Exploiting Generative AI to Scale up Intelligent Tutoring Systems

Miguel Ferreira
DCC – FCUP, Porto, Portugal

José Paulo Leal
CRACS – INESC TEC, Portugal
DCC – FCUP, Porto, Portugal

Abstract

The research described in this paper explores the integration of Generative Artificial Intelligence (GenAI) into Intelligent Tutoring Systems (ITS) with the aim of improving programming education. Specifically, it extends Agni, a web-based programming learning platform, by automating the construction of key ITS components that are traditionally built manually. The methodology leverages Large Language Models (LLMs) to extract programming concepts from educational materials, map their relationships via concept graphs and diagnose specific student knowledge gaps. Additionally, GenAI provides real-time, contextual feedback during programming exercises, mimicking the personalized support typically offered by a human tutor. The work evaluates whether GenAI can effectively replace or augment the manual creation of domain models while preserving the pedagogical benefits of ITS. The obtained results suggest the potential of combining AI with ITS to improve scalability and reduce manual workload.

2012 ACM Subject Classification Applied computing → Interactive learning environments; Computing methodologies → Natural language generation

Keywords and phrases Intelligent Tutoring Systems, Large language Models

Digital Object Identifier 10.4230/OASIcs.ICPEC.2026.2

Funding This work is funded by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the support UID/50014/2025 (`https://doi.org/10.54499/UID/50014/2025`).

1 Introduction

Programming is an important skill in modern society, with strong demand across industries. Despite this demand, learning to program remains challenging for beginners. Research shows that while novices can understand individual programming concepts, they often struggle to combine them into coherent programs. Because of this, effective programming education relies heavily on practice supported by good feedback from tutors.

Intelligent Tutoring Systems have traditionally addressed this need by providing adaptive feedback and personalized instruction. However, a major limitation of classical ITS lies in their reliance on manually engineered components, such as concept graphs. Constructing these components requires significant effort from tutors and does not scale well across different subjects.

Recent advances in GenAI, particularly LLMs, created new capabilities like producing new content, with natural language input, by learning from large datasets, this provides a new opportunity to automatically generate the key components of an ITS. Thus, the research question that drove this work was the following: **Can an LLM replace a human tutor in the repetitive and labor intensive tasks required by ITS?**

This research question led to the goal of extending Agni to support ITS. Agni is a web-based playground to learn programming [3]. By automating the construction of ITS components, this approach aims to reduce the manual effort required, improving scalability while preserving the pedagogical effectiveness of ITS. This way, educators could more easily adopt ITS in real educational environments, making it more scalable.

The main objectives of this work are:

Automatic Concept Extraction: Develop methods to extract the taught programming concepts from educational materials using GenAI.

Concept Graph Generation: Use GenAI to construct structured representations of knowledge (concept graphs) that capture dependencies between programming concepts.

Failed Concepts Diagnosis: Find which programming concepts the student failed on an exercise by using GenAI.

Feedback: Provide contextualized feedback during an exercise attempt to the student, with the use of GenAI.

The paper is structured into several sections. Section 2 reviews the state of the art, focusing on ITS and the use of GenAI in programming education. Section 3 describes the system design, including the underlying tools and integration of AI components. Section 4 presents the validation methodology and discusses the obtained results. Finally, Section 5 concludes the paper by summarizing the main contributions and outlining possible directions for future work.

The goal of the validation process is not to evaluate the effectiveness of ITS themselves, as they have already been studied in the literature, but rather to assess the quality of the AI outputs in comparison with human judgment

2 State of the Art

This chapter reviews the current state of research relevant to this work, establishing the foundation for the proposed research and development. The review focuses on ITS and their underlying principles, as well as recent advances in AI, particularly the use of LLMs in programming education.

2.1 Intelligent Tutoring Systems

Intelligent Tutoring Systems (ITS) are systems designed to simulate the one-on-one cognitive and pedagogical benefits of a human tutor by adapting instruction to the needs, knowledge state, and learning pace of individual students.

Kurt Vanlehn [17] describes these systems as computer-based instructional environments that provide immediate, customized instruction and feedback to learners without requiring human intervention. Comprehensive analyses by Wei Ma et al. [10], and James Kulik and J. D. Fletcher [8] demonstrate that ITS interventions significantly improve learning outcomes and knowledge retention when compared to traditional studying from static texts.

Beverly Woolf [19] formalizes the classical ITS architecture as comprising a domain model, a student model and a tutoring model, often extended with an interface model.

