elsa Ferreira Gomes, and Pedro Rangel Henriques. Animation and automatic evaluation in supporting the teaching of programming. In *2015 10th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–4, 2015. doi:10.1109/CISTI.2015.7170548.

Bridging Language Barriers: A Comparative Review and Empirical Evaluation of Source-To-Source Transpilers

André Freitas
ALGORITMI Research Centre / LASI, Dept. of Informatics, University of Minho, Braga, Portugal

Tiago Baptista
ALGORITMI Research Centre / LASI, Dept. of Informatics, University of Minho, Braga, Portugal

Pedro Rangel Henriques
ALGORITMI Research Centre / LASI, Dept. of Informatics, University of Minho, Braga, Portugal

Abstract

Source-to-source transpilation plays a pivotal role in modern software engineering by enabling code migration, feature adoption, and cross-language interoperability without sacrificing semantic integrity. The contributions discussed in this paper can be split into two. The first is a comprehensive literature review that aims at defining what transpilers are, traces their historical evolution from early Fortran/COBOL preprocessors to more recent tools like Babel and TypeScript, and examines key parsing methodologies, AST representations, and transformation strategies. The second is an experimental investigation which assesses several popular transpilers – selected by GitHub popularity and unique language-pair capabilities, when applied to an equivalent code snippet designed to sum even numbers and identify the maximum element. The metrics evaluated were the execution time, CPU, memory consumption, output accuracy and usability.

2012 ACM Subject Classification General and reference → Empirical studies; General and reference → Surveys and overviews; Software and its engineering → Syntax; Software and its engineering → Semantics; Software and its engineering → Translator writing systems and compiler generators; Software and its engineering → Source code generation; Software and its engineering → Parsers; Software and its engineering → Interpreters

Keywords and phrases Source-to-source translation, Code transformation, Parsing, Lexical analysis, Syntax analysis, Semantic analysis, Transpilation

Digital Object Identifier 10.4230/OASIcs.SLATE.2025.11

Supplementary Material *Software (Tools/Scripts)*: `https://justandre02.github.io/Comparative-Review-and-Empirical-Evaluation-about-transpilers/`

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Unit Project Scope UID/00319/Centro ALGORITMI.

1 Introduction

Context and Motivation

As modern software systems grow in size and complexity, developers face ever-increasing challenges in maintaining, migrating and interoperating code across heterogeneous platforms and language versions. Another challenge that companies face is related to the modernization and maintenance of legacy systems that use older languages such as COBOL (according to a survey from Micro Focus from 2022, 800 billion lines of code of COBOL are used in production) since there is a shortage of developers to maintain it and the costs of running them are high due to the need of having specific hardware infrastructure (IBM mainframe for example).

Source-to-source compilers – commonly known as *transpilers* – have emerged as significantly important tools in this context, enabling the automated translation of code from one language or dialect into another while preserving its original semantics and intent.

Objectives

This paper has two main investigation objectives. First, it presents a comprehensive literature review, that traces the conceptual foundations of transpilation: from early **Fortran** and **COBOL** processors in the 1960s, through the advent of AST-driven program transformations, to contemporary tools such as **Babel** and the **TypeScript Compiler**. Second, we complement this theoretical foundation with an independent empirical study that evaluates a curated set of fourteen transpilers – selected by **GitHub** popularity and functional scope – across uniform test cases measuring *execution time, CPU/memory footprint, output precision and usability.*

Document Structure

By joining these two perspectives – the broad bibliographic synthesis and the hands-on performance benchmarks – the research project here reported delivers both a context on the transpilation landscape and concrete guidelines for tool selection in real-world projects. Section 2 details the review methodology and taxonomy of transpilation techniques. Section 3 describes the design of the empirical evaluation, including test configuration and metric definitions. Section 4 presents and analyses the quantitative findings, while Section 5 discusses their implications and trade-offs. The paper concludes in Section 6 with a summary of contributions and possible future work, such as AI-augmented, context-aware code transformations and deeper paradigm-shifting transpilation.

2 Literature Review: Transpilation Techniques and Frameworks

Transpilers: definition and scope

Transpilers – also known as source-to-source compilers – translate code between programming languages or language versions while preserving its original semantics. Contrarily to compilers, rather than emitting machine code, they operate at a higher level through a three-stage process: lexical and syntactic analysis to construct an Abstract Syntax Tree (AST), semantic and syntactic transformations, and generation of equivalent target code [6, 25]. This approach allows developers to adopt new language features yet maintain backward compatibility and interoperability across diverse platforms [11, 19]. Beyond pure translation, transpilers are crucial for modernizing legacy systems by supporting gradual migrations and reducing technical debt [12].

Historical Evolution of Source-to-Source Translation

The roots of source-to-source translation trace back to **Fortran** and **COBOL** preprocessors of the 1960s, which provided macro facilities and dialect conversion to optimise code for different hardware architectures [21, 7]. Early program transformation research introduced AST representations to enhance portability and optimization [14].

During the 1980s and 1990s, the emergence of functional and object-oriented paradigms drove more sophisticated transformation frameworks capable of rewriting code while preserving functional equivalence [4, 10]. Enhanced parsing techniques and pattern-based rewrite algorithms laid the foundation for modern transpilers.

The exponential growth of web applications required tools to reconcile rapidly evolving **ECMAScript** standards with a variety of browsers. Babel[1] became a benchmark by enabling developers to author in **ESNext** and transpile to **ES5**, ensuring compatibility with older environments [1]. Concurrently, **TypeScript** introduced static typing on top of **JavaScript**, further enriching the transpilation ecosystem [20].

2.1 Parsing Methodologies and AST Representations

The parsing phase can be split into lexical analysis (tokenisation) of source text into identifiers, literals and symbols, and syntactic analysis with grammar validation and parse-tree construction [5]. Top-down parsers (like the descent) are straightforward to extend but cannot handle left recursion; bottom-up parsers (LR) support more complex grammars at the expense of implementation effort [8]. Generalised LR (GLR) parsing extends this flexibility, resolving ambiguities and accommodating context-sensitive constructs [15].

A parse tree is distilled into an AST, which removes redundant syntactic details and emphasizes the programme's semantic structure. Key characteristics include a hierarchical node representation, elimination of syntactic sugar, and explicit preservation of control and data flow dependencies which are essential for language-agnostic transformations [13, 10, 9, 23].

2.2 Code Transformation Strategies

Code transformation represents the core computational process of transpilation, where the AST is systematically modified to generate equivalent code in the target language or version [27]. This phase involves multiple strategies to ensure semantic preservation while adapting to the target language's specific syntactic and structural requirements [2].

The transformation process typically encompasses **several key strategies:**

- Structural transformation of language constructs;
- Semantic mapping between different language features;
- Handling of language-specific idioms and patterns;
- Generating optimized and compatible target code.

Complex transformations may involve multiple traversals over the AST, each one addressing different aspects of code translation [27]. These can include:

- Feature adaptation *(e.g., translating modern **JavaScript** features to **ES5**)*;
- Paradigm translation (converting functional programming constructs to imperative styles);
- Semantic equivalence preservation;
- Performance optimization.

2.3 Taxonomy of Transpilers

Reading the available literature, we found that the landscape of transpilation tools can be classified according to the type of translation they perform. They usually are categorised in three classes, as will be described below.

[1] `https://babeljs.io/docs/`

Language-to-Language Transpilers

Language-to-language transpilers represent an approach to cross-language code transformation, enabling developers to translate source code between fundamentally different programming languages. These transpilers address the challenge of linguistic heterogeneity in software ecosystems, facilitating code reuse, platform migration, and technological integration [28].

One of the most popular language-to-language transpilers is **Java2Python**[2] which transpiles from Java to Python. Although a popular tool, it was forgotten over the years due to the fact it only works with Python 2 and was left with no more updates since 9 years ago.

The translation process involves semantic mapping, addressing not just syntactic differences but also paradigmatic variations between source and target languages. For example, transpilers might translate between statically-typed and dynamically-typed languages, or between languages with different memory management approaches, requiring sophisticated computational strategies to preserve the original code's computational intent [22, 16].

Version Transpilers

Version transpilers focus on managing the evolutionary challenges of programming languages by enabling code migration between different versions of the same language. These tools are particularly useful in rapidly evolving language ecosystems where backward compatibility and feature adoption present a significant challenge for developers [3].

Modern version transpilation goes beyond syntax updates, addressing semantic changes, deprecation of language features, and introducing modern language capabilities to legacy codebases. **JavaScript** transpilers like **Babel** exemplify this approach, allowing developers to use next-generation **ECMAScript** features while maintaining compatibility with older browser environments [26].

Paradigm Transpilers

The most complex category, paradigm transpilers translate across programming paradigms: functional, imperative, declarative, demanding profound semantic transformations to preserve computational logic while adapting control flow, data abstractions and side-effect management [1].

It's mentioned as the most complex set of transpilers since the gap between languages of different paradigms is often hard to bridge, requiring first to understand and design how certain statements from a source language, which are not supported or don't exist in the target language, will be mapped into the target language and then apply multiple AST transformations in order to incrementally bridge the gap between both languages. **ClojureScript**[3] is a great example of this type of transpiler because it is a compiler for **Clojure** (dynamic, general-purpose programming language) that targets JavaScript.

[2] `https://github.com/natural/java2python`
[3] `https://clojurescript.org/`

2.4 Software Infrastructure Mapping in Transpilation

Infrastructure Dependencies and Library Ecosystem Challenges

The difficult task of mapping entire software infrastructures is included in the transpilation process, which goes far beyond translating syntactic and semantic code. Standard libraries, runtime environments, package management systems, and platform-specific components are all part of the extensive ecosystems that surround programming languages and make it possible to design and implement applications.

When transpiling real-world applications, developers face the fundamental challenge of relying on substantially different infrastructure foundations. A **Java** application, for instance, depends not only on the **Java** language syntax but also on the extensive **Java Standard Library**, **Java Virtual Machine runtime**, **Maven** or **Gradle** build systems, and Java-specific frameworks. Transpiling such an application to **Python** requires mapping these dependencies to equivalent **Python** infrastructure: the **Python Standard Library**, **CPython interpreter**, pip package manager, and corresponding **Python frameworks**.

Binary Component Integration and Platform Dependencies

When working with dependencies and binary components specific to the platform that cannot be translated immediately through transformation of the source code, the difficulty increases. Database drivers, system APIs, native libraries, and third-party components that are available as built binaries rather than source code are all integrated into many production applications. Pure source-to-source translation is unable to handle the extra layers of complexity created by these dependencies.

Successful application migration in such contexts requires hybrid approaches that combine code transpilation with infrastructure redesign. This process often involves identifying equivalent binary components in the target ecosystem, developing adapter layers to maintain interface compatibility, or replacing entire subsystems with functionally equivalent alternatives that align with the target platform's architectural patterns.

3 Independent Empirical Study: Methodology and Setup

The theoretical research (presented in Section 2) carried out on various transpilers provided valuable information on the characterisation and usage of these tools. However that task paved the way to a more practical and experimental study that could support a deeper analysis of their characteristics. This section describes an experiment aimed at providing concrete results that allow for several relevant conclusions to be identified, highlighting their advantages, limitations and ideal contexts of use. Table 1 exhibits the main observations drawn from the literature review, showing the parameters that better characterise the transpilation tools selected for the empirical study.

3.1 Test Cases and Selection Criteria

The empirical evaluation uses a uniform test suite in which each transpiler is asked to translate a short programme that (a) computes the sum of the even numbers in a given list and (b) finds the maximum value among the list elements. This logic was expressed in the various input languages supported by the tools under test: **JavaScript/TypeScript, Nim, Clojure, Haxe, C** and **Java**. The six code snippets written for each test can be found in

Appendix A. By holding the computational intent constant, we ensure that performance and correctness metrics reflect the capabilities of each transpiler rather than variations in algorithmic complexity.

The **compilers were selected primarily on the basis of the popularity in GitHub**, measured by star count, to capture tools with significant community adoption and support. Additionally, the **Java2Python** converter was included despite its lower visibility, as it fulfilled the specific requirement of translating **Java** code to **Python**. During setup, several candidates could not be installed or executed on our Linux environment due to outdated dependencies or compatibility issues; the final benchmark comprises only the subset of tools successfully integrated into the testbed.

Table 1 Key characteristics of the selected transpilers, with category placed after the tool name.

Tool	Category	Input Languages	Output Languages	AST Handling	Ease of Use	Stability & Known Issues
TypeScript Compiler	Version	TypeScript	JavaScript	Complete	Simple	High stability, few errors
Babel	Version	JavaScript, TypeScript	JavaScript	Complete	Intuitive	High stability, occasional errors
eslint	Version	JavaScript, TypeScript	–	Partial	Moderate	High stability, frequent reports
Nim	Language-to-Language	Nim	C, C++, JavaScript	Complete	Moderate	Stable, actively developed
ClojureScript	Paradigm	Clojure	JavaScript	Complete	Hard	Stable, some reported errors
jscodeshift	Version	JavaScript, TypeScript	JavaScript	Complete	Easy	Moderately stable, occasional errors
ast-grep	Version	JavaScript, TypeScript	–	Simple	Moderate	Moderately stable
Haxe	Language-to-Language	Haxe	JavaScript, Python	Complete	Moderate	Stable, frequent updates
c2rust	Paradigm	C	Rust	Complete	Moderate	Stable, but complex bugs
Fennel	Paradigm	Fennel (Lisp dialect)	Lua	Complete	Simple	Stable, sporadic maintenance
Cito	Language-to-Language	C	JavaScript	Partial	Moderate	Moderately stable
TypeScriptToLua	Language-to-Language	TypeScript	Lua	Complete	Moderate	Stable, minor known bugs
godzilla	Language-to-Language	JavaScript	Lua	Complete	Moderate	Moderately stable
jsweet	Language-to-Language	Java	JavaScript, TypeScript	Complete	Moderate	Moderately stable, some bugs
j2cl	Language-to-Language	Java	JavaScript	Complete	Moderate	Stable, interoperability bugs
Java2Python	Language-to-Language	Java	Python (2.x)	Simple	Easy	Stable, but obsolete (Python 2 only)

Column Definitions

In this table, the **Tool** column lists the name of each transpiler under evaluation for reference. The **Category** column classifies each tool according to its conversion type – *"Language-to-Language"* for those translating between different languages, *"Version"* for those handling differences between versions of the same language, and *"Paradigm"* for those

mapping between distinct programming paradigms. The **Input Languages and Output Languages** columns indicate, respectively, which source languages the tool accepts and which target languages it can generate. The **AST handling** column describes the extent of AST support: *"Complete"* denotes full coverage of nodes and complex transformation passes; *"Partial"* signifies limited support for specific nodes or operations; and *"Simple"* refers to basic functionality for pattern matching or data extraction without deep rewriting capabilities. The Ease of Use column provides a qualitative assessment of the learning curve and interface clarity: *"Simple"* or *"Easy"* indicates straightforward, well-documented APIs requiring minimal configuration; *"Intuitive"* implies a coherent workflow that is naturally accessible despite some complexity; and *"Moderate"* signals that additional configuration steps or intermediate concepts must be mastered before the tool can be utilised to its full potential and *"Hard"* signifies that independent research must be conducted before using the tool. And finally the **Stability & Known Issues** column summarises the general maturity and reliability of each tool, noting whether it is actively maintained, prone to occasional errors or complex bugs, and any recurring issues that users should be aware of.

3.2 Experimental Environment and Tool Installation

All experiments were conducted on a Linux workstation equipped with a multi-core (8 cores) CPU and 32 GB of RAM. Transpilers were installed via their standard package managers or repositories *(e.g. npm for **JavaScript**-based tools, cargo for **Rust**-based tools)*. Where necessary, minor adjustments (such as version pinning or patching build scripts) were applied to resolve compatibility issues. Despite these efforts, some tools remained unusable and were excluded from the study.

Each transpiler was invoked with default settings except where command-line options were required to specify input and output languages. Execution time, CPU utilization and peak memory usage were recorded using system profiling utilities, like the *time* tool integrated in Linux. Output code was then compiled or interpreted to verify correctness, and a precision score from 0 to 100 was assigned based on successful execution and fidelity to the original specification.

4 Results

Table 2 summarizes the four parameters – *execution time, CPU used, memory consumed, and accuracy* – measured to compare the nine Transpilers chosen to conduct the experiment described in Section 3. The following subsections analyse in detail performance, precision, resource consumption, and usability. The last subsection sums up the study conclusions.

4.1 Performance and Efficiency

Execution times varied markedly across tools. **Nim** – transpiling to **C, C++** and **JavaScript** – achieved average runtimes of 0.61 s, 0.59 s and 0.12 s respectively, with moderate memory use (77.8 MB, 79.4 MB, 24.5 MB). In contrast, **ClojureScript** exhibited the slowest performance at 27.44 s despite low memory consumption (36.0 MB), rendering it unsuitable for time-sensitive workflows. The **jscodeshift** transformer delivered the fastest translation (0.06 s) and minimal memory footprint (1.76 MB), although this speed came at the expense of lower precision (see Section 4.2).

