# 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.

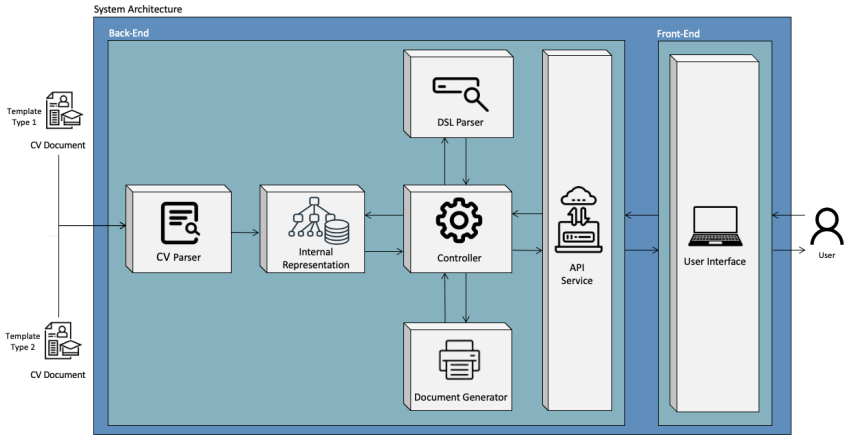



**Figure 1** System Architecture.

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.

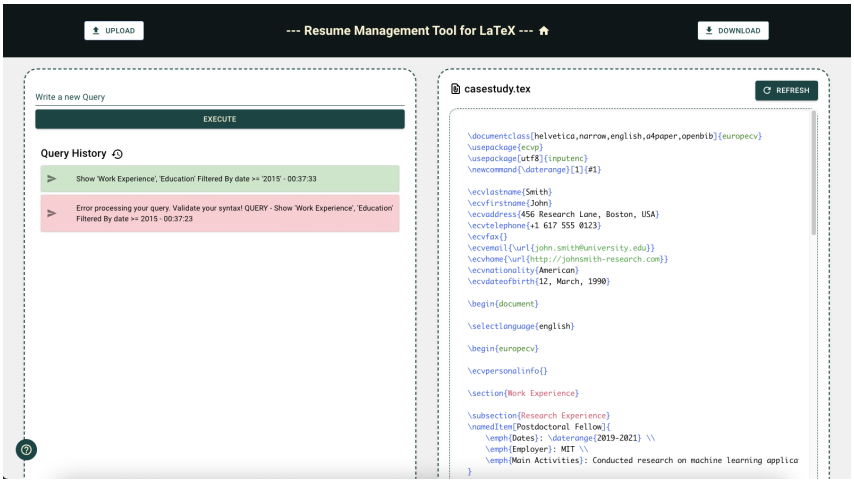



**Figure 2** Tool Page.

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.

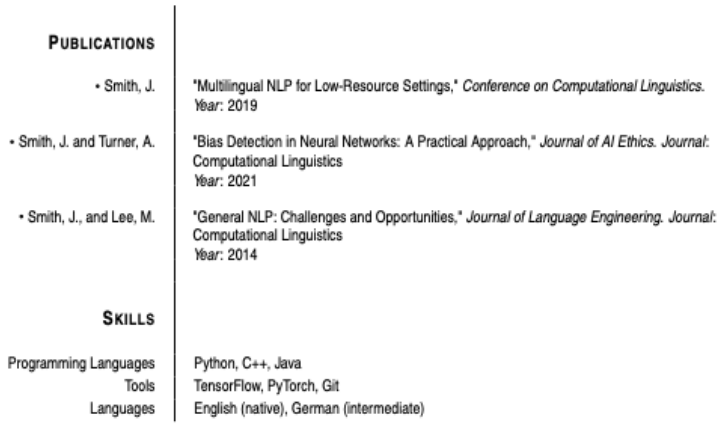
## EDUCATION

- 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<br>*Instituicao*: Universidade de Harvard<br>*Grau*: Ciencia da Computacao |

### EXPERIENCIA DE TRABALHO

#### EXPERIENCIA EM PESQUISA

| | |
|---|---|
| • Bolsista de Pos-Doutorado | *Datas*: 2019-2021<br>*Empregador*: COM<br>*Principais Atividades*: Conduziu pesquisas sobre aplicacoes de aprendizado de maquina em robotica. |
| • Assistente de Pesquisa | *Datas*: 2015-2019<br>*Empregador*: Universidade de Harvard<br>*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*.<br>*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.