The domain model represents the knowledge and skills to be taught. This is typically encoded as production rules or knowledge components (KCs), which correspond to skills required to solve a problem [2]. For example, in programming, KCs might include skills such as conditional statements or variables. In our work, instead of manually defining KCs, programming knowledge is represented using programming concepts automatically extracted from learning materials.

Knowledge graphs represent learning content as nodes (concepts) connected by directed edges (prerequisite relationships). For example, a simple concept graph might include nodes such as variables, functions, recursion, with edges indicating dependencies, such as variables to functions and functions to recursion.

Traditional ITS required experts to manually construct these KCs, this process is labor-intensive, domain-specific and difficult to scale. This domain building bottleneck limited the deployment of adaptive systems [16].

The student model maintains a dynamic representation of the learner's current knowledge state. It estimates mastery levels for individual KCs and updates these estimates based on observed performance. One of the most influential approaches is Bayesian Knowledge Tracing (BKT), introduced by Albert T. Corbett and John R. Anderson [4].

BKT models the acquisition of individual skills as a hidden Markov process in which each knowledge component is assumed to be either mastered or not mastered. It is parameterized by four probabilities: the initial probability of mastery, the probability of learning (transition from unmastered to mastered) and the probabilities of guessing and slipping, which account for correct responses despite lack of mastery and incorrect responses despite mastery, respectively. Based on student interactions, the model updates the probability that a learner has mastered a given concept. A concept is considered not yet mastered when this probability remains below a predefined threshold, indicating insufficient evidence of consistent performance.

An alternative approach to student modeling is Deep Knowledge Tracing (DKT), which uses recurrent neural networks to capture complex learning patterns, while it as shown effectiveness, it struggles with smaller datasets [20].

In this research, BKT was selected due to being effective in eductional settings with limited data and its simplicity, serving as a historically validated mechanism for updating student mastery in ITS.

The tutoring model governs instructional decisions. It selects tasks, generates hints, determines feedback timing and decides when a learner has achieved sufficient mastery. In cognitive tutors, this model often relies on mastery thresholds derived from probabilistic student modeling [2].

Finally, the interface model mediates the interaction between learner and system. The interface influences cognitive load and learner engagement.

Together, these components form the structural backbone of ITS and remain conceptually stable despite shifts in underlying computational techniques.

2.2 Generative AI in Programming Education

Generative Artificial Intelligence, particularly LLMs based on transformer architectures [18], present significant opportunities for programming education. Due to their ability to process and generate natural language, as well as support active interaction with the user, LLMs are well-suited for integration into learning environments. However, challenges regarding academic integrity, bias and hallucinated information remain [1]. As a result, prompting techniques such as zero-shot, few-shot, and chain-of-thought prompting have become increasingly important for ensuring the reliability and effectiveness of these systems [15].

Recent literature highlights several uses of GenAI in programming education. One prominent use is the provision of personalized feedback. LLMs are capable of interpreting problem statements alongside student submissions to generate contextualized hints. For instance, Tung Phung et al. [14] reported that while human tutors scored 92.0/100 on hint utility metrics, GPT-4 still attained a competitive score of 66.0/100. Similarly, Maciej

Pankiewicz and Ryan Baker [13] demonstrated that LLM-generated feedback for compiler errors reduced the number of non-compiling submissions, with 81% of students rating the feedback as useful.

Furthermore, incorporating submission history into feedback generation has been shown to improve user preference, with students favoring this approach 69% of the time [12]. Another important application of GenAI is exercise generation. These systems can significantly reduce instructor workload by automatically generating multiple-choice questions (MCQs) aligned with specific learning objectives [5]. Moreover, contextualizing programming exercises using AI has also been shown to yield high-quality problems that increase student engagement and motivation [6]. LLMs have also demonstrated the ability to generate quizzes for Java programming courses with a relatively low error rate of approximately 10% [9].

Subhankar Maity and Aniket Deroy [11] highlight possible applications for GenAI in ITS. These include automated question generation tailored to the learner´s current level of understanding, providing personalized feedback tailored to the specific errors and misconceptions identified and iterative dialog systems to simulate human-like conversations and guide students through learning complex concepts in a conversational manner.

3 System Design

Agni is a web-based platform designed for learning programming. It consists of a teacher interface (for content and module management) and a student interface (for accessing exposition materials and evaluative components like quizzes and coding exercises).