Table 2 Performance, resource usage and accuracy results for each transpiler.

Tool	Execution Time (s)	CPU Utilisation (%)	Memory Used (MB)	Accuracy (0-100)
TypeScript Compiler	1.79	86	188.0	80.72
Babel (JS → JS)	8.66	16	92.4	80.72
Babel (TS → JS)	3.29	20	82.4	80.72
Nim (→ C)	0.61	101	77.8	86.25
Nim (→ C++)	0.59	99	79.4	86.25
Nim (→ JS)	0.12	90	24.5	89.50
ClojureScript	27.44	62	36.0	57.70
jscodeshift	0.06	235	1.76	75.00
c2rust	0.16	81	135.3	75.00
Fennel	0.10	98	7.8	79.75
TypeScriptToLua	5.93	59	302.3	72.25
Java2Python	0.09	25	39.9	55.00

4.2 Output Precision

Because the experiment was carried out in multiple tools involving multiple programming languages, it was not possible to use a single application that would evaluate all codes using the same parameters.

So in order to quantify how faithfully each transpiler preserved the semantics of the original programs, we applied a composite scoring methodology on a 0–100 scale. We developed custom evaluation scripts in **Python, JavaScript, Nim,** and other host languages, one per transpiled output, to execute identical input sets and compare results against reference implementations. Although the code under test spanned multiple languages, all source files implement the same algorithmic logic, ensuring a uniform basis for comparison.

The overall precision score $S \in [0, 100]$ for each tool is computed as a weighted sum of five orthogonal categories:

$$S = 40\% \times \mathit{FunctionalCorrectness} + 25\% \times \mathit{SemanticEquivalence} \tag{1}$$

$$+ 15\% \times \mathit{CodeQuality} + 10\% \times \mathit{StructuralSimilarity} + 10\% \times \mathit{ErrorHandling}. \tag{2}$$

Each sub-score is normalized to the range $[0, 100]$ before weighting:

- **Functional Correctness** (40%): Does the transpiled program compile/run and produce the correct outputs for all test inputs? This is the highest-weight category, since a single failure yields a 0 in this dimension.

- **Semantic Equivalence** (25%): Do intermediate states, control flow structure, and side effects match the behaviour of the original program with equivalent inputs?
- **Code Quality** (15%): Does the output follow best practices and idiomatic style for the target language (e.g., Pythonic constructs, proper naming, concise expressions)?
- **Structural Similarity** (10%): Is the high-level structure (functions, classes, loops) aligned with the source?
- **Error Handling** (10%): Are edge cases and exceptions handled appropriately (e.g., null checks, boundary conditions)?

A perfect score (100/100) indicates that the transpiler produced code that compiles/runs without modification, maintains equivalent logic and control flow, adheres to target-language idioms, preserves structural patterns, and robustly handles errors across all test inputs.

Detailed Example: Java2Python (55.00/100)

The **Java2Python** tool translated our sample *'MaxFinder.java'* to *'max_finder.py'* with near-perfect style, structure, and error handling (100% in the last four categories, except the last one that scored 50%), but failed 'Functional Correctness' entirely (0/100 in that category) because of a single method call mismatch:

Original **Java** file:

```
int maxNumber = numbers.get(0);
```

Transpiled **Python** code:

```
maxNumber = numbers.get(0)
```

Since **Python** lists do not implement *'get()'*, this line causes a runtime exception for all test inputs. Correct behaviour requires:

```
maxNumber = numbers[0]
```

As a result, **Java2Python's** functional correctness sub-score is 0, producing a final precision of

$$0.40 \times 0 + 0.25 \times 100 + 0.15 \times 100 + 0.10 \times 100 + 0.10 \times 100 = 55.00.$$

All reported scores reflect the same composite evaluation. Tools scoring below 70-75 may therefore require manual correction or additional validation before deployment.

4.3 Resource Consumption

Memory usage showed significant divergence. **TypeScriptToLua** consumed the most RAM (302.3 MB), followed by the **TypeScript Compiler** (180.2 MB) and **c2rust** (135.3 MB). Conversely, **jscodeshift** (1.76 MB) and **Fennel** (7.8 MB) demonstrated exceptional frugality, making them attractive for resource-constrained environments. CPU utilisation generally remained below 100% of a single core; **jscodeshift** peaked at 235%, reflecting its multi-threaded execution across several cores.

Why in some cases the CPU usage exceeded 100% ?

CPU usage can exceed 100% because modern systems have several CPU cores and the percentage is calculated in relation to the capacity of a single core. When the percentage gets above 100%, it means that the process is using several CPU cores simultaneously.

In the case of **jscodeshift**, CPU usage reaching 235% means that your transformation is effectively using around 2.35 CPU cores on average. This is possible because: **Node.js** (which runs **jscodeshift**) is multi-threaded through its event loop and job threads. The **V8 JavaScript** engine (used by **Node.js**) can parallelise certain operations.

4.4 Usability and Stability

Developer experience was assessed qualitatively. **Babel** and **jscodeshift** stood out for ease of use, due to intuitive command-line interfaces and extensive plugin ecosystems. The **TypeScript Compiler** and **Babel** also showed high stability with minimal runtime errors. **Nim**'s tooling proved reliable but is under active development, which may introduce future breaking changes. Although **Java2Python** delivered perfect accuracy and stability, its support for only **Python 2** renders it impractical for modern codebases.

4.5 Practical Recommendations

Based on these findings, the research reach the conclusion that:

- For **JavaScript/TypeScript** projects: use **Babel** or the **TypeScript Compiler** for a balance of precision, stability and moderate resource demands.
- Where raw performance is critical: employ **Nim** for its rapid and accurate transpilation to **C**, **C++** or **JavaScript**.
- For quick, lightweight transformations: opt for **jscodeshift**, acknowledging potential trade-offs in precision.
- In specialised migrations *(e.g. **C→Rust**, **Fennel→Lua**)*: consider **c2rust** or **Fennel** with the understanding that additional verification may be required.

5 Conclusion

In the context of a research project, an extensive review of the literature was carried out focusing on transpilation: definition, evolution, principles, techniques, and tools. Following that bibliographic study, an empirical study has been conducted to compare some of the most relevant transpilers found and available. In the next paragraphs, the most relevant outcomes are presented.

Insights from the Literature Review

The state of the art analysis confirms that modern transpilers have matured into sophisticated systems capable of transforming code across languages, versions and paradigms while preserving semantic integrity. Early preprocessors evolved into AST-centric frameworks, and tools such as **Babel** and the **TypeScript Compiler** now underpin large-scale web applications by reconciling cutting-edge language features with legacy environments. Key findings include:

- **Semantic Preservation**: Multi-stage AST transformations enable faithful code translation even between drastically different paradigms.

- **Performance Trade-Offs**: Parsing and code-generation introduce overheads that must be balanced against compatibility gains.
- **Tool Ecosystems**: Plugin architectures *(e.g. **Babel** plugins)* and rich analysis APIs foster extensibility and customisation.

Insights from the Empirical Study

The independent benchmarks highlight the practical trade-offs faced when choosing a transpiler:

- **Raw Performance**: **Nim** outperforms all other tools, delivering sub-second translations with perfect accuracy.
- **Balanced Options**: **Babel** and the **TypeScript Compiler** offer strong precision (91%) and stability with moderate resource demands.
- **Lightweight Transforms**: **jscodeshift** achieves near-instantaneous conversions at minimal memory cost, albeit with lower accuracy.
- **Specialised Tools**: Converters such as **c2rust** and **Java2Python** address niche migrations effectively but require careful validation and environment compatibility.

These results underscore the importance of aligning transpiler selection with project priorities – whether that be throughput, fidelity or ecosystem integration.

5.1 Future Work

Building on both theoretical and practical insights, several avenues for further research were identified:

AI-Augmented Transpilation

Integrating machine learning techniques to produce context-aware transformations promises higher fidelity and automatic correction of edge cases [18]. In addition,nual labor rlabord to create a transpiltranspiler is eliminated,nated, from building a parser to a source language to apply AST transformations and code generation for a target language.

Paradigm-Bridging Frameworks

Advancing paradigm transpilers to accurately convert between functional, imperative and declarative models remains an open challenge, with potential in new algorithmic strategies [17].

Distributed and Cloud-Native Scenarios

As software architectures grow more distributed, transpilers must support microservices and server less deployments, optimizing for networked environments and heterogeneous runtimes [24]. In order to, for instance, provide transpilation as a service without requiring a user to have a local system with hardware capabilities to handle the migration of projects with millions of lines of code.

Enhanced Benchmarking

Future empirical studies should incorporate multi-core scaling, plugin overhead, and real-world codebases to refine our understanding of performance versus precision trade-offs.

Toolchain Integration and Automation

Developing unified workflows that integrate transpilers with CI/CD pipelines and automated testing frameworks will streamline large-scale migrations and continuous modernization efforts.

5.2 Additional information

If this article interests you, it is important to mention that there is a complementary website where you can find the article you read, the transpilers (and their version) used along with the links so you can easily access the same tools used in the experiment. After that, the webpage displays all the scripts used to evaluate the transpiled code, so that you can replicate the same experiment and further extend it. Right at the end there is a link to another document that dives into the state-of-the-art study about transpilers.

Here is the link:
`https://justandre02.github.io/Comparative-Review-and-Empirical-Evaluation-about-transpilers/`

References

1 Bastidas F. Andrés and María Pérez. Transpiler-based architecture for multi-platform web applications. In *2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)*, pages 1–6, 2017. doi:10.1109/ETCM.2017.8247456.

2 Thomas Baar and Slaviša Marković. A graphical approach to prove the semantic preservation of uml/ocl refactoring rules. In *Perspectives of Systems Informatics: 6th International Andrei Ershov Memorial Conference, PSI 2006, Novosibirsk, Russia, June 27-30, 2006. Revised Papers 6*, pages 70–83. Springer, 2007.

3 Andrés Bastidas Fuertes, María Pérez, and Jaime Meza. Transpiler-based architecture design model for back-end layers in software development. *Applied Sciences*, 13(20):11371, 2023.

4 Victor Berdonosov and Alena Zhivotova. The evolution of the object-oriented programming languages. *Scholarly Notes of Komsomolsk-na-Amure State Technical University*, 1:35–43, June 2014. doi:10.17084/2014.II-1(18).5.

5 A. Cox and C. Clarke. Syntactic approximation using iterative lexical analysis. In *11th IEEE International Workshop on Program Comprehension, 2003.*, pages 154–163, 2003. doi:10.1109/WPC.2003.1199199.

6 Andrés Bastidas Fuertes, María Pérez, and Jaime Meza Hormaza. Transpilers: A systematic mapping review of their usage in research and industry. *Applied Sciences*, 13(6):3667–3667, 2023. doi:10.3390/app13063667.

7 Nadja Gaudillière-Jami. AD Magazine: Mirroring the Development of the Computational Field in Architecture 1965–2020. In *ACADIA 2020: Distributed Proximities*, volume 1, pages 150–159, Online, France, October 2020. B. Slocum, V. Ago, S. Doyle, A. Marcus, M. Yablonina, M. del Campo. URL: https://hal.science/hal-04588619.

8 Joshua Goodman. Parsing algorithms and metrics. In *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, ACL '96, pages 177–183, USA, 1996. Association for Computational Linguistics. doi:10.3115/981863.981887.

9 Tobias Grosser, Sven Verdoolaege, and Albert Cohen. Polyhedral ast generation is more than scanning polyhedra. *ACM Transactions on Programming Languages and Systems*, 37, July 2015. doi:10.1145/2743016.

10 Dick Grune and Ceriel J. H. Jacobs. *Introduction to Parsing*, pages 61–102. Springer New York, 2008. doi:10.1007/978-0-387-68954-8_3.

11 Evgeniy Ilyushin and Dmitry Namiot. On source-to-source compilers. *International Journal of Open Information Technologies*, 4(5):48–51, 2016.

12 Philip Japikse, Kevin Grossnicklaus, and Ben Dewey. *Introduction to TypeScript*, pages 413–468. Springer, December 2019. doi:10.1007/978-1-4842-5352-6_10.

13 Hui Jiang, Linfeng Song, Yubin Ge, Fandong Meng, Junfeng Yao, and Jinsong Su. An ast structure enhanced decoder for code generation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, PP:1–1, December 2021. doi:10.1109/TASLP.2021.3138717.

14 A. Johnson. Fortran preprocessors. *Computer Physics Communications*, 45:275–281, August 1987. doi:10.1016/0010-4655(87)90164-0.

15 Adrian Johnstone, Elizabeth Scott, and Giorgios Economopoulos. Evaluating glr parsing algorithms. *Science of Computer Programming*, 61:228–244, August 2006. doi:10.1016/j.scico.2006.04.004.

16 Kostadin Kratchanov and Efe Ergün. Language interoperability in control network programming, 2018.

17 Chenyang Lyu, Zefeng Du, Jitao Xu, Yitao Duan, Minghao Wu, Teresa Lynn, Alham Fikri Aji, Derek F Wong, Siyou Liu, and Longyue Wang. A paradigm shift: The future of machine translation lies with large language models. *arXiv preprint arXiv:2305.01181*, 2023.

18 André Melo, Nathan Earnest-Noble, and Francesco Tacchino. Pulse-efficient quantum machine learning. *Quantum*, 7:1130, 2023. doi:10.22331/Q-2023-10-09-1130.

19 Thiago Nicolini, Andre Hora, and Eduardo Figueiredo. On the usage of new javascript features through transpilers: The babel case. *IEEE Software*, pages 1–12, 2023. doi:10.1109/ms.2023.3243858.

20 Thiago Nicolini, Andre Hora, and Eduardo Figueiredo. On the usage of new javascript features through transpilers: The babel case. *IEEE Software*, 41(1):105–112, 2024. doi:10.1109/MS.2023.3243858.

21 Alberto Pettorossi, Maurizio Proietti, Fabio Fioravanti, and Emanuele De Angelis. A historical perspective on program transformation and recent developments (invited contribution). In *Proceedings of the 2024 ACM SIGPLAN International Workshop on Partial Evaluation and Program Manipulation*, PEPM 2024, pages 16–38, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3635800.3637446.

22 David Plaisted. Source-to-source translation and software engineering. *Journal of Software Engineering and Applications*, 06:30–40, January 2013. doi:10.4236/jsea.2013.64A005.

23 Hardik Raina, Aditya Kurele, Nikhil Balotra, and Krishna Asawa. Language agnostic neural machine translation. In *Proceedings of the 2024 Sixteenth International Conference on Contemporary Computing*, IC3-2024, pages 535–539, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3675888.3676109.

24 Burkhard Ringlein, Francois Abel, Alexander Ditter, Beat Weiss, Christoph Hagleitner, and Dietmar Fey. Programming reconfigurable heterogeneous computing clusters using mpi with transpilation. In *2020 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, pages 1–9, November 2020. doi:10.1109/H2RC51942.2020.00006.

25 Larissa Schneider and Dominik Schultes. Evaluating swift-to-kotlin and kotlin-to-swift transpilers. In *Proceedings of the 9th IEEE/ACM International Conference on Mobile Software Engineering and Systems*, MOBILESoft '22, pages 102–106, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3524613.3527811.

26 P Thomas Schoenemann. Syntax as an emergent characteristic of the evolution of semantic complexity. *Minds and Machines*, 9:309–346, 1999. doi:10.1023/A:1008360020568.

27 Patanamon Thongtanunam, Chanathip Pornprasit, and Chakkrit Tantithamthavorn. Autotransform: automated code transformation to support modern code review process. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, pages 237–248, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3510003.3510067.

28 Shanshan Wang and Xiaohui Wang. Cross-language translation and comparative study of english literature using machine translation algorithms. *Journal of Electrical Systems*, 2024. doi:10.52783/jes.3136.

A Code snippets used

To consolidate the experimental study we have conducted, and that was discussed in Sections 3 and 5, Listings 1 to 6, presented below, show the code snippets used for the transpilation across various tools.