The frontend of the platform was developed with Vue and the backend with Strapi. Vue is a component based JavaScript framework used for building user interfaces and single-page applications. While Strapi is an open-source headless content management system (CMS).

Within the context of this work, Agni serves as the core platform where the proposed features are implemented and evaluated.

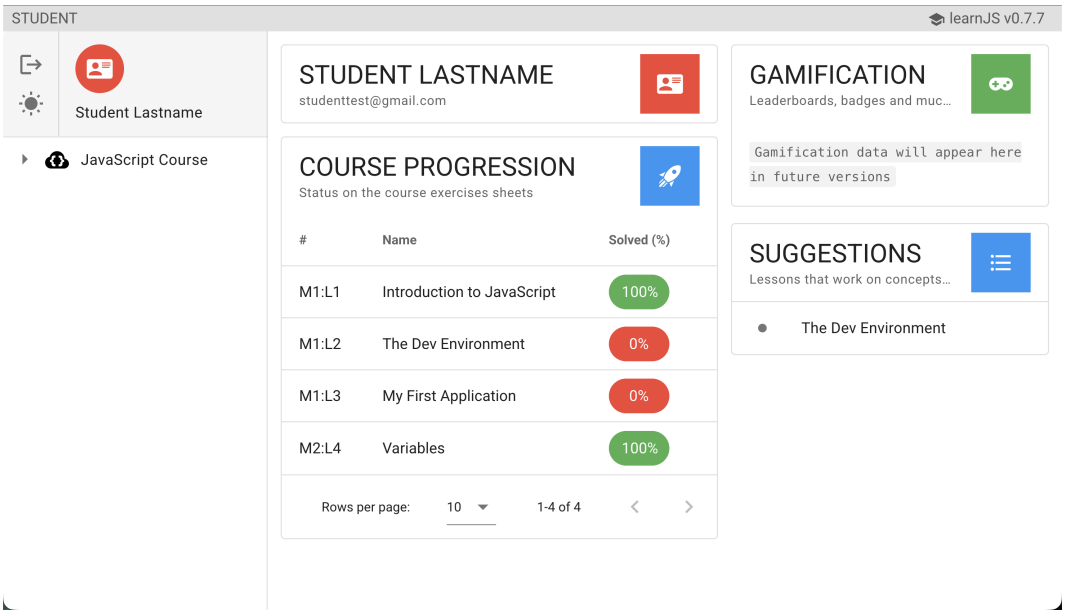

Figure 1 Student Interface.

3.1 Adaptive Learning

To track student knowledge, the system integrates Bayesian Knowledge Tracing (BKT), enabling a probabilistic representation of student mastery of individual concepts. To support this, a new ConceptMastery collection was introduced in the data model. This structure associates each student with a set of probabilities representing their mastery level for each concept.