Listing 1 JavaScript test case.

```
class Main {
    public static function processNumbers(numbers:Array<Int>):
                                        {sumEven:Int, maxNumber:Int} {
        var sumEven = 0;
        var maxNumber = numbers[0];

        for (num in numbers) {
            if (num % 2 == 0) {
                sumEven += num;
            }
            if (num > maxNumber) {
                maxNumber = num;
            }
        }

        return {sumEven: sumEven, maxNumber: maxNumber};
    }
}
```

Listing 2 Nim test case.

```
proc processNumbers(numbers: seq[int]):
                          tuple[sumEven: int, maxNumber: int] =
    var sumEven = 0
    var maxNumber = numbers[0]

    for num in numbers:
        if num mod 2 == 0:
            sumEven += num
        if num > maxNumber:
            maxNumber = num

    return (sumEven, maxNumber)

let numbers = @[1, 2, 3, 4, 5, 6]
let result = processNumbers(numbers)
```

Listing 3 Clojure test case.

```
(defn process-numbers [numbers]
  (let [sum-even (reduce + (filter even? numbers))
        max-number (reduce max numbers)]
    {:sum-even sum-even
     :max-number max-number}))
```

Listing 4 Haxe test case.

```
class Main {
    public static function processNumbers(numbers:Array<Int>):
                                    {sumEven:Int, maxNumber:Int} {
        var sumEven = 0;
        var maxNumber = numbers[0];

        for (num in numbers) {
            if (num % 2 == 0) {
                sumEven += num;
            }
            if (num > maxNumber) {
                maxNumber = num;
            }
        }

        return {sumEven: sumEven, maxNumber: maxNumber};
    }
}
```

Listing 5 C test case.

```
#include <stdio.h>

typedef struct {
    int sumEven;
    int maxNumber;
} Results;

Results processNumbers(int numbers[], int length) {
    int sumEven = 0;
    int maxNumber = numbers[0];

    for (int i = 0; i < length; i++) {
        if (numbers[i] % 2 == 0) {
            sumEven += numbers[i];
        }
        if (numbers[i] > maxNumber) {
            maxNumber = numbers[i];
        }
    }

    Results results = {sumEven, maxNumber};
    return results;
}
```

Listing 6 Java test case.

```java
import java.util.List;

public class Main {
    public static Result processNumbers(List<Integer> numbers) {
        int sumEven = 0;
        int maxNumber = numbers.get(0);

        for (int num : numbers) {
            if (num % 2 == 0) {
                sumEven += num;
            }
            if (num > maxNumber) {
                maxNumber = num;
            }
        }

        return new Result(sumEven, maxNumber);
    }
}

class Result {
    public int sumEven;
    public int maxNumber;

    public Result(int sumEven, int maxNumber) {
        this.sumEven = sumEven;
        this.maxNumber = maxNumber;
    }
}
```

Portuguese Far-Right Discourse on Social Media: Insights from Topic Modeling

Mauro Cardoso
Instituto Universitário de Lisboa (ISCTE-IUL), Portugal
INESC-ID Lisboa, Portugal

Eugénio Ribeiro
Instituto Universitário de Lisboa (ISCTE-IUL), Portugal
INESC-ID Lisboa, Portugal

Fernando Batista
Instituto Universitário de Lisboa (ISCTE-IUL), Portugal
INESC-ID Lisboa, Portugal

Abstract

This study analyzes the social media discourse of leading figures from Portugal's far right party CHEGA, examining 10,323 posts on X (formerly Twitter) published between late 2019 and mid-2024. Using BERTopic, 59 latent topics clustered into two main discursive dynamics were found: (1) ideological and public, and (2) party, electoral and parliamentary related. Within the first dynamic, we conducted a focused sub-analysis of themes related with identity, immigration and security narratives – topics that display posting peaks around electoral cycles, suggesting the strategic use of emotionally charged, identitarian frames for political mobilization. The model exhibits strong topic coherence and lexical diversity, indicating its robustness in extracting thematic structures from politically polarized microtexts. Nevertheless, our findings are constrained by source, the absence of interaction metrics, and the unmet need to link online discourse to offline events. This study demonstrates how computational topic modeling can reveal strategic communication patterns in far-right political discourse and underscores the need for cross-platform and interaction-level research to assess broader societal impact.

2012 ACM Subject Classification Computing methodologies → Natural language processing; Human-centered computing → Social network analysis; Computing methodologies → Topic modeling; Computing methodologies → Language resources; Applied computing → Sociology

Keywords and phrases Political Discourse, Topic Modeling, Far-Right, CHEGA (Portugal), Social Media

Digital Object Identifier 10.4230/OASIcs.SLATE.2025.12

Funding This work was supported by Portuguese national funds through Fundação para a Ciência e a Tecnologia (FCT) (Reference: UIDB/50021/2020, DOI: 10.54499/UIDB/50021/2020).

1 Introduction

We live in a time when free expression is widely valued, but with that freedom comes a notable increase in polarizing communication practices, particularly on digital platforms. The growth of far-right politics in Portugal reflects a broader trend observed in several European countries in recent years. Factors such as economic crisis, social tensions, and discontent with political elites have created favorable conditions for the emergence of popular disgust and polarized discourse [30, 10].

CHEGA, one of Portugal's most prominent far-right parties, along with its key figures, has capitalized on these tensions, consolidating its presence on the national political scene [14, 21]. The participatory nature of social media reinforces this process, allowing political figures, activists, and supporters to interact, produce content, and share opinions without

the intervention of traditional news outlets [1, 25], often creating echo chambers where their messages are amplified while divergent opinions are marginalized [3]. Through these platforms, far-right actors extend their reach beyond national borders, contributing to a transnational network unified by opposition to globalization and multiculturalism. Digital infrastructures accelerate the circulation of ideologies rooted in fear and resentment, particularly around immigration, national identity, and economic insecurity [16]. Meanwhile, the line between offensive and hateful speech is often unclear, making content moderation and regulation more difficult [5]. Studying these interactions is crucial to understand how individuals and communities navigate discussions on sensitive topics, respond to political and social events, and contribute to the formation of public opinion.

This research aims to explore these online dynamics and their connection to the rise of far-right politics in Portugal, an issue that has raised concerns both nationally and internationally [3]. By analyzing these communication patterns and their underlying structures, this research aims to contribute to a more comprehensive understanding of the digital ecosystem and its impact on contemporary political landscapes. In this context, understanding the topics inherent in this type of data over time is crucial. An established method in Natural Language Processing (NLP) for identifying topics in text is topic modeling. For this research, we used BERTopic [12], a topic-modeling technique that leverages transformer-based sentence embeddings together with a class-based variant of Term Frequency-Inverse Document Frequency (TF-IDF). This robust and flexible approach allows for the capture of semantic and contextual relationships between words, overcoming the limitations of traditional methods such as Latent Dirichlet Allocation (LDA). BERTopic makes it possible to generate more coherent and interpretable thematic representations, as well as supporting dynamic topic modeling, which makes it possible to analyze the temporal evolution of discursive themes in sequential corpora. This functionality is particularly relevant for the study of political discourses and ideological dynamics, as it makes it possible to observe the transformations in topics over time, as demonstrated in the model's applications with Donald Trump's tweets and the transcripts of the United Nations general debates [12]. This research aims to answer the following questions:

- What are the key drivers behind the rise of far-right politics in Portugal and in what ways does the political discourse play a role in shaping and contributing to the growth of these movements?
- What is the role of social media in amplifying the far-right discourse?
- What are the main topics discussed during election periods? How do these topics evolve over time?

This document is organized as follows: Section 2 provides an overview of the related work. Section 3 describes the research methodology, including data collection and processing, as well as model implementation. Section 4 starts with a quantitative evaluation of the models produced and then delves into the analysis of the identified topics and their relationship with the far-right political discourse. Finally, Section 5 summarizes the main findings of the study, discusses its limitations, and provides directions for future research.

2 Related Work

This section introduces the study into the existing literature, beginning by highlighting its social context and then analyzing the most relevant methodological approaches.

2.1 Social Context

The rise of far-right movements in recent decades has generated a substantial interdisciplinary debate. Scholars identify structural and sociocultural drivers, economic pressures, perceived cultural insecurity, and shifting structures of political opportunities while also underscoring how the technical advantages of social media platforms reshape mobilization dynamics. Across national settings, far-right actors rely on a remarkably stable discursive repertoire anchored in nationalism, anti-globalism, anti-elitism, anti-immigration, and cultural nativism. These themes coalesce in narratives that cast society as under attack of external enemies (immigrants or supranational bodies), internal moral decay, and betrayal by domestic elites are presented as simultaneous threats. By positioning themselves as the sole authentic voice of the people, such actors advance xenophobic policy prescriptions while explicitly distancing their programs from fascism and biological racism. Notably, European parties frequently frame the European Union's freedom-of-movement regime as an existential menace, conflating immigration with both economic decline and cultural degradation. Anti-immigration frames therefore remain indispensable for recruiting sectors worried about the growing societal heterogeneity [26].

Until 2019, Portugal stood out in western Europe for the absence of an electorally viable far-right party. The founding of CHEGA ended this phase of exceptionalism, introducing Portuguese politics into the larger European populist wave [20]. The domestic populist field spans multiple dimensions: historical myths, religious invocations, and strategic competition among parties. Media outlets routinely frame populist rhetoric as demagogic or simplistic; strikingly, this label is applied not only to radical actors in CHEGA but also to centrist figures like President Marcelo Rebelo de Sousa whenever emotive, moralizing discourse is employed. Survey data reveal that citizens with populist sympathies resemble habitual abstentionists in institutional distrust, euroscepticism, and perceived identity threat, yet differ markedly in their higher political interest and engagement. These traits strongly predicted support for CHEGA's leader André Ventura in the 2021 presidential election, whereas in the 2022 legislative contest a different determinant, trust in the incumbent government, proved more decisive [20]. Personality analysis also links populist attitudes to conscientiousness, extraversion, neuroticism, and low openness to experience, consistent with a preference for firm immediate solutions in uncertain contexts. Importantly, the Portuguese case shows that populist attitudes can persist without immediate electoral translation, demonstrating the need to consider cultural, institutional, and situational factors together [20].

An online survey carried out between May and June 2021 (n = 3,183) offers the first systematic portrait of CHEGA's militants. Predominantly male, middle-class, educated, and politically active; many without any previous party affiliation, they express great dissatisfaction with democracy, without adopting antidemocratic principles. Instead, they support plebiscitary mechanisms that elevate the will of the people, defending a nativist logic, based on traditional customs, relegating religious identity. Although explicit biological racism is uncommon, essentialist views on culture and ethnicity, especially Roma, are widespread. Aligned with their leader André Ventura, their personalist and punitive populism, characterized by an anti-corruption zeal, calls for strong authority, rejection of multiculturalism, opposition to progressive agendas, emerge as the main drivers of CHEGA's growth, placing the party between protest populism and cultural nationalism [21].

The electoral rise of CHEGA is linked to the confluence of political discourse and weak digital skills among large segments of the Portuguese population. The limited capacity to verify sources exposes citizens to misinformation, while social media algorithms favor sensationalism and polarization of content, reinforcing echo chambers [14]. CHEGA's communication

repertoire: fearmongering, stigmatization, anti-elitism, narrative manipulation, achieves high visibility and engagement online. Comparative parallels with Ventura, Bolsonaro, and Trump underscore a shared playbook: manipulation of electoral narratives, strategic use of Artificial Intelligence (AI) in communication, and selective appropriation of democratic symbols that ultimately erode institutional trust. Therefore, scholars recommend media literacy policies, platform regulation, source accountability, and the reinforcement of professional journalism as countermeasures [14]. The offline ramifications are tangible: the mobilization of the digital platform has preceded street protests and, in some cases, incidents of violence [33, 15]. As social networks consolidate as primary platforms for communication [19], researchers now rely on NLP to classify and interpret large volumes of contentious text.

2.2 Computational Methods for Political Discourse Analysis

Through a systematic review of 64 studies [32] on the application of NLP in the analysis of extremism, the field has been mapped in terms of techniques, tools, datasets, and methodological approaches used to describe and detect extremist discourse. The reviewed literature is organized into five analytical dimensions: research topics, techniques employed, empirical applications, available software tools, and accessible datasets. The theoretical framework carefully distinguishes between extremism and radicalization, defining extremism as an antidemocratic ideological movement that may or may not involve violence, while radicalization is conceptualized as a psychological process of detachment from democratic norms. The operational definition of extremist discourse is articulated through five dominant narrative types: political, historical, sociopsychological, instrumental, and theological-moral. These narratives are often accompanied by discursive elements such as hate speech, war metaphors, and dehumanization strategies.

From a technical point of view, the studies reveal a predominance of classical NLP techniques such as TF-IDF, n-grams, and sentiment analysis with feature extraction techniques for syntactic and semantic analysis frequently include Part-of-Speech Tagging (POS), Named Entity Recognition (NER), and topic modeling with LDA. However, there is a growing adoption of deep learning models, showing particular promise in distinguishing extremist discourse, especially when combined with deep learning algorithms. Among the challenges identified are the limited explainability of deep learning models, the complexity of handling multilingual and code-mixed texts, and the lack of publicly available annotated datasets. However, NLP emerges as a robust and rapidly evolving set of tools to detect and characterize extremist narratives, with significant applications in both academic research and the development of public policies and preventive strategies against online radicalization.

Topic modeling has been instrumental in identifying discourse patterns in large volumes of text and has historically been dominated by methods such as LDA that marked a turning point in probabilistic topic modeling for textual corpora by proposing a generative approach grounded in hierarchical Bayesian principles [2]. However, such approaches are based on bag-of-words representations, which limits their ability to capture semantic and contextual relationships between words. When applied to data from social media platforms, the technique must account for the challenges inherent in short, fragmented, and noisy content. LDA remains one of the most widely used probabilistic models for this task; however, its performance is limited in contexts involving microtexts such as tweets due to the sparsity and lack of contextual co-occurrence [18].

Recent studies comparing LDA, Non-negative Matrix Factorization (NMF), Top2Vec, and BERTopic in 31,800 tweets about travel during the COVID-19 pandemic confirm the well-known obstacles of topic modeling in microtexts: brevity, lexical noise, and heterogeneity. The

methodological pipeline included standard preprocessing steps (punctuation and stopword removal, lemmatization, and normalization). This type of common preprocessing techniques, including others like textual noise reduction (e.g., eliminating hashtags, URLs, and slang), lower-casing, are critical to ensure data quality. Furthermore, term weighting methods such as TF-IDF are often applied to optimize word relevance and topic representativeness [32, 18]. While LDA tends to generate generic and overlapping topics [8, 18], NMF improves thematic coherence but remains limited to TF-IDF representations. Embedding-based models, in particular BERTopic, overcome these barriers by producing semantically more cohesive, stable, and easily visualizable topics between domains [12], offering modern features that make it scalable and flexible for large, short, or polarized corpora, such as social networks and political discourse studies. However, they are limited to assigning one dominant topic per document and can generate many topics and outliers that require manual inspection.

The study closest to this work, both in terms of subject and methodology, applied BERTopic to a corpus of more than 750,000 tweets about the German federal elections of 2021 [13]. The key topics identified included COVID-19, climate policy, tax policy, digitalization, anti-Semitism, gender equality, and the legalization of cannabis. Although some themes, such as the pandemic and digitalization, were common in all subgroups, others, such as humor and cannabis, appeared more prominently in public mentions. Specific global and national events, such as the Taliban's resurgence in August 2021 and debates surrounding Israel, were reflected in temporary spikes in topic-related activity. Sentiment analysis revealed that discussions around the pandemic and immigration were more frequently associated with negative sentiment, especially in tweets from official accounts. Temporal analysis of topic distributions reveals that certain themes gain prominence during specific time periods, reflecting changes in public attention and discourse engagement. These findings suggest that the integration of topic modeling with temporally-aware and linguistically-informed preprocessing pipelines provides a robust foundation for analyzing discursive trends in political social media data. In the same study, it is noted that although BERTopic was found to be effective in capturing latent themes, some limitations were observed, including challenges in interpreting topics with low coherence and the restriction of limiting the generation of topics, which may have excluded less frequent but significant content. The approach proved effective in extracting complex discursive patterns in electoral contexts, demonstrating the usefulness of BERTopic as a tool for the computational investigation of political communication on digital platforms. The integration of multimodal data, temporal analysis, and sentiment analysis is the key to capture the dynamics and diversity of online political discourse [13].

3 Research Methodology

This section outlines the data collection process, the steps taken to prepare the data, and the setup and implementation of the topic modeling pipeline.

3.1 Data Collection

A total of 10,323 unique posts were extracted using the API of the X platform between late 2019 and mid-2024, focusing on posts from prominent figures in the CHEGA Party, including its leader. The selection of these figures was based on their ranking within the party's official list of members, as published on its website[1], as well as their number of

[1] https://partidochega.pt/index.php/orgaos-nacionais

followers, posting activity, and account longevity. Figure 1 shows the distribution of tweets per source. Furthermore, the party's political performance data by district was manually collected from officials at the Portuguese government site for the same period, with 2019 marking the party's foundation. Metrics were compiled from these data, for example, voter totals over time. The data collected was then transformed, mapped, and stored in tabular structures, facilitating analysis and further exploration. Figure 2 illustrates the temporal distribution of the data, per day. Initially, from 2019 to early 2022, there was a relatively low level of activity, with sporadic spikes in the number of posts. From mid-2022, there is a noticeable increase in the volume of tweets. This trend intensifies significantly throughout 2023 and into 2024, with consistent peaks reaching more than 20 tweets per day, coinciding with the party's electoral growth.