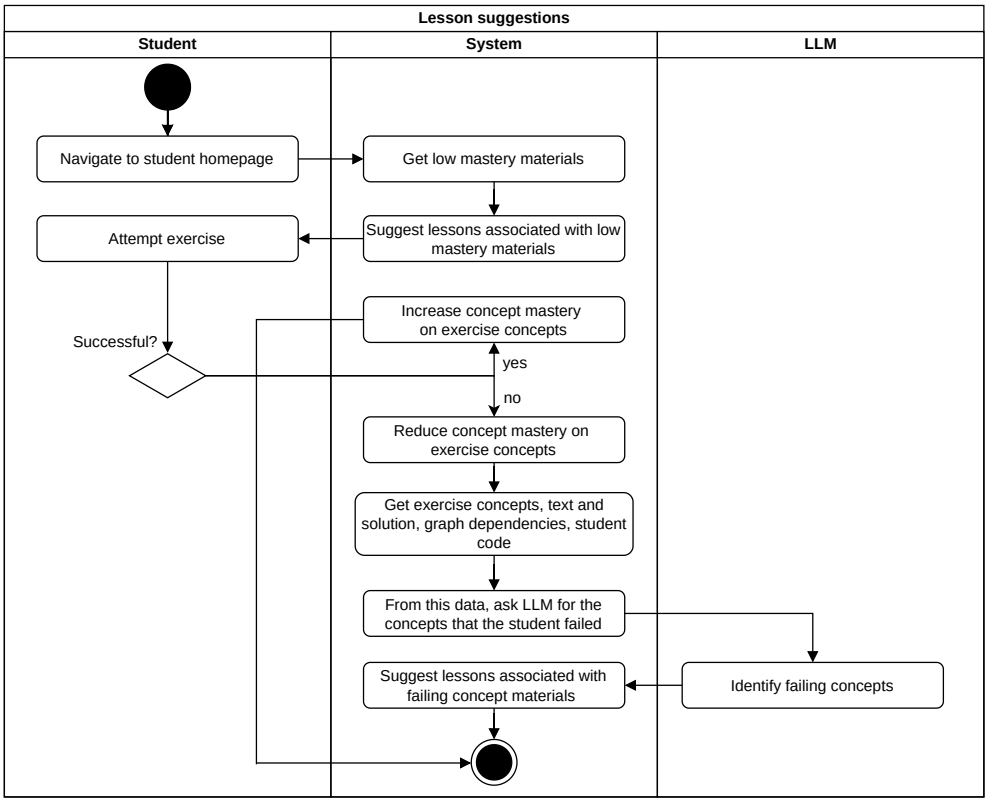

Figure 2 Lesson suggestions activity diagram.

As illustrated in Figure 2, the process begins when a student interacts with the system by attempting an exercise. Based on the outcome, the system determines whether the attempt was successful and updates, using BKT, the corresponding student mastery values of the concepts associated with the exercise through the ITS system. The previous mastery values of the concepts and the attempt outcome are used in this calculation. A successful attempt increases the probability of mastery for the concepts associated with the exercise, while an unsuccessful attempt decreases it. This allows the system to maintain a continuously updated representation of the student's knowledge state.

When a student fails to solve an exercise, the system performs a deeper analysis to identify the underlying causes. It retrieves relevant information, including the concepts associated with the material, their dependencies within the concept graph, the student's submitted code, the problem description and the expected solution. The described information is then provided to the LLM through a prompt that requests the concepts that likely caused the student to fail, this identifies the specific concepts the student is struggling with. Having the list of failed concepts returned by the AI, the system searches and recommends lessons associated with these concepts through the learning materials.

In addition, the system is capable of suggesting lessons covering materials for which the student has low mastery by getting all concept masteries calculated with BKT, checking which ones are below a defined threshold and finding lessons associated with those concepts through the Expositives or Evaluatives, illustrated in Figure 1 through the Suggestions panel. These recommendations are presented through the Agni interface, guiding the student toward relevant content.

This allows the system to personalize the learning experience by guiding students toward content that addresses their weaknesses, improving learning efficiency and engagement.

3.2 Concept Extraction

A key component of the system is the automatic extraction of programming concepts from educational materials. As illustrated in Figure 3, this process involves the teacher, the system and an LLM. The output generated by the system within Agni can be seen in figure 4.

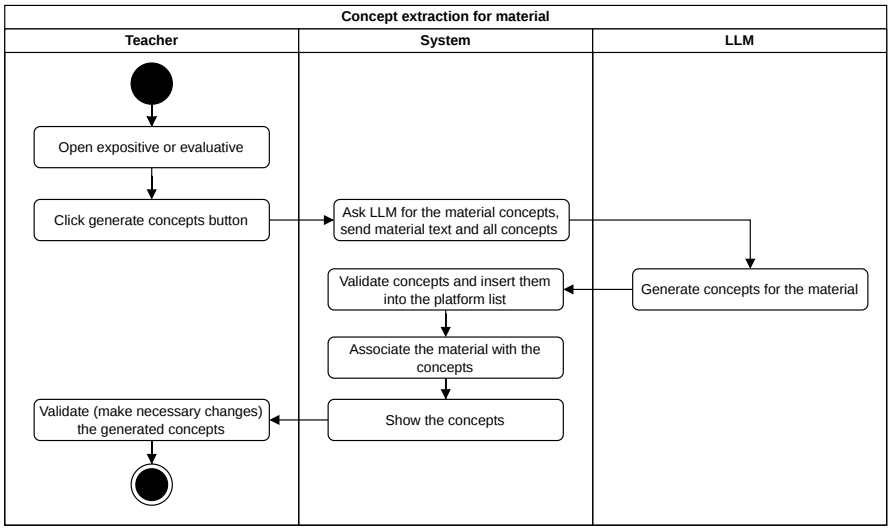

Figure 3 Concept extraction activity diagram.

The workflow begins when the teacher initiates the process by selecting the generate concepts option within a selected Expositive (material) or Evaluative (exercise) in the Agni platform.