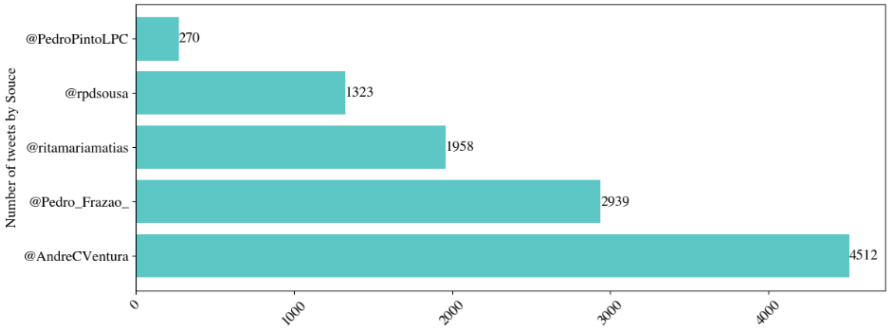


Figure 1 Number of tweets extracted from the account of each of CHEGA's main politicians.

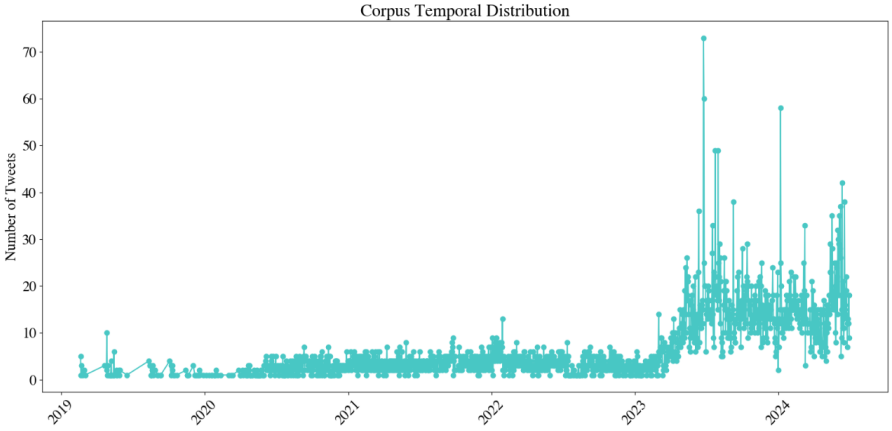


Figure 2 Temporal distribution of the tweets.

3.2 Data Preprocessing

As discussed in Section 2, the preprocessing stage is essential when dealing with social network texts, as informality and network specific mention formats introduce additional problems for analysis and model refinement [17, 32]. The collected texts were subjected to a series of preprocessing steps designed to clean, standardize, and improve their consistency for analysis. Generic mentions of user accounts were removed and/or replaced. The text was then cleaned of URLs, punctuation and special characters, as well as emojis that could

interfere with token recognition. Next, the text was tokenized, simultaneously filtering out stopwords that have been aggregated from different sources, to eliminate terms without semantic relevance. After this process, the empty texts were removed. Finally, duplicates (mainly retweets) were removed to avoid redundancies in subsequent tasks.

To ensure consistency of entity representations, a normalization process based on NER techniques was implemented. It arose from the need to consolidate the variants that would appear in the top words between the topics, following the in-depth experimentation described in Section 3.3. Specifically, all variants of the same person or institution were converted into a common identifier. Table 1 shows some examples of this process.

Table 1 Example of substitutions of singular and pair terms.

Original Term	Substitution
"andréventura" OR "andreventura" OR "venturas" OR "ventura" OR "andrecventura" OR "andré ventura" OR "andre ventura"	andre_ventura
"partidochega" OR "chega" OR "grupoparlamentarchega" OR "partido chega"	partido_chega
"costa" OR "antonio costa" OR "antónio costa"	antonio_costa
"montenegro"	luis_montenegro
"rocha" OR "rui rocha"	rui_rocha
"cotrim" OR "joão cotrim" OR "joao cotrim"	joao_cotrim
"matias" OR "rita matias"	rita_matias

3.3 Model Implementation

To identify emerging topics, a topic modeling pipeline based on the BERTopic model was implemented, adapted to the linguistic and thematic context of the study. A TF-IDF vectorization technique was applied using 5,000 features, designed to reduce dimensionality and mitigate the impact of sparse terms while preserving a rich representation of textual data. The frequency thresholds were empirically defined, with a minimum document frequency of 1% and a maximum of 85%. These thresholds served two purposes: first, to exclude rare and potentially irrelevant terms that appear in only a small number of documents, thus reducing noise and improving topic stability; and second, to remove overly frequent terms (similar to contextual stopwords) that do not contribute meaningfully to semantic differentiation across documents. This process ensured that the resulting feature space emphasized terms with greater topical relevance and discriminative value.

A customized list of stopwords was incorporated to remove common and politically irrelevant tokens, including elements specific to the social media platform, such as "rt" and "id". These tokens were identified through multiple iterations of the model output. The pipeline was applied to both the preprocessed data and the raw data.

Text embeddings were generated using the multilingual model Language-agnostic BERT Sentence Embedding (LaBSE) [9], which was trained to capture semantic similarity between languages. We opted for this model as, in preliminary experiments, it led to the generation of more consistent and interpretable topics than the multilingual Sentence BERT models [29] and the Portuguese-specific models of the Serafim family [11].

Dimensionality reduction was performed using Uniform Manifold Approximation and Projection (UMAP) with cosine distance and custom configurations for `random_state=30`, `n_neighbors=5`, and `min_dist=0.03`, ensuring replicability [24]. This step aimed to improve the separation of latent topics in the embedded space. Clustering was performed using

the Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) algorithm, configured for increased sensitivity to local point density (`min_cluster_size=50` and `min_samples=3`) [4, 23, 22]. In a context where data density varies, the adjustment and definition of these parameters were experimental to balance granularity and thematic sensitivity. This enabled for more sparse clusters and significant semantic heterogeneity between documents.

3.4 Evaluation Methodology

To perform a solid analysis based on the topics identified by a topic model, it is important to assess the intrinsic quality of the model. For this purpose, we rely on two metrics. Semantic topic coherence is a fundamental step in validating topic models, particularly when the goal is to ensure that the generated topics are interpretable by humans. One of the most widely used metrics for this purpose is c_v coherence, proposed by [31], which measures how well the words of a topic fit semantically by comparing their large-scale co-occurrence patterns with a vector metric, offering an efficient compromise between performance and interpretability. It has proven to be robust even when working with short texts or when topics are generated using modern approaches such as BERTopic. Topic models with higher coherence scores tend to be more semantically interpretable and meaningful. Beyond semantic coherence, topic diversity is another crucial dimension for evaluating topic model quality. Topic diversity quantifies the lexical overlap between topics and is defined as the proportion of unique words across the top-n terms from all generated topics [7, 6]. Values near zero indicate significant topic redundancy, while values approaching one suggest a greater variety and distinction among topics.

4 Result Analysis

This section starts with the quantitative evaluation of the topic model that serves as the basis for our analysis. Then, a combined quantitative and qualitative discussion of the topics is presented, covering their semantic consistency, thematic grouping, and relationships with each other. In line with the related work in Section 2, the findings are expected to revolve predominantly around key themes such as immigration, national identity, corruption, and public safety. These themes are likely to be articulated in a personalized and emotionally charged manner, consistent with populist frames that dramatize a moral divide between virtuous citizens and a corrupt elite. The repeated emphasis on safeguarding order, combating political decadence, and defending nativist values supports a discursive construction of internal or external enemies, thus legitimizing exceptional political interventions centered on strong leadership.

4.1 Model Evaluation

The model identified 59 distinct topics, including the outlier class (topic -1). No application was used to avoid outliers, in order to maintain a more truthful nature of the data [6]. The topics achieved an average coherence score of 0.55, with values ranging from a minimum of 0.29 to a maximum of 0.73. These results indicate a relatively balanced distribution, with most topics displaying satisfactory levels of semantic coherence, particularly considering the nature of the collected data. Regarding topic diversity, the results indicate a high level of lexical diversity between topics, with a maximum value of 0.96 when considering the top 10 terms in each topic. Even with an increase in the number of words analyzed per topic,

diversity remains high, 0.92 (top-20) and 0.88 (top-30), showing that the repetition of terms between different topics is low. The slight decrease in diversity as the number of terms evaluated increases is expected, given the increased likelihood of lexical overlap. However, the value of 0.81 for the top-50 terms continues to indicate a good thematic separation between topics, reinforcing the robustness of the model in capturing the distinct dimensions of the discourse.

4.2 Topic Analysis

A short description was attributed to each topic by providing its top 50 keywords and the domain context to the GPT-4o Large Language Model (LLM) [28, 27]. The generated descriptions were then manually curated. Considering the hierarchical nature of the topics generated by BERTopic, this process also enabled the preliminary grouping of topics into thematic categories, as shown in Table 2 and Figure 3.

Together, the groups map ten broad domains of the party's discourse: institutional politics and electoral strategy; corruption and scandals; ideological conflict and online polarization; national identity and historical memory; European and transnational far-right coordination; social tensions around migration, crime and minority rights; religion and moral issues; economic hardship and public health crises; media and communication dynamics; and a residual class that gathers low-frequency or heterogeneous items.

Figure 3 represents the thematic hierarchy map, together with the corresponding topic identifiers (T) and the number of associated documents (D). The groups were supported and explored with hierarchical clustering (as provided by BERTopic). This structure supported the assignment of broader interpretative meanings, resulting in a hierarchically organized thematic framework. Three overarching domains emerged: (0) Other or outliers (2,382; 23%), being a residual class; (1) Ideological and public dynamics (2 981; 29%), divided between security-identity discourses (1 017 documents) and moral-religious discourses (797), accompanied by 1 037 colloquial digital speeches that amplify these frames; and (2) Party, electoral, and parliamentary dynamics (4,960; 48%), covered by 1,094 texts on public policies, leadership and economy, 1,007 on territorial mobilization, and above all, 2,859 disputative-style speeches combining accusations of corruption with campaign communication, highlighting the party's central media strategy. Taken together, the hierarchy exposes three interdependent rhetorical pillars: an identitarian securitarianism based on the "us" versus "them" dichotomy, a countercultural moralism oriented toward defending traditional values, and a dramatized antisystem denunciation, disseminated through strong media and digital presence. The uneven distribution of the documents, with almost half dedicated to political confrontation and around a third to the identity-moral axis, suggests that the visibility gained in parliamentary debates and in the media serves as a vehicle for reinforcing emotive messages of security, nationalism, and moral order. This empirical framework thus offers a robust starting point for temporal analyses of the party's thematic evolution and for checking how peaks in public attention correlate with the strategic use of certain frames.

Figure 4 shows the aggregate monthly evolution of the three main thematic dynamics mentioned previously, from late 2019 to mid-2024, highlighting the relevant electoral moments marked in red. The graph shows not only the volume of monthly mentions within each category, but also allows us to observe patterns of discursive intensification associated with key moments in the national political calendar, such as legislative and presidential elections. The overlap of curves and markers indicates a possible correlation between the increase in discursive activity on X (formerly Twitter) and electoral cycles, reflecting both the growth of the party's support base and its strategic positioning on social networks over time.

1. Others: – **T:** -1 **D:** 2382
2. Ideological and Public Dynamics
   - 2.1 Public Order, Clash of Values, Religious and Identitarian Issues
     - 2.1.1 Conservative Approach, Public Order, Nationalism/Patriotism and Traditional Values
       - 2.1.1.1 Social, Political Accountability and Dissatisfaction – **T:** 45, 47 **D:** 130
       - 2.1.1.2 National Identity and Tradition, Ethnic Issues, Immigration and Security – **T:** 7, 12, 14, 21, 39, 41, 43, 48 **D:** 1017
     - 2.1.2 Ideological Positions, Moral, Sexual and Religious Debates
       - 2.1.2.1 Sexual Crimes, Feminism/Abortion, Religious Education and Gender/Sexuality Debates – **T:** 24, 27, 44, 46 **D:** 357
       - 2.1.2.2 Religious Conflicts, Political Ideologies, Extreme Disputes, Relations between Belief, Morals and Politics – **T:** 8, 32, 34, 50 **D:** 440
   - 2.2 Discourse of Social and Digital Interaction, Internet Colloquial Tone – **T:** 5, 6, 29, 35, 37, 38, 40, 57 **D:** 1037
3. Party, Electoral and Parliamentary Dynamics
   - 3.1 Governance, Economy and Social Policies, Public Health
     - 3.1.1 Public Management, Health, Investigations/Scandals and Power Conflicts – **T:** 22, 30, 31, 33 **D:** 390
     - 3.1.2 Socio-Economic Criticism and Debate – **T:** 11, 16, 23, 51 **D:** 539
     - 3.1.3 Controversies and Disputes about Political Leadership and Government Management – **T:** 53, 56, 58 **D:** 165
   - 3.2 Political-Electoral Dynamics, Public Management, Ideological Debates and Media Coverage
     - 3.2.1 Political Dispute, Ideological Positions, Communication and Media – **T:** 0, 1, 2, 3, 4, 10, 15, 18, 25, 26, 36, 42, 52, 54, 55 **D:** 2859
     - 3.2.2 Political-Parliamentary and Electoral Cycle, Mobilization, Interaction and Event Organization – **T:** 9, 13, 17, 19, 20, 28, 49 **D:** 1007

Figure 3 Thematic hierarchy map.

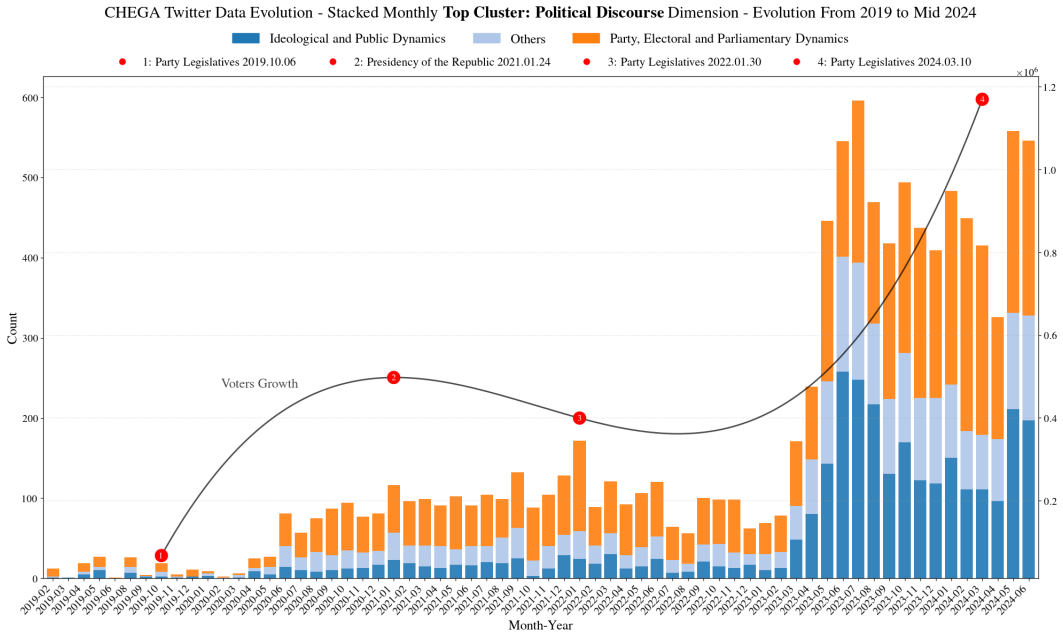


Figure 4 Monthly Evolution of Political Discourse on X (formerly Twitter) associated with CHEGA (2019-2024).

Table 2 Topic-model labels assigned to the 59 BERTopic clusters.

ID	Definition
-1	Others
0	Communication, the Press and Political Mobilization
1	Political Vandalism and Ideological Conflict
2	Corruption and Abuse of Power
3	Political Coalitions, Controversies and Political Responsibility
4	Media Events and Political Events and Campaigning
5	Informal Expressions and Personal Interaction
6	Online Debates and Interpersonal Criticism
7	Social Tensions and Immigration
8	Indoctrination and Ideological Polarization
9	Parliamentary Activities and Constitutional Review
10	Elections and Political Strategy
11	Tax Burden and Social Costs
12	Crime and Public Safety
13	Political Events and Campaigning
14	Immigration and Border Control
15	European Right and Political Coordination
16	Economic and Social Crisis
17	National Identity and Patriotism, Youth and Education
18	Political and Institutional Conflicts
19	Agriculture, Fisheries and Rural Development
20	Local and Regional Elections
21	Family, Faith, Celebrations and Religion
22	Crisis and Collapse in Health
23	Corruption, Public Management and Clientelism
24	Criminal Justice, Violence and Crime
25	Alternative, Criticism and Political Confidence
26	European Right and National Identity
27	Gender Equality and Feminist Dynamics
28	Elections, Regional Campaigns and International Far-Right Movements
29	Insults and Ideological Conflicts, Disinformation and Public Exposure

ID	Definition
30	Diplomacy, Budget and Government Relations
31	Governance, Controversies and Political Responsibility
32	Patriotism, Controversy and Mobilization
33	Opposition, Criticism and Political Management
34	Ideologies and Individual Freedoms
35	Social Interactions, Congratulations and Emotions
36	Controversies and Political Scandals
37	Expressions, Rhetoric and Emotions
38	Racism, Public Opinion and Prejudice
39	Public Safety, Crime and Minorities
40	Disinformation, Defamation and Social Networks
41	Colonialism, Historical Memory and Reparations
42	European Politics, Territorial Action and Institutional Participation
43	Patriotism, National Identity and Historical Memory
44	Indoctrination, Religion and Moral Education
45	Protest, Indignation and Political Outcry
46	Sexuality, Ideology and Child Protection
47	Corruption, Impunity and Criticism of the Political System
48	National Honor, State Careers and Patriotic Duty
49	Public Space, Mobility and Local Identity
50	Religious Conflicts and Terrorism
51	Justice, Cost of Living and Economy, Social Exclusion
52	Elections and Political Participation
53	Government Crisis and Public Administration
54	Political Participation and Institutional Action
55	Political Mobilization and Campaign Rhetoric
56	Public Health and the Hospital System
57	Media, Journalism and Television Representation
58	Participation and Political Figures

For analytical purposes, we have selected the category “national identity and tradition, ethnic issues, immigration, and security”, which is a predominant area of focus within this study. It comprises 1,017 documents, representing approximately 10% of the dataset, and belongs to the category of ideological and public dynamics. Table 3 shows eight topics from the previously selected category. Each topic is associated with a meaningful label and is represented by the top-50 most relevant keywords of the topic.

The three topics directly related to immigration and public order (topics 7, 12 and 14) share the words “threat”, “violence”, and “lack of control”, which reinforces the idea that internal security and border management merge into a single narrative about external risk. In contrast, topic 21 (“Family, Faith, Celebrations, and Religion”) shifts the focus from security anxiety to a register of values and affective belonging. Words like “brotherhood’, “Fátima”, “fatherhood”, and “happyfamily” refer to Catholic festivities, family cohesion, and a positive imaginary community. This vocabulary acts as a discursive counterbalance: Where the migratory topics point to a threat, this one evokes the protection of traditions. Topics 39, 41, 43 and 48 fall between these two poles. Topic 39 takes up the “crime” but adds ethnic markers (“gypsies”, “negro”), indicating that public security is also framed by specific minority categories. Topics 41 and 43 invoke the historical past (“carnation”, “Salazar”,

"Camões") to legitimize a selective memory of the nation, sometimes celebratory ("heroes", "homeland", "pride") and sometimes resentful ("reparations", "traitors"). Finally, topic 48 converges on the valorization of state careers ("military", "firefighters", "police officers") and the "homeland" as a duty, connecting national honor with corporate recognition.

Table 3 Selected topics, manually categorized, representing the national identity, tradition, ethnic issues, immigration, and security.

ID	Topic Definition	Top 50 Keywords
7	Social Tensions and Immigration	veremos, confiar, martim, moniz, oferecer, envergonham, atlântico, ex, seleção, limpo, limpeza, nepalês, louvor, paris, lembrome, policial, juntar, acordar, suíça, convosco, terminamos, amadora, ficaria, contamos, prontos, resistir, podridão, traficantes, mostram, hino, comunidade, destino, revolta, segredo, falava, ferro, sai, bocado, indivíduos, merecem, fraudes, frança, anedota, ocasião, abortar, marine, básicos, pais, islamização, pedras, rainha, símbolo
12	Crime and Public Safety	terror, faca, assaltos, adulto, tiroteio, esfaqueou, multiculturalismo, imigrante, moradores, indivíduo, islâmico, permitimos, incêndio, chamas, rixa, afegão, provocam, agressões, feridos, policial, insegurança, destruídos, armado, incendiários, agredir, orações, ourém, pornografia, plena, drogas, consumo, atirados, martim, moniz, terrorista, fumar, acontecem, ferida, islamização, metro, homicídio, brutalmente, violentos, cenário, agressor, bárbara, agente, infantil, ignoram, droga, psp
14	Immigration and Border Control	lampedusa, ilegais, deportação, aima, imigrantes, integração, imigrante, deportar, migração, massiva, tendas, quotas, refugiados, agência, descontrolados, espadas, asilo, facas, desembarque, refugiado, gratuito, migrações, migrantes, habitantes, apresentarem, dupla, entram, seguras, pacto, indiano, legalização, pedidos, controlada, utilizam, descontrolada, descontrolo, afegãos, idade, regras, chegada, fronteiras, fiscalização, zonas, alterar, chegam, ilegal, nacionalidade, chegaram, recorde, encontram
21	Family, Faith, Celebrations and Religion	brotherhood, tabuleiros, familyparty, happyfamily, fatherhood, familianumerosa, brothers, amam, catedral, parentalidade, family, santo, califórnia, festas, ventre, toiro, espírito, áfrica, irmãos, bispo, salomé, espada, mães, alegria, jesus, magnífica, colegas, natal, jerusalém, estátua, humildes, fátima, companheiros, international, nascimento, santidade, desempenha, paixão, reta, partidária, santuário, activista, dedicação, henriques, palestra, muçulmanos, emocional, católico, empresarial, cartaxo
39	Public Safety, Crime and Minorities	ciganos, magistrados, policiais, criminalidade, elogiar, gnr, seguro, monsaraz, negro, cigano, estatísticas, agir, probabilidade, crimes, violentos, reguengos, polícia, polícias, ligações, criminosos, bandidos, autoridade, publicações, luxo, agressores, agressões, mortos, diariamente, resistiremos, delinquentes, referente, trata, nacionalidade, acontecem, seixal, serem, culpas, coitadinho, homicídio, cigana, colonial, escravatura, prisionais, morto, erro, bandido, nulo, forças, arriscar, perderem, meios
41	Colonialism, Historical Memory and Reparations	cartoon, excolónias, orgulha, rejeitar, traidores, lgbti, desculpa, domínio, mansão, territórios, obscuros, humilhar, vergonhosa, antigas, degradação, antepassados, indemnizações, armadas, impostos, moçambique, desrespeito, forças, governa, pagar, promove, contentes, manter, colapso, corruptos, segurança, tratados, ações, pedimos, protege, pediu, ana_gomes, agem, insultar, compensações, online, cultural, colonial, paga, ilegal, imóveis, sintra, garantir, humanos, prec, polícia, fatura
43	Patriotism, National Identity and Historical Memory	milenar, históricos, lutam, tradições, agente, cravos, camões, dou, colectiva, antepassados, carneiro, perdida, amamos, psp, necessárias, desculpa, brutalmente, sá, portugalidade, símbolos, comunidades, palhaçada, lutaram, identidade, medalha, agredido, espírito, feitos, pátria, orgulho, portuguesas, heróis, hino, fantasma, espadas, protege, serviu, londres, veste, continuarmos, endémica, metro, racismo, independência, escumalha, fábio, deixarmos, bonitos, pergunta, salazar
48	National Honor, State Careers and Patriotic Duty	feriado, reivindicações, queridos, divulgação, policiais, militares, justa, inspiração, novembro, homenagem, entidades, exército, militar, agradecimento, prometido, heróis, sacrifício, protesto, lutaram, carreiras, combatentes, forças, lealdade, data, altas, bombeiros, antigos, empenhados, segurança, corpo, isabel, celebrar, esquecidos, básicos, ideologias, totalitarismo, membro, rainha, prisionais, castelo, seguido, delito, particularmente, operacional, proteção, humana, pátria, armadas, testemunhar, patriotismo

Figure 5 shows the monthly evolution of topics included in the dimension of national identity, tradition, ethnic issues, immigration, and security in the political discourse associated with CHEGA on X. The distribution shows a progressive increase in the incidence of topics related to immigration, public safety, crime, nationalism, and social tensions. The graph shows not only the volume of monthly mentions within each category, but also allows us to observe patterns of discursive intensification associated with key moments in the national political calendar, such as legislative and presidential elections. The overlap of curves and markers indicates a possible correlation between the increase in discursive activity on X and electoral cycles, reflecting both the growth of the support base of the party and its strategic positioning on social networks over time. The pattern of discursive activity remains consistent with that shown in Figure 4. The relationship between the curve and the electoral

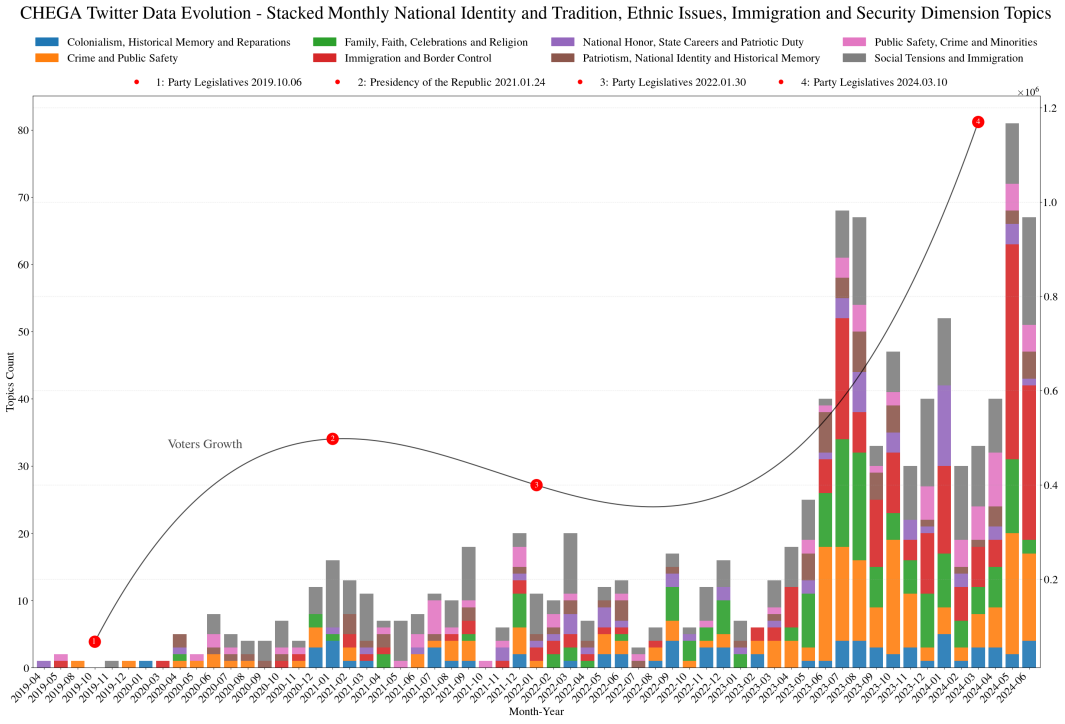


Figure 5 Monthly Evolution of Topics Related to National Identity, Tradition, Ethnic Issues, Immigration, and Security Dimensions (2019-2024).

cycles is also maintained. This growth becomes particularly expressive throughout 2023 and into the early months of 2024, culminating in a sharp discursive peak just before and after the legislative elections in March 2024. This is reflected in the growth in voting numbers, with the range of figures in the 2022 parliamentary elections being around 400,000, rising to more than a million in 2024.

Among the most recurrent subthemes are “social tensions and immigration”, “crime and public safety”, “immigration and border control”, and “family, faith, celebrations, and religion”. In particular, social tensions and immigration appear more consistently throughout the timeline, suggesting its centrality in the long-term discursive strategy of the party. Collectively, these themes account for a substantial portion of the discursive volume within this dimension, indicating a communication strategy grounded in the mobilization of negative affect, particularly fear, insecurity, and perceived threat, often linked to the presence of minorities or immigrants, and emotionally reinforced through references to family, faith, and religious values. Further strengthening this divisive narrative, topics such as “colonialism, historical memory, and reparations”, “patriotism, national identity, and historical memory”, and “national honor, state careers, and patriotic duty” also appear consistently.

These topics suggest a symbolic articulation between security, identity, and morality, constructing a vision of national cohesion under threat. The temporal overlap between the rise in discursive activity and the electoral cycles indicates an instrumental use of these narratives for political mobilization. The overlaid curve, representing the electoral growth of the party, appears to align with the intensification of the security and identity-oriented rhetoric, strengthening the hypothesis that these themes play a strategic role in consolidating and expanding the voter base of the party.

5 Conclusions and Future Work

This study analyzed 10,323 unique posts on X (formerly Twitter) published by key figures in CHEGA between late 2019 and mid-2024, using BERTopic to uncover 59 latent topics in the political discourse of the party. The evaluation of the model revealed an average topic coherence of 0.55 and a topic diversity exceeding 0.80 of the top 50 terms, validating the robustness of the approach. In terms of empirical contributions, hierarchical aggregation enabled the identification of two major discursive dynamics: (1) ideological and public, and (2) political, electoral, and parliamentary. Within the first dynamic, the national identity, tradition, immigration and security subcategory, which represents approximately 10% of the corpus, followed the general pattern of discursive growth observed across the dataset, particularly during electoral periods, culminating in between early 2023 and mid 2024. Although this trend reflects the political nature of the corpus as a whole, the prominence of this subcategory suggests a communication strategy that takes advantage of negative emotional appeals (fear, insecurity), symbolically reinforced through identity and moral narratives, to amplify the political message.

The combination of LaBSE embeddings, UMAP-based dimensionality reduction, and HDBSCAN clustering, together with a recent instruction-tuned LLM (GPT-4o [28, 27]) and manual curation, proved effective in analyzing multilingual and politically polarized corpora. The convergence between peaks in security and identity-related discourse and electoral cycles underscores the need for media literacy policies and content moderation strategies that discourage the dissemination of messages exploiting fear or aversion to the *other*.

However, it is important to acknowledge the potential biases in this study. On the one hand, relying on a single data source introduces selection bias: the focus on the accounts of CHEGA key figures excludes interactions with supporters and counter-discourses. Additionally, the use of monthly snapshots limits the temporal resolution of the analysis, preventing a more fine-grained estimation of causal relationships between offline events and topic fluctuations. On the other hand, although we attempted to minimize human-introduced bias in the topic generation process (e.g. by supporting decisions with quantitative metrics such as c_v coherence and lexical diversity, and by using an LLM to generate the short descriptions of the topics) the manual curation and interpretation of topics remains subjective and susceptible to bias.

Future work should focus on expanding the sample and triangulating across platforms by incorporating data from additional social networks to assess narrative consistency and detect cross-platform variation. Including replies, for instance, would help map the interactional ecology and organic reach of the messages. Estimating causal links between discursive peaks and real-world events remains a critical objective for understanding the dynamics at play.

In the BERTopic results derived from hierarchical clustering, a notably high proportion of cases (23%) were considered outliers (represented by "Others"). While this category is often overlooked, its significant size warrants further analysis and interpretation.

Finally, developing an interactive and comprehensive artifact that includes the full topic analysis can increase the study's accessibility and public relevance. It can also support more detailed classifications, such as identifying emotional tones or rhetorical strategies within thematic dimensions and associated topics.

References

1 Rémi Almodt. The Right-Wing Perspective: Populist Frames and Agenda on Facebook in Central and Eastern Europe. *Central European Journal of Communication (CEJC)*, 15(3(32)):434–463, 2023. doi:10.51480/1899-5101.15.3(32).6.

2 David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research (JMLR)*, 3:993–1022, 2003. doi:10.5555/944919.944937.

3 Manuela Caiani and Patricia Kröll. The Transnationalization of the Extreme Right and the Use of the Internet. *International Journal of Comparative and Applied Criminal Justice*, 39(4):331–351, 2014. doi:10.1080/01924036.2014.973050.

4 Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-Based Clustering Based on Hierarchical Density Estimates. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 160–172, 2013. doi:10.1007/978-3-642-37456-2_14.

5 Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. Automated Hate Speech Detection and the Problem of Offensive Language. In *Proceedings of the International AAAI Conference on Web and Social Media (ICWSM)*, pages 512–515, 2017. doi:10.1609/icwsm.v11i1.14955.

6 Muriël de Groot, Mohammad Aliannejadi, and Marcel R. Haas. Experiments on Generalizability of BERTopic on Multi-Domain Short Text. In *Widening Natural Language Processing (WiNLP)*, 2022. doi:10.48550/arXiv.2212.08459.

7 Adji B. Dieng, Francisco J. R. Ruiz, and David M. Blei. Topic Modeling in Embedding Spaces. *Transactions of the Association for Computational Linguistics*, 8:439–453, 2020. doi:10.1162/tacl_a_00325.

8 Roman Egger and Joanne Yu. A Topic Modeling Comparison Between LDA, NMF, Top2Vec, and BERTopic to Demystify Twitter Posts. *Frontiers in Sociology*, 7:886498, 2022. doi:10.3389/fsoc.2022.886498.

9 Fangxiaoyu Feng, Yinfei Yang, Daniel Cer, Naveen Arivazhagan, and Wei Wang. Language-agnostic BERT Sentence Embedding. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, volume 1: Long Papers, pages 878–891, 2022. doi:10.18653/v1/2022.acl-long.62.