The system then constructs a structured prompt that includes the material text along with any existing concepts in Agni and sends this information to the LLM. Based on this input, a list of concepts that represent the knowledge explicitly covered in the material is generated. Since the output of this process has high variance, this operation is repeated 5 times and the concepts the appear the majority of the time stay in the final output.

Once the concepts are returned, the system performs a validation step to ensure consistency. This includes aligning the generated concepts with the existing ones in the platform. The validated concepts are then used to automatically tag the material, establishing a relationship between the content and the extracted concepts. The resulting concept list is presented to the teacher through the Agni interface.

At this stage, the teacher remains in the loop and can add or remove concepts as needed, maintaining control over the domain model. This semi-automated approach balances workload with user control. To support this functionality, the data model was extended with a new Concept collection.

For example, given a learning resource introducing basic programming, the system may extract concepts such as variables or functions. These concepts are then associated with the material and can later be used for tasks such as student modeling and graph generation.

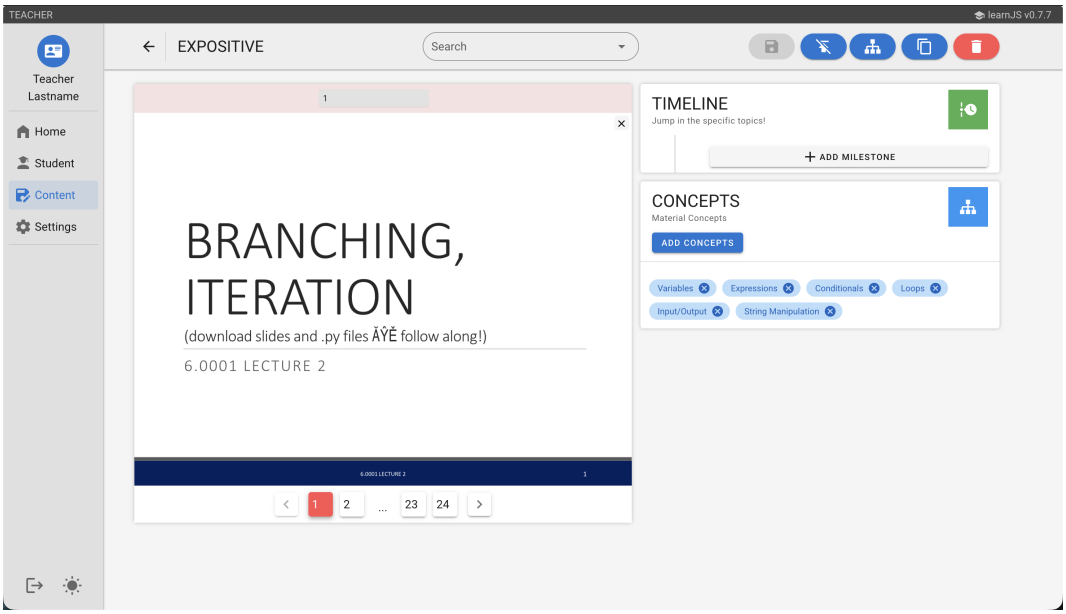

Figure 4 Extracted Concepts.

3.3 Concept Graph Generation

Building on the extracted concepts, the system supports the automatic generation of graphs representing relationships between programming concepts at the course level. As illustrated in Figure 6, this process involves the teacher, the system and an LLM. The teacher provides high-level validation, while the LLM assists in identifying and structuring relationships between concepts.

The workflow is initiated by selecting the generate graph option within a selected course in the Agni platform. The system first ensures that all course materials are associated with concepts. For materials that have not yet been tagged, the system queries the LLM to extract the corresponding concepts using the approach described in Section 3.2.

Once every material in the course has an associated list of concepts, the system constructs a structured prompt that includes all the course material concepts and the graph rules, such as output directed edges and use only the provided concepts. Prompt-engineering techniques are employed, namely few-shot and constraint-based prompting, and a request to the selected LLM is made, generating a concept graph. The output is a directed graph in which nodes represent concepts and edges represent prerequisite relationships. This representation captures dependencies between topics, enabling the system to model the progression of knowledge.