10 Matt Golder. Explaining Variation in the Success of Extreme Right Parties in Western Europe. *Comparative Political Studies*, 36(4):432–466, 2003. doi:10.1177/0010414003251176.

11 Luís Gomes, António Branco, João Silva, João Rodrigues, and Rodrigo Santos. Open Sentence Embeddings for Portuguese with the Serafim PT* Encoders Family. In *Proceedings of the EPIA Conference on Artificial Intelligence*, pages 267–279, 2024. doi:10.1007/978-3-031-73503-5_22.

12 Maarten Grootendorst. BERTopic: Neural Topic Modeling with a Class-based TF-IDF Procedure. *Computing Research Repository*, arXiv:2203.05794, 2022. doi:10.48550/arXiv.2203.05794.

13 Nils Constantin Hellwig, Jakob Fehle, Markus Bink, Thomas Schmidt, and Christian Wolff. Exploring Twitter Discourse with BERTopic: Topic Modeling of Tweets Related to the Major German Parties during the 2021 German Federal Election. *International Journal of Speech Technology*, 27:901–921, 2024. doi:10.1007/s10772-024-10142-4.

14 Joana Jerónimo. Comunicação na Era da Desinformação: o Crescimento da Extrema-Direita e a Iliteracia Digital. *The Trends Hub*, 1(4), 2024. doi:10.34630/tth.vi4.5683.

15 Ofra Klein and Jasper Muis. Online Discontent: Comparing Western European Far-Right Groups on Facebook. *European Societies*, 21(4):540–562, 2019. doi:10.1080/14616696.2018.1494293.

16 Tiago Lapa and Branco di Fátima. Hate Speech Among Security Forces in Portugal. *Communication Library*, 7:277–293, 2023. doi:10.25768/654-916-9.

17 Xiaohua Liu, Ming Zhou, Xiangyang Zhou, Zhongyang Fu, and Furu Wei. Joint Inference of Named Entity Recognition and Normalization for Tweets. In *Proceedings of the Annual*

Meeting of the Association for Computational Linguistics (ACL), volume 1: Long Papers, pages 526–535, 2012. URL: https://aclanthology.org/P12-1055/.

18 Diana Lopes-Teixeira, Fernando Batista, and Ricardo Ribeiro. Discovering Trends in Brand Interest through Topic Models. In *Proceedings of the International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)*, pages 245–252, 2018. doi:10.5220/0006936202450252.

19 Adrian Lüders, Alejandro Dinkelberg, and Michael Quayle. Becoming "us" in Digital Spaces: How Online Users Creatively and Strategically Exploit Social Media Affordances to Build Up Social Identity. *Acta Psychologica*, 228, 2022. doi:10.1016/j.actpsy.2022.103643.

20 Luca Manucci. Portuguese Populism: People, Parties, and Politics. *Análise Social*, 59(251), 2024. doi:10.31447/202200.

21 Riccardo Marchi and José Pedro Zúquete. Far Right Populism in Portugal: The Political Culture of Chega's Members. *Análise Social*, 59(251), 2024. doi:10.31447/2022116.

22 Leland McInnes and John Healy. Accelerated Hierarchical Density Clustering. In *Proceedings of the IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 33–42, 2017. doi:10.1109/ICDMW.2017.12.

23 Leland McInnes, John Healy, and Steve Astels. HDBSCAN: Hierarchical Density Based Clustering. *The Journal of Open Source Software*, 2(11):205, 2017. doi:10.21105/joss.00205.

24 Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *Computing Research Repository*, arXiv:1802.03426, 2018. doi:10.48550/arXiv.1802.03426.

25 George Newth. 'Talking About' the Far Right and Common Sense. A Case Study of Matteo Salvini's Buon Senso Trope on Twitter (2018–2023). *Acta Politica*, 60:361–384, 2025. doi:10.1057/s41269-023-00327-1.

26 Seyed Nader Nourbakhsh, Seyyed Abbas Ahmadi, Qiuomars Yazdanpanah Dero, and Abdolreza Faraji Rad. Rise of the Far Right Parties in Europe: from Nationalism to Euroscepticism. *Geopolitics Quarterly*, 18(4):47–70, 2021. URL: https://journal.iag.ir/article_129481.html.

27 OpenAI et al. GPT-4 Technical Report. *Computing Research Repository*, arXiv:2303.08774, 2023. doi:10.48550/arXiv.2303.08774.

28 OpenAI et al. GPT-4o System Card. *Computing Research Repository*, arXiv:2410.21276, 2024. doi:10.48550/arXiv.2410.21276.

29 Nils Reimers and Iryna Gurevych. Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4512–4525, 2020. doi:10.18653/v1/2020.emnlp-main.365.

30 Cesáreo Rodríguez-Aguilera. The rise of the far right in europe. *IEMed Mediterranean Yearbook*, 2014. URL: https://www.iemed.org/publication/the-rise-of-the-far-right-in-europe/.

31 Michael Röder, Andreas Both, and Alexander Hinneburg. Exploring the Space of Topic Coherence Measures. In *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM)*, pages 399–408, 2015. doi:10.1145/2684822.2685324.

32 Javier Torregrosa, Gema Bello-Orgaz, Eugenio Martinez-Camara, Javier Del Ser, and David Camacho. A Survey on Extremism Analysis using Natural Language Processing. *Computing Research Repository*, arXiv:2104.04069, 2021. doi:10.48550/arXiv.2104.04069.

33 Mattias Wahlström and Anton Törnberg. Social Media Mechanisms for Right-Wing Political Violence in the 21st Century: Discursive Opportunities, Group Dynamics, and Co-Ordination. *Terrorism and Political Violence*, 33(4):766–787, 2021. doi:10.1080/09546553.2019.1586676.

Mining GitHub Software Repositories to Look for Programming Language Cocktails

João Loureiro
ALGORITMI Research Centre/LASI - DI, University of Minho, Braga, Portugal

Alvaro Costa Neto
ALGORITMI Research Centre/LASI - DI, University of Minho, Braga, Portugal
Research Centre in Digitalization and Intelligent Robotics (CeDRI),
Laboratório Associado para a Sustentabilidade e Tecnologia em Regiões de Montanha (SusTEC),
Instituto Politécnico de Bragança, Portugal
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, Barretos, Brazil

Maria João Varanda Pereira
Research Centre in Digitalization and Intelligent Robotics (CeDRI),
Laboratório Associado para a Sustentabilidade e Tecnologia em Regiões de Montanha (SusTEC),
Instituto Politécnico de Bragança, Portugal

Pedro Rangel Henriques
ALGORITMI Research Centre/LASI - DI, University of Minho, Braga, Portugal

Abstract

In light of specific development needs, it is common to concurrently apply different technologies to build complex applications. Given that lowering risks, costs, and other negative factors, while improving their positive counterparts is paramount to a better development environment, it becomes relevant to find out what technologies work best for each intended purpose in a project. In order to reach these findings, it is necessary to analyse and study the technologies applied in these projects and how they interconnect and relate to each other. The theory behind *Programming Cocktails* (meaning the set of programming technologies – *Ingredients* – that are used to develop complex systems) can support these analysis. However, due to the sheer amount of data that is required to construct and analyse these Cocktails, it becomes unsustainable to manually obtain them. From the desire to accelerate this process comes the need for a tool that automates the data collection and its conversion into an appropriate format for analysis. As such, the project proposed in this paper revolves around the development of a web-scraping application that can generate Cocktail Identity Cards (CIC) from source code repositories hosted on GitHub. Said CICs contain the Ingredients (programming languages, libraries and frameworks) used in the corresponding GitHub repository and follow the ontology previously established in a larger research project to model each Programming Cocktail. This paper presents a survey of current Source Version Control Systems (SVCSs) and web-scrapping technologies, an overview of Programming Cocktails and its current foundations, and the design of a tool that can automate the gathering of CICs from GitHub repositories.

2012 ACM Subject Classification Software and its engineering → Software libraries and repositories

Keywords and phrases Software Repository Mining, Source Version Control, GitHub Scraping, Programming Cocktails

Digital Object Identifier 10.4230/OASIcs.SLATE.2025.13

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UID/00319/2023. The work of Maria João and Alvaro was supported by national funds: UID/05757 - Research Centre in Digitalization and Intelligent Robotics (CeDRI); and SusTEC, LA/P/0007/2020 (DOI: 10.54499/LA/P/0007/2020).

1 Introduction

As technology rapidly advances, more and more ways of creating and developing applications surface. Each project uses its own blend of programming languages, libraries and frameworks (that is, its Programming Cocktail [6]). As the development of a project begins with their own unique goals in mind, it becomes important to choose the right combination of these technologies in order to ensure its success. For example, it has been considered that *JavaScript*[1] is a better choice when it comes to developing a web application than *Python*, due to faster response times [4]. By analysing Programming Cocktails and their characteristics, many useful statistics can be extracted, such as which combinations of technologies work best with each other, and what particular technology is best suited for a specific task [26]. In summary, the more data obtained about these Programming Cocktail, the stronger and more reliable structural conclusions become. One strong caveat with this approach is the difficulty in obtaining said Cocktails for analysis, as currently there is not a tool that could gather them automatically, and the process to so by hand is inefficient and time consuming.

A possible solution to this problem would be the development of a web-scraping tool to mine software repositories, like the ones hosted on GitHub[2]. By extracting the relevant data needed to generate a Cocktail Identity Card (further explained in Section 2), one could structurally define the project's Programming Cocktail. This solution would allow for many Cocktails to be gathered automatically, accelerating further development into their theories and uses. This paper describes the main techniques and results already obtained in this endeavour, while establishing the architectural and functional components of a Programming Cocktail Identity Card extractor, currently under development.

This article is divided in six sections: after the Introduction, Section 2 introduces the main concepts behind Programming Cocktails and Cocktail Identity Cards (CIC). Section 3 provides an overview of Source Version Control Systems (SVCS), with emphasis on Git and GitHub, while Section 4 presents some web scrapping techniques and tools and discusses related work. Following that, Section 5 proposes the architecture and discusses the development of the automatic CIC extractor for GitHub, and finally, Section 6 summarizes the main concepts and ideas proposed, while providing remarks for the next phases of the project.

2 Programming Cocktails

The development process of an application requires the use of many different interacting components, each with its own purpose. As such, it can be useful to identify and categorize them for organizational and statistical purposes. The concept of a *Programming Cocktail* was created to aid in these tasks. It can be defined as an agglomerate of computer programming technologies (the *Ingredients*) that are used to develop a specific software application [6]. The four types of Ingredients are:

Language: any text or graphic based language used for programming, specifying, scripting, and so on (examples: Python, JavaScript, HTML, SQL, etc.);

Library: chunks of code that augment programming languages' standard instructions and commands (the language native statements) with new or additional functionality that increase their expressiveness (examples: BS4, GLib, etc.);

[1] Mozilla page about JavaScript: `https://developer.mozilla.org/en-US/docs/Web/JavaScript`
[2] Official GitHub website: `https://github.com`

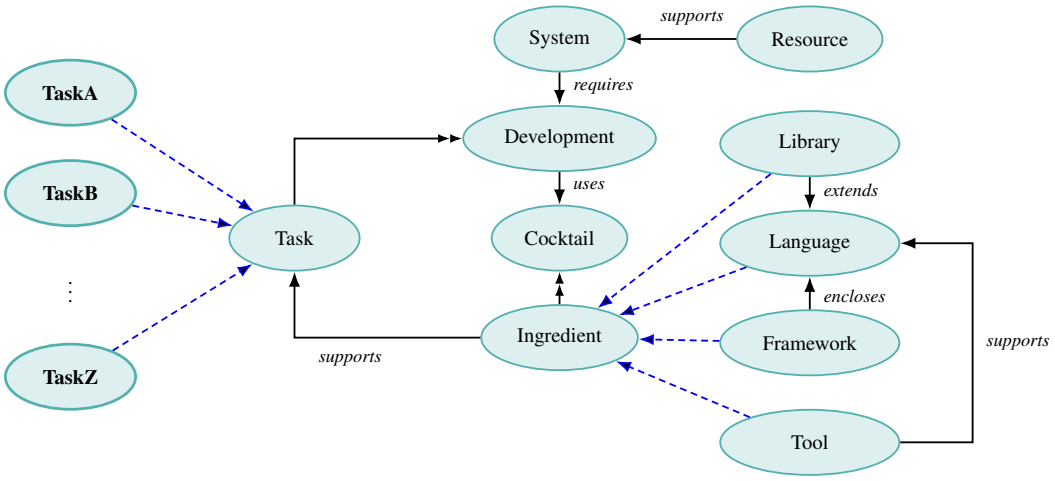


Figure 1 OntoCoq, the conceptual model for Programming Cocktails [6].

Framework: language extensions that enhance functionality and provide additional capabilities, while enforcing organizational patterns (examples: Scrapy, .NET, etc.);

Tool: any software package directly used in programming tasks during the software development process, like editors, debuggers, etc.

The ideas behind Programming Cocktails have been modelled through an ontology that describes their main concepts (see Figure 1, where the dashed blue arrows represent the "subclass-of (*isa*)" relationship, and the double-headed arrows represent the "is-part-of (*iof*)" relationship). In the ontology, each *Cocktail* is associated to a *System* that is or was under *Development*. *Systems* can be supported by *Resources*, that represent external systems and services used by the application that do not participate directly on its development (such as operating systems, database server runtimes, etc.). Finally, *Ingredients* support *Tasks* in the System's *Development*, and can represent simple, general tasks, such as frontend programming and full-stack development, or more specialized and distinct ones, like specific database communication, and memory management [6].

In order to store, identify and construct knowledge on Programming Cocktails, the concept of a Cocktail Identity Card (CIC) was created – an example of a CIC can be observed in Figure 2, modelled after a Q&A web application. The graph depicted in Figure 2 models "App1"s Programming Cocktail to formally define its Identity. Moreover, extra data can be added to the ontology to enable more types of analysis, such as risk, costs, cognitive load, etc. [5]. In short, a CIC is a resumed version of the conceptual model's instantiation, that can provide insights into the components used in the development of an application, and how they relate to each other. It also allows for the evaluation of different aspects of the application's development, such as the dependency on external resources, possible redundancies, tasks depending on many *Ingredients*, to name a few. Its continuous use can also support the documentation of a project's evolution, as long as the CIC is updated and versioned to form a Programming Cocktail history log.

3 Software Repositories

Version control is an important part of software development that consists in the management of changes to various files: programs, documents, models or other artefacts. It is a common aspect of managing a team in charge of large scale projects, offering control over any changes

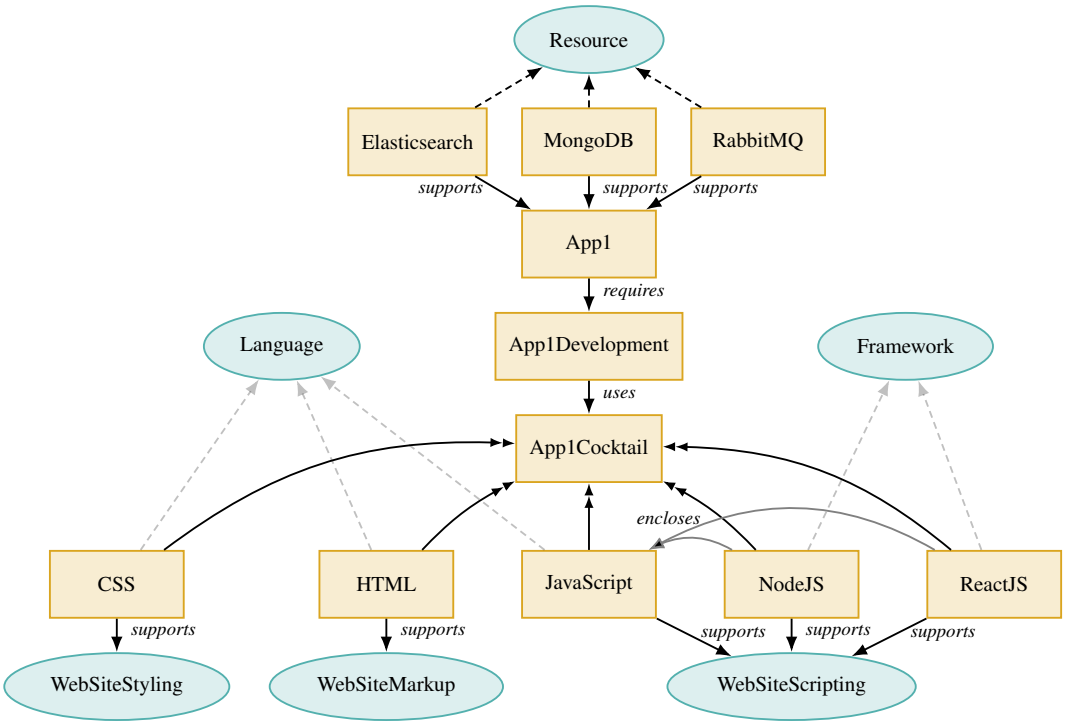


Figure 2 A fragment of OntoCoq to illustrate a Cocktail Identity Card for App1 [6].

made to the source code through various operations such as edit tracking, corrections and vandalism/spamming protection [8]. As such, Source Version Control Systems (SVCS) were devised to automate, synchronize, better safeguard, and improve efficiency of the entire version control process. By copying, merging and saving all the changes made to files, it is possible to overcome typical inefficiencies and errors in manual source control [16].