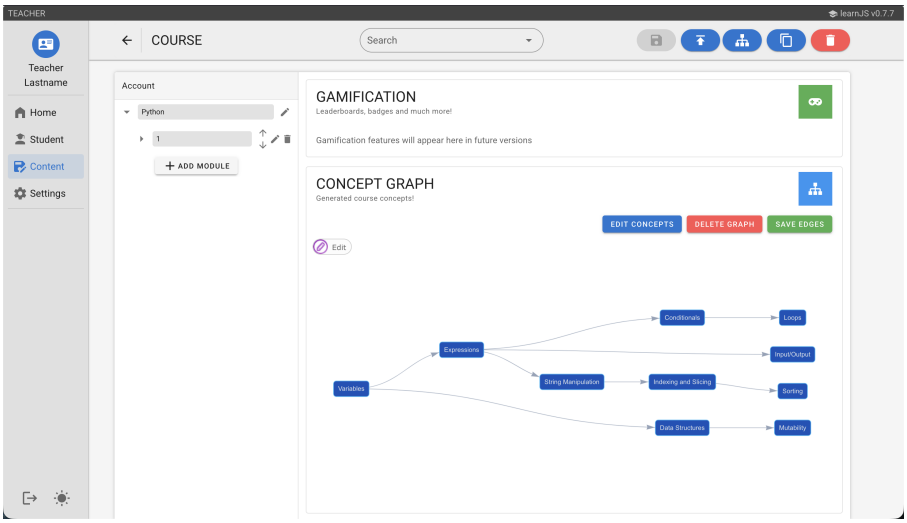

Figure 5 Generated Graph.

The generated graph is then presented to the teacher through Agni, as shown in Figure 5. As with concept extraction, the teacher remains in the loop and can make adjustments such as adding or removing concepts and edges, to ensure control remains with the user.

Strapi was extended with new ConceptGraph and ConceptEdges collections and the existing Course collection was updated with a conceptGraph field, representing the relationship to the ConceptGraph.

By automating this process, this system reduces the need for manual building of the domain model while preserving teacher control. This approach reduces workload and improves scalability.

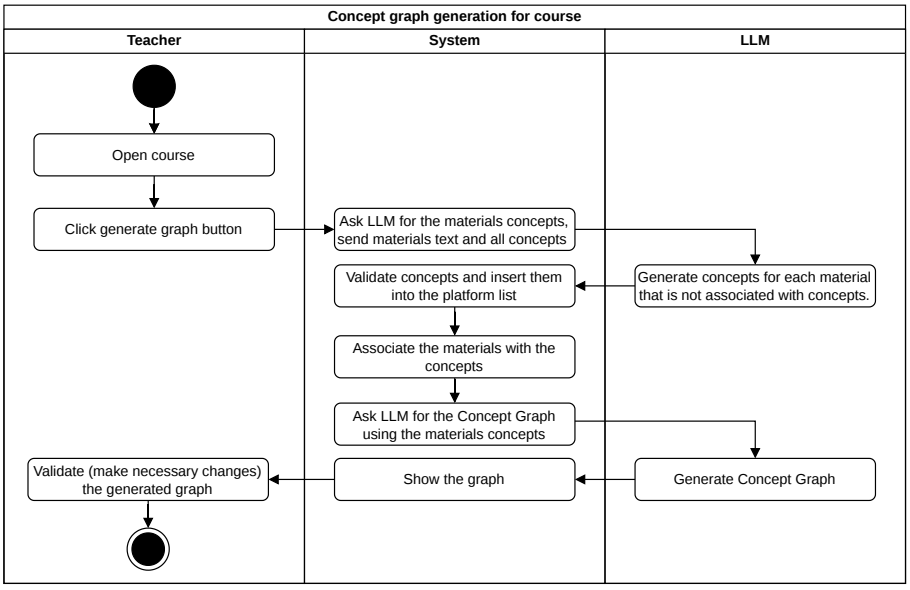

Figure 6 Graph generation activity diagram.

3.4 Feedback Generation

Generative Artificial Intelligence is leveraged to provide feedback during programming exercises. During an exercise attempt, the system analyzes the student's code alongside the problem description and expected solution. This information is provided to the LLM through a prompt, that includes rules such as providing feedback about only one bug, not providing the answer or any code and limiting the response to two sentences at most.

The system generates hints that encourage incremental progress without revealing the answer, as described by the prompt rules.

The generated feedback is delivered through the Agni interface. By integrating feedback generation, the system enhances the learning experience by providing personalized assistance that mimics the support of a human tutor.

3.5 AI Application Interfaces

The system integrates multiple AI APIs to support its functionalities. These APIs provide access to large language models used for concept extraction, feedback generation and other intelligent features. The integrated APIs are Iaedu, Groq and Gemini.

Iaedu[1] is a service from Fundação para a Ciência e a Tecnologia (FCT) developed by the digital services of FCT that aims to provide students, teachers and researchers access to different large language models; at the time of this work only the GPT-4o model is available through the API. The main advantage of using Iaedu is the absence of strict usage limits, making it suitable for applications with high token consumption.

Groq[2] is a low cost service that contains plenty of GenAI models to choose from. In this work, the Groq API is used to access language models, specifically a 120-billion-parameter model (GPT 120B). One of the main advantages of using Groq lies in its optimized inference performance. The platform is designed to deliver responses significantly faster than traditional GPU-based solutions The use of the Groq API involves considerations similar to other AI services, like usage limits

Gemini[3] is an LLM, developed by Google, designed to support tasks involving natural language understanding and multimodal reasoning. Using the Gemini API involves costs associated with the selected model and the volume of data processed, typically measured in tokens.

The Strapi data model was changed to accommodate these new services. Tutors can configure their own API keys on each supported API.

4 Validation

The evaluation is a fundamental component of this work, as it aims to assess the effectiveness and reliability of the proposed GenAI functionalities within the learning environment. It was conducted through a series of targeted questionnaires using Google Forms, each addressing a specific component of the system. These components include concept identification, concept graph generation, failed concept detection and feedback generation.

This section is divided into two parts. First, the methodology subsection describes the design of the evaluation process, including the questionnaires, participant profiles, and the metrics used to assess different system components. Second, the results and discussion

[1] https://iaedu.pt
[2] https://groq.com
[3] https://gemini.google.com

subsection presents and analyzes the findings obtained from these evaluations, covering concept identification, concept validation, concept graph validation, failed concept detection, and feedback rating.

4.1 Methodology

The evaluation methodology was designed to assess the quality of AI-generated outputs by their alignment with human judgment and the agreement between humans when identifying concepts. Multiple questionnaires were created, each corresponding to specific system components. Two of the questionnaires addressed concept identification. A third questionnaire focused on concept graph validation. The fourth one covered failed concept detection and feedback generation.

The respondents to the questionnaires consisted of 6 individuals, 4 male and 2 female, ages ranging from 20 to 30, with backgrounds in computer science and related fields, including a student enrolled in the Master in Computer Science Program of FCUP, one enrolled in the Master in Information Security Program of FCUP, another enrolled in the Master in Artificial Intelligence Program of FCUP and FEUP, a researcher at INESC TEC with a Master's degree in Computer Science from FCUP, a student in the Informatics Engineering Degree at ISTEC, and a software developer holding a degree in Informatics Engineering from ISEP.

Inter-annotator agreement is the degree to which different individuals provide consistent judgments when evaluating the same data. It was measured for concept identification and concept validation using Gwet's AC1 metric [7]. The computations were performed using the irrCAC library in Python. Gwet's AC1 produces a coefficient ranging from -1 to 1, where higher values indicate stronger agreement among annotators. A value of 1 represents perfect agreement, 0 indicates random chance agreement and -1 represents perfect disagreement. Also, for concept graph edges and concepts validation, the mean of the responses was measured to calculate how much the participants agree with the AI generated structures on average.

In the concept identification task, participants and the three AI APIs were asked to analyze two selected programming learning materials and manually extract the concepts being taught. The two materials are from a Python course[4], the first has the title "Branching, Iteration" and the second is titled "Tuples, Lists, Aliasing, Mutability, Cloning".

Subsequently, a second questionnaire was constructed by aggregating all concepts identified by the participants and the ones Gemini extracted from the materials. Iaedu and Groq were not used when building this questionnaire because Iaedu was not reliably working when it was made and Groq wasn't as tested in the Agni platform yet. In this phase, participants were asked to validate each concept and indicate their agreement using a binary (agree/disagree) response.

Concept graph validation required participants to evaluate whether they agreed with the AI generated relationships between concepts. Each relationship (edge) was presented individually, and participants indicated agreement using a binary (agree/disagree) response.

To evaluate failed concept detection, participants assessed student solutions to two programming exercises, the first is about string compression and the second about finding the max number in an array, and were asked to identify which concepts led to failure from a list of concepts, this was then compared to the concepts diagnosed by the system.

[4] Introduction to Computer Science and Programming in Python

Finally, in the feedback generation task, participants reviewed wrong student submissions to the same two programming exercises and evaluated the generated hint on a scale from 1 to 5.