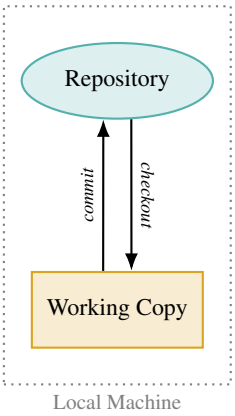
Serving as the foundation of any SVCS, a *repository* is the component responsible to store the *data and metadata* of each version of a file in a directory, as well as the history of changes made to its contents [14]. This archive is what allows one to switch between different versions, in case the need arises. The location and functionality of the repository greatly differs on what type of SVCS is used:

Local: in this type (exemplified by Figure 3a), as the name suggests, the repository containing all versions is maintained locally in the user's file system [16];

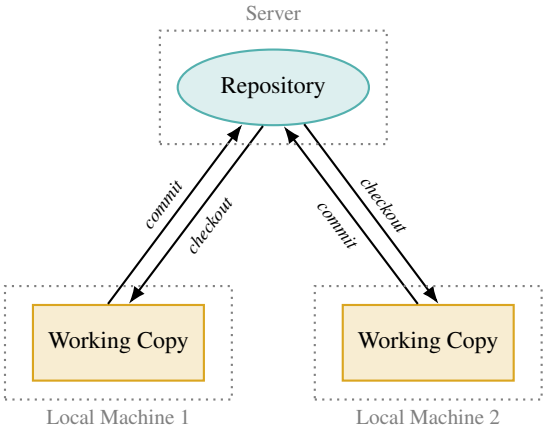
Centralized: a single repository is maintained centrally in this type of SVCS (see Figure 3b), working in a client-server fashion and storing a master copy of all files and their change history [28];

Distributed: this type of SVCS is a combination of the previous two types. It locally stores the entire history of the files in the repository, while also being capable of synchronizing the local changes back to a shared repository (Figure 3c) [28].

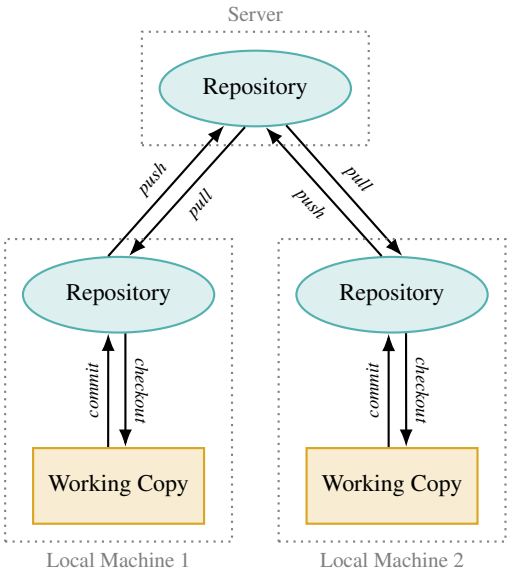
The *metadata* of the project stored in the repository is another key element for the version control. It refers to additional information about the content of the files and the repository itself, such as keywords, tags, formats, *etc.* providing an insight of the development context and history, without the need to look directly into the source code [13].



(a) Local version control system architecture.



(b) Centralized version control system architecture.



(c) Distributed version control system architecture.

Figure 3 Different repository architectures.

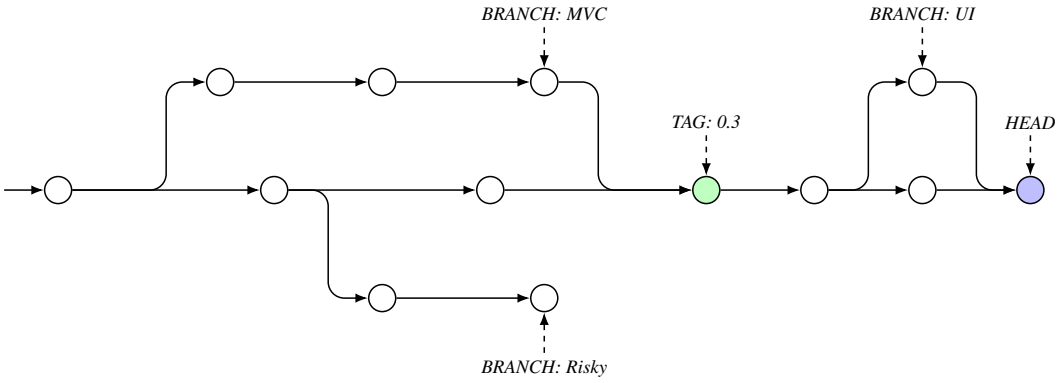


Figure 4 Representation of a Git commit graph.

3.1 Version Control Systems

There have been numerous SCVS implementations, the first notable one being Source Code Control System (SCCS)[3], a local version control system responsible for laying the base foundations of versioning files and how to store and retrieve them [12]. Revision Control System (RCS)[4] came after, allowing collaborative work between users while building upon the functionalities offered by SCCS as well as modifying how changes are stored. It resulted in a faster and more accessible alternative [25]. Finally, Concurrent Versions System (CVS)[5] is a centralized version control system that also applies the tree branch based system [2] employed by RCS to keep track of changes, while using a different method to handle changes made to the main repository: while RCS uses a lock system that prevents any changes being made simultaneously – a safer but more restrictive method – CVS employs an automatic merge system, attempting to merge any parallel changes that don't cause conflicts [2].

Out of all existing SCVS, *Git*[6] stands out as the most used SVCS according to Stack Overflow's 2022 Annual Developer Survey [19]. Beyond the previously discussed attributes of a *distributed* version control system, Git utilizes *commits* as the main way of discerning the various versions of a project. A commit is a snapshot of the current version, generally accompanied by a descriptive message. The repository is then comprised by a collection of commits organised into a graph that represents not only the current version, but also all versions that contributed to its state (see Figure 4 in which each circle represents a commit). Users can also create branches that diverge from the main one, allowing for parallel and simultaneous development of features without interference, that can be later merged into each other [17].

In terms of storage structures, Git implements different types of data objects for both structural info and files:

Blobs: store the content of any file put through version control;

Trees: store folders, meaning that they can also store other blobs and trees;

Commits: store metadata relevant to a revision, references to parent commits and the root tree;

Branches and Tags: files within Git that point to a commit [3];

[3] Open Group's Source Code Control System page: `https://pubs.opengroup.org/onlinepubs/9699919799/utilities/sccs.html`
[4] GNU OS' Revision Control System page: `https://www.gnu.org/software/rcs/`
[5] Concurrent Versions System info page: `https://www.nongnu.org/cvs/`
[6] Official Git site: `https://git-scm.com`

As for metadata, Git stores a collection of content taken from each commit, including its unique ID, author, attached message, creation date, and information pertaining to the files that were modified [13]. This means that the metadata is the main description for the versioning and subsequent identification of the repository's files, while being the identifying factor and differentiator of all versions.

3.2 GitHub

GitHub is an on-line development platform that allows users to store, manage and share code with one another through its repository hosting services. As the name implies, GitHub employs Git for version control, and offers additional services such as issue and bug tracking, access control, hosting of documentation, and more. It is also the site that houses the largest amount of source code repositories [27]. All projects hosted on GitHub are rooted on their Git code repositories, and, according to [1], one can identify five main elements in each one:

Commits: the change unit in a repository that represents a modification in the code, while also functioning as a snapshot of that version;

Issues: used to report bugs, questions pertaining to the project or announcements from the developers;

Pull requests: requests to perform changes to the code, such as merging branches;

Code reviews: contain suggestions and changes to be incorporated in the associated pull request;

Comments: the main form of communication between users, being them collaborators or not. Can be attached to issues, pull requests and code reviews.

A screenshot of a standard GitHub repository[7] can be seen in Figure 5.

As for users, they can be categorized in three types: *individual* which represents an account that belongs to a single user of the platform; *organization*, which entails a company, project or initiative; and *bot*, that provides services for repositories such as automated validation tasks [1].

4 Web Scrapping in Repositories

Data mining from the Web means to extract and compile relevant information from Webpages. One of the most used methods to achieve this is *Web Scraping*, a method that is used to extract data from websites either by directly accessing them through HTTP requests, or via a web browser. While it can be done manually, Web Scraping typically refers to the automated process that uses a *Web Crawler* to copy data found in the Web and store it locally to be later analysed and processed [22].

Since webpages are written using markup languages, such as Hypertext Markup Language (HTML), much of the desired information is in text format, allowing for the mining process to be relatively fast and flexible. As such, Web Scrapping is considered a solid approach for applications that collect several kinds of metrics, such as price trackers, product review collectors, weather data monitoring, and more [24]. It can be done through different techniques, such as HTML parsing or Document Object Model (DOM) Parsing. The former consists in fetching the raw HTML source code of a desired web page and generating its syntactic token parse tree. Then, a specialized extractor tool or system utilizes the generated

[7] KutoSchemaTools repository's page: `https://github.com/github/KustoSchemaTools`

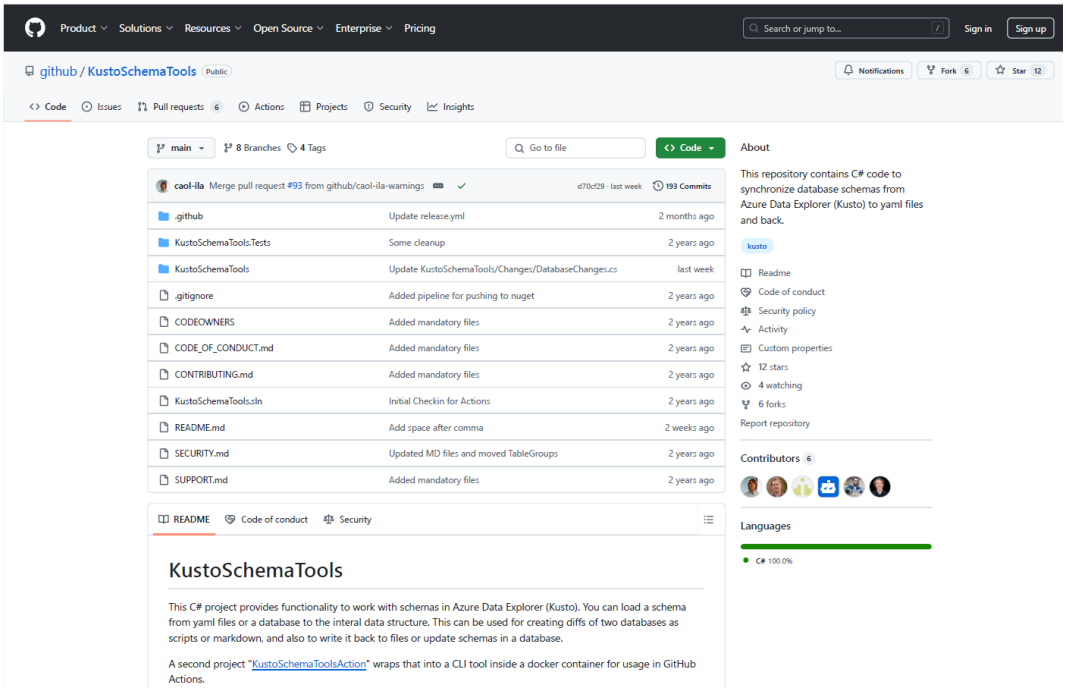


Figure 5 A GitHub repository's page as viewed in a web browser.

tree in order to extract the relevant information contained in the tags that make it up [21]. The latter utilizes the DOM[8] to create a structural tree out of the desired document. Said tree is comprised of various nodes, each representing an element/individual component of the source document, as well as itself. Within each element, there can be an identifying attribute, sub-elements, themselves elements, or text, where the context is located [23].

4.1 Technologies and Techniques to Implement Web Crawlers

There is a wide range of technologies that allow users to create a specialized Web Crawler. These can range from:

- Libraries, such as BeautifulSoup4 (BS4)[9] in Python or Jsoup[10] in Java, that implement their respective languages idioms on top of a HTML parser, allowing for the search, navigation and modification of the generated parse tree [20, 9];
- Frameworks, like Scrapy[11], that allows users to create Web Crawlers (*spiders*, in the framework's lingo) that extract data from websites or even perform automated testing [10];
- Or even tools, like Selenium[12], an open-source automation tool supported by various programming languages to test web applications across many different platforms and browsers.

[8] A platform-independent interface that allows the creation, navigation and modification of elements and content of HTML and XML documents.
[9] Official BS4 website: `https://www.crummy.com/software/BeautifulSoup/`
[10] Official Jsoup website: `https://jsoup.org`
[11] Official Scrapy website: `https://scrapy.org`
[12] Official Selenium website: `https://www.selenium.dev`

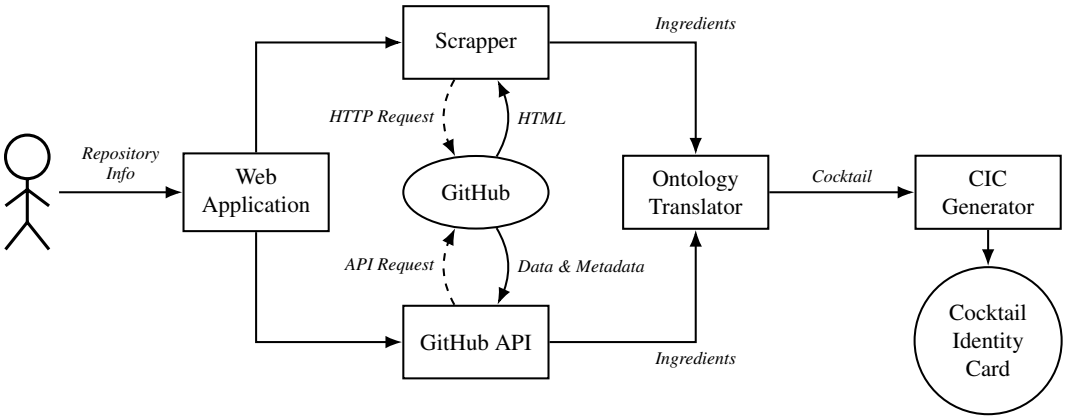


Figure 6 Architecture for the Cocktail Identity Card extractor.

Libraries like BS4 and Jsoup are easier to use and faster than more complex frameworks like Scrapy, that are more suited to scrape complex websites, and thus, larger projects [20]. These technologies can also be combined in order to cover each others limitations. As explored in [18], Selenium is used to take care of browser interactions that others cannot, in this case, to dynamically send data to input fields and press buttons on a website, while BS4 is used to grab the data from the tags.

In the specific case of mining GitHub repositories, [7] details a data extractor that was implemented to obtain characteristics frequently used as project selection criteria. It consists of a hybrid application that makes requests to GitHub's official Representational State Transfer (REST) Application Programming Interface (API) and also utilizes a specialized Web Crawler to scrape the repository's webpage. Said API allows users to extract information about the repositories it hosts, from general information like the description, owner and languages, to its contents or even analytics and collaboration data. The Crawler is used as a complement to obtain information that cannot be retrieved from the API alone. It was built primarily using Jsoup, while applying Selenium to handle the dynamically generated content. As the use of Selenium results in a large hit to the application's performance, it is only utilized as a backup strategy when Jsoup returns an error. The works detailed in [11] and [15], while similarly related in regards to the subjects of scrapping GitHub data by foraging repositories and using it to achieve a certain goal, have no direct relevancy to this project.

5 Mining GitHub to Extract Programming Cocktails

The focus of this paper is the proposal of a system able to extract, process, and analyse data from GitHub repositories in order to create Cocktail Identity Cards. Figure 6 shows its architecture.

The process of obtaining a CIC begins with the user inserting the target repository's information into the application, which in turn passes it into the *Scraper*. The *Scraper* uses this info to make requests to the *GitHub* servers in order to retrieve the information of the repository page and begin discovering its Ingredients. The application also utilizes the information to make requests to the *GitHub API* and obtain additional information on the repository. The relevant data that was scraped is then gathered and passed onto the *Ontology Translator*. This component will rearrange and transform the data into the ontology's syntax

and format. By following the conceptual model (Figure 1), it generates the concepts, the individuals and their relationships that can be identified in the repository. With the full ontology instantiated, the *CIC Generator* trims it down into its *Cocktail Identity Card*, which is then presented to the user.