4.2 Results and Discussion

With the results from the concept identification questionnaire, inter annotator agreement was calculated from the responses, the agreement between 6 humans without the AI responses was 26.6% for material 1 and 52% for material 2, while with the AI participants added it was 26.8% for material 1 and 19% for material 2. This shows that when participants have to identify the concepts without having to choose from a predefined list they tend to list different concept names. Also, the calculated agreement with AI as another participant can be on par with the humans as seen with the calculated agreement for all annotators and the agreement between annotator pairs illustrated by the heat map in Figure 7 for material 1, showing that AI and humans can have similar agreement. However, the annotator agreement varied greatly according to the analyzed material, material 1 had almost equal annotator agreement between humans and AI, while for material 2 it was the opposite. Even with this discrepancy, in the validation step the participants showed agreement with the AI generated concepts, which might mean the discrepancy comes from differences in generalization or naming of the knowledge and not disagreement.

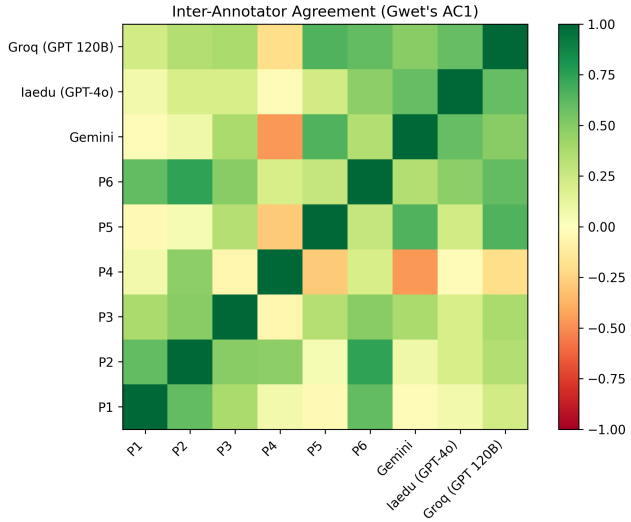

Figure 7 Inter-annotator agreement between pairs for identification (Material 1).

Having the concept validation responses for two materials, annotator agreement between participants and average agreement with AI concepts was calculated. For material 1, the annotator agreement was 75% and the average agreement with AI generated concepts was 83%. While for material 2 the annotator agreement was 63% and AI concepts average agreement was 69%. The agreement between annotator pairs for material 1 is illustrated in Figure 8. These results show good participant agreement with each other in both materials and high agreement from the participants with the AI generated concepts. Supporting the idea that GenAI can extract explicitly taught concepts from learning materials. This also shows that if given a predefined list of concepts, participants will have higher agreement with each other, most likely explained by differences in generalization or naming of the concepts, as illustrated in the contrast between Figures 7 and 8.

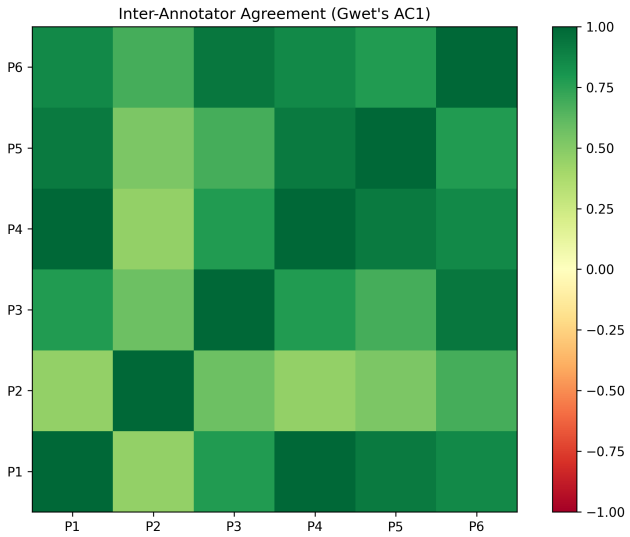

Figure 8 Inter-annotator agreement between pairs for validation (Material 1).

On the topic of the concept graph validation. Overall, the agreement with AI generated edges was on average 78%. Indicating a strong alignment with the AI generated relationships from the participants. Suggesting that the proposed approach is effective in capturing meaningful conceptual dependencies between programming concepts.

For failed concept detection, agreement between participants and the system was evaluated across two programming exercises. In Exercise 1, the system identified String Manipulation and Expressions as the failed concepts. All participants agreed with String Manipulation, while only one selected