5.1 Identifying Ingredients

In regards to identifying the Ingredients, possible strategies have already been devised, including:

- Scraping every webpage in the repository, including each individual file, as GitHub displays a specific webpage that contains the contents of each file in a repository.
- Searching specific contents in different file types, as each language can directly import modules or libraries via specific commands.
- Applying machine learning techniques to the repository's structure and the content of the *README* file (if present), which usually includes information about the project.
- Analysing the contents of common dependency files (such as `pyproject.toml` for Python, and `package.json` for JavaScript), which contain key components used in the project.

The final method for gathering Ingredients will include a combination of these strategies. When possible, the tool will prioritize API calls over Web Scraping to improve the speed and efficiency of the fetching and extracting processes.

As of the writing of this paper, the prototype of the application makes calls to GitHub API to retrieve information about the repository in JSON format. After inserting a valid personal access token, and the URL of the desired public repository, the application makes an request to the repository's endpoint[13] in order to retrieve general information about the repository such as its name, detailed information about the owner, its approximated size in kilobytes, the name of the default/main branch, other endpoint URLs that can be used to retrieve other information (tags, branches, collaborators...) and many others.

Next, it makes a request to the `languages` endpoint[14] that returns a JSON dictionary where the keys represent the languages that GitHub detected in the project, and the values associated to each key are the number of bytes of code that were written in that language, as seen in Listing 2. The information retrieved from this endpoint will be used to instantiate the *Language* concept of the ontology and all of the ingredients of this type. An example of a resulting Cocktail Identity Card containing only the languages obtained from the API is presented in Figure 7.

The `readme` endpoint[15] is then used to retrieve the raw content of the project's main `README.md` file. As specified in the API documentation, this content is encoded in Base64 and, as such, it is appropriately decoded back to UTF-8 using Python's `base64` module[16].

Then the application recursively uses the `branch` endpoints[17] to retrieve the name and path of all files present in the repository. This endpoint requires the specification of the name of the branch from which to pull the files from, with the desired target being the default branch – most commonly referred to as `main`. However, since this naming is not a rule and more of a convention, the application uses the `default_branch` parameter retrieved from

[13] `https://api.github.com/repos/{owner_name}/{repository_name}`
[14] `https://api.github.com/repos/{owner_name}/{repository_name}/languages`
[15] `https://api.github.com/repos/{owner_name}/{repository_name}/readme`
[16] Base64 module documentation: `https://docs.python.org/3/library/base64.html`
[17] `https://api.github.com/repos/{owner_name}/{repository_name}/git/trees/{branch_name}?recursive=1`

Listing 1 Example of a response from a repository endpoint. Repository used: https://github.com/cosmochefe/Strawberry.

```
{
  "id": 14406041,
  "node_id": "MDEwOlJlcG9zaXRvcnkxNDQwNjA0MQ==",
  "name": "Strawberry",
  "full_name": "cosmochefe/Strawberry",
  ...
  "owner": {
    "login": "cosmochefe",
    "id": 988484,
    ...
  }
  "html_url": "https://github.com/cosmochefe/Strawberry",
  "description": "Compilador did\u00e1tico para Portugol.",
  ...
  "collaborators_url": "https://api.github.com/repos/cosmochefe/
    ↪ Strawberry/collaborators{/collaborator}",
  "branches_url": "https://api.github.com/repos/cosmochefe/Strawberry/
    ↪ branches{/branch}",
  "tags_url": "https://api.github.com/repos/cosmochefe/Strawberry/tags",
  ...
  "size": 324,
  "default_branch": "master",
  ...
}
```

Listing 2 Example of a response from a repository's languages endpoint. Repository used: https://github.com/cosmochefe/Strawberry.

```
{
  "languages": {
    "C": 73481,
    "C++": 291
  }
}
```

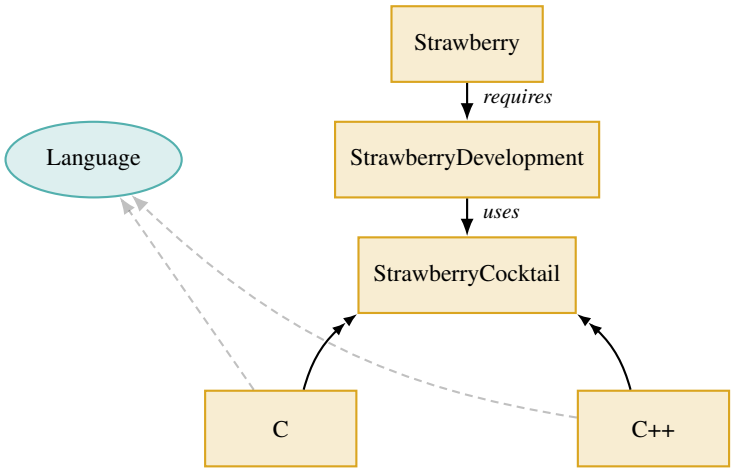


Figure 7 Cocktail Identity Card for the Strawberry repository (see Listing 2).

Listing 3 Example of a response from a repository's branch endpoint. Repository used: https://github.com/rust-lang/rust.

```
{
  ... ,
  {
    "path": "compiler",
    "type": "tree",
    "url": "https://api.github.com/repos/rust-lang/rust/git/trees/
    ↪ a0b0dae8da54f929eae78a96ce14fe526ea91e70"
  },
  {
    "path": "compiler/rustc",
    "type": "tree",
    "url": "https://api.github.com/repos/rust-lang/rust/git/trees/6
    ↪ f54be4ccb00e6baa2c949de41120a831645e8d0"
  },
  {
    "path": "compiler/rustc/Cargo.toml",
    "type": "blob",
    "url": "https://api.github.com/repos/rust-lang/rust/git/blobs/
    ↪ f4caa3ef769d5f9d9e6c960af53b7d3f0cca9cb8"
  },
  ...
}
```

the repository's endpoint to ensure that it is always the one targeted. The `recursive=1` flag is used to ensure that the file paths are retrieved recursively. The response is a JSON dictionary that contains a list of objects, one for each file and their attributes. Three main attributes, the most relevant to the task at hand, are chosen to be processed: `path`, the name of the path where one can locate the file, `type`, the Git type of the file as previously explained in Section 3.1 and `url`, the API endpoint of the file. This information is later used to reassemble the complete file structure of the repository.

Lastly, the application utilizes the GraphQL endpoint[18] in order to fetch the content of certain common dependency files. This endpoint is particularly useful for more complex and specific requests as it allows the use of GraphQL[19] queries. In this case, the query requests the dependency files seen in Table 1 by name, that when processed will return the raw content of those that are present. The information gathered from this endpoint, along with the one from the `readme` and the `branch` one, will be used to instantiate the *Library*, *Framework* and possibly *Tool* concept of the ontology and to later represent the ingredients of these types.

One big limitation encountered so far is related to the GitHub API rate limits. In order to obtain the information of a repository, multiple calls to the API must be made. Without user authentication, only 60 requests per hour can be made, while when authenticated it is possible to perform up to 5000 request or 15000 if the user is related to the GitHub Enterprise Cloud[20]. This means that the application will have to enforce GitHub authentication of some kind in order to increase the instance limit, try to limit as much as possible the number of requests made to the API and block usage once the limit has been reached for that hour.

[18] `https://api.github.com/graphql`
[19] Official GraphQL website: `https://graphql.org`
[20] `https://docs.github.com/en/rest/using-the-rest-api/rate-limits-for-the-rest-api`

Table 1 Dependency files that the application currently requests, sorted by their language.

Language	Dependency Files
Python	requirements.txt, pyproject.toml, setup.py
JavaScript	package.json, yarn.lock, package-lock.json
Java	pom.xml, build.gradle
Ruby	Gemfile, Gemfile.lock
PHP	composer.json, composer.lock
Go	go.mod, go.sum
Rust	Cargo.toml, Cargo.lock
.NET Languages	.csproj, packages.config

Listing 4 Format of the GraphQL query used to retreive the contents of the dependency files.

```
query FetchDependencyFiles(${owner_name}: String!,
                           ${repo_name}: String!) {
  repository(owner: ${owner_name}, name: ${repo_name}) {
    # Repeat for each files
    requirements: object(expression: "HEAD:{file_name}") {
      ... on Blob { text }
    }
  }
}
```

Immediate follow-up work will involve analysing the data currently being fetched to determine what information can be extracted and the best methods for doing so. Alternate ways to obtain information that may arise will also be explored and implemented if deemed acceptable, such as the implementation of some of the web scraping techniques discussed in order to retrieve information only present in the repository's HTML page, or even potentially replace an endpoint request to reduce the number of requests made.

6 Conclusion

This paper highlighted the main concepts behind data mining of software repositories, more specifically, GitHub. It also presented the foundations of Programming Cocktails, Cocktail Identity Cards and their ontology-based models. Finally in the paper we proposed, as the main contribution, the architecture of an automated web application aimed at discovering Programming Cocktails and generate their Identity Cards based on data extracted from GitHub repositories. These results will provide a deeper understanding on the myriad of programming technologies commonly present in the development of complex applications. By automating the gathering of Programming Cocktails, more results and conclusions on this topic can be reached, extending knowledge on both software development complexity and evolution.

Currently, the extractor is capable of constructing CICs containing the languages identified in the Github API, while also gathering data from many other sources of information. Such as the `README.md` file, the overall repository folder structure and some characteristic dependency files.

The next phase of this project will continue with the implementation of the Cocktail Identity Card extractor, taking into account possible paths for discovery, such as the ones presented in Section 5, as well as the development and implementation of methods to analyse

and extract Ingredients from the retrieved information. After that, a set of public GitHub repositories will be defined to be used as a case study to test and optimize said methods, as well as identifying other possible additions and options to the process. Beyond what was discussed in this article, other alternative approaches could be explored, such as one based purely on machine learning techniques, focused on the creation and training of a robust identification model. Lastly, some other techniques for automatic discovery of *Resources* and *Tasks* are planned to be implemented in order to complete the extraction of Cocktail Identity Cards.

References

1 Adem Ait, Javier Luis Cánovas Izquierdo, and Jordi Cabot. An empirical study on the survival rate of github projects. In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22, pages 365–375, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3524842.3527941.

2 John Alhson. Parallel (concurrent) code development or how to develop software in a distributed workgroup or the concurrent versions system, cvs. *Nuclear Physics B - Proceedings Supplements*, 61(3):674–677, 1998. Proceedings of the Fifth International Conference on Advanced Technology and Particle Physics. doi:10.1016/S0920-5632(97)00636-1.

3 Natanael Arndt, Patrick Naumann, Norman Radtke, Michael Martin, and Edgard Marx. Decentralized collaborative knowledge management using git. *Journal of Web Semantics*, 54:29–47, 2019. Managing the Evolution and Preservation of the Data Web. doi:10.1016/j.websem.2018.08.002.

4 Sai Sri Nandan Challapalli, Prakarsh Kaushik, Shashikant Suman, Basu Dev Shivahare, Vimal Bibhu, and Amar Deep Gupta. Web development and performance comparison of web development technologies in node.js and python. In *2021 International Conference on Technological Advancements and Innovations (ICTAI)*, pages 303–307, 2021. doi:10.1109/ICTAI53825.2021.9673464.

5 Alvaro Costa Neto, Maria João Varanda Pereira, and Pedro Rangel Henriques. Application of programming cocktails identity cards to development complexity analysis. In *Proceedings of the 23nd Belgium-Netherlands Software Evolution Workshop (BENEVOL 2024)*. CEUR-SW.org, 2024. To be published.

6 Alvaro Costa Neto, Maria João Varanda Pereira, and Pedro Rangel Henriques. An ontology to understand programming cocktails. In Maria Ganzha, Leszek Maciaszek, Marcin Paprzycki, and Dominik Ślezak, editors, *Proceedings of the 19th Conference on Computer Science and Intelligence Systems*, Annals of Computer Science and Information Systems. IEEE, 2024. To be published.

7 Ozren Dabic, Emad Aghajani, and Gabriele Bavota. Sampling projects in github for msr studies. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 560–564, 2021. doi:10.1109/MSR52588.2021.00074.

8 N. Deepa, B. Prabadevi, L.B. Krithika, and B. Deepa. An analysis on version control systems. In *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, pages 1–9, 2020. doi:10.1109/ic-ETITE47903.2020.39.

9 Yılmaz Dikilitaş, Çoşkun Çakal, Ahmet Can Okumuş, Halime Nur Yalçın, Emine Yıldırım, Ömer Faruk Ulusoy, Bilal Macit, Aslı Ece Kırkaya, Özkan Yalçın, Ekin Erdoğmuş, and Ahmet Sayar. Performance analysis for web scraping tools: Case studies on beautifulsoup, scrapy, htmlunit and jsoup. In Fausto Pedro García Márquez, Akhtar Jamil, Alaa Ali Hameed, and Isaac Segovia Ramírez, editors, *Emerging Trends and Applications in Artificial Intelligence*, pages 471–480, Cham, 2024. Springer Nature Switzerland.

10 M El Asikri, S Knit, and H Chaib. Using web scraping in a knowledge environment to build ontologies using python and scrapy. *European Journal of Molecular & Clinical Medicine*, 7(03):2020, 2020.

11 Hamzeh Eyal Salman. Ai-based clustering of similar issues in github's repositories. *Journal of Computer Languages*, 78:101257, 2024. doi:10.1016/j.cola.2023.101257.

12 Alan L. Glasser. The evolution of a source code control system. *SIGSOFT Softw. Eng. Notes*, 3(5):122–125, January 1978. doi:10.1145/953579.811111.

13 Youngtaek Kim, Jaeyoung Kim, Hyeon Jeon, Young-Ho Kim, Hyunjoo Song, Bohyoung Kim, and Jinwook Seo. Githru: Visual analytics for understanding software development history through git metadata analysis. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):656–666, 2021. doi:10.1109/TVCG.2020.3030414.

14 Ali Koc and Abdullah Uz Tansel. A survey of version control systems. *ICEME 2011*, 2011.

15 Sandeep Kaur Kuttal, Se Yeon Kim, Carlos Martos, and Alexandra Bejarano. How end-user programmers forage in online repositories? an information foraging perspective. *Journal of Computer Languages*, 62:101010, 2021. doi:10.1016/j.cola.2020.101010.

16 Rana Majumdar, Rachna Jain, Shivam Barthwal, and Chetna Choudhary. Source code management using version control system. In *2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pages 278–281, 2017. doi:10.1109/ICRITO.2017.8342438.

17 Edi Muškardin, Tamim Burgstaller, Martin Tappler, and Bernhard K. Aichernig. Active model learning of git version control system. In *2024 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 78–82, 2024. doi:10.1109/ICSTW60967.2024.00024.

18 Simona Vasilica Oprea and Adela Bâra. Why is more efficient to combine beautifulsoup and selenium in scraping for data under energy crisis. *Ovidius University Annals, Economic Sciences Series*, 22(2):146–152, 2022.

19 Stack Overflow. Stack overflow developer survey 2022. https://survey.stackoverflow.co/2022/#version-control-version-control-system, 2022. Last Accessed: 12/01/2025.

20 Sakshi Pant, Er. Narinder Yadav, Milan, Monnie Sharma, Yash Bedi, and Anshuman Raturi. Web scraping using beautiful soup. In *2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS)*, volume 1, pages 1–6, 2024. doi:10.1109/ICKECS61492.2024.10617017.

21 Pranit Patil, Pramila Chawan, and Prithviraj Chauhan. Parsing of html document. *International Journal of Advanced Research in Computer Engineering & Technology*, 1:2278–1323, July 2012.

22 Ruchitaa Raj N R, Nandhakumar Raj S, and Vijayalakshmi M. Web scrapping tools and techniques: A brief survey. In *2023 4th International Conference on Innovative Trends in Information Technology (ICITIIT)*, pages 1–4, 2023. doi:10.1109/ICITIIT57246.2023.10068666.

23 Martina Radilova, Patrik Kamencay, Robert Hudec, Miroslav Benco, and Roman Radil. Tool for parsing important data from web pages. *Applied Sciences*, 12(23), 2022. doi:10.3390/app122312031.

24 Vidhi Singrodia, Anirban Mitra, and Subrata Paul. A review on web scrapping and its applications. In *2019 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–6, 2019. doi:10.1109/ICCCI.2019.8821809.

25 Walter F. Tichy. Design, implementation, and evaluation of a revision control system. In *Proceedings of the 6th International Conference on Software Engineering*, ICSE '82, pages 58–67, Washington, DC, USA, 1982. IEEE Computer Society Press. URL: http://dl.acm.org/citation.cfm?id=807748.

26 Federico Tomassetti and Marco Torchiano. An empirical assessment of polyglot-ism in github. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2601248.2601269.

27 Wikipedia. Comparison of source-code-hosting facilities. https://en.wikipedia.org/wiki/Comparison_of_source-code-hosting_facilities#Popularity, 2024. Last Accessed: 05/10/2024.

28 Nazatul Nurlisa Zolkifli, Amir Ngah, and Aziz Deraman. Version control system: A review. *Procedia Computer Science*, 135:408–415, 2018. The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life. doi:10.1016/j.procs.2018.08.191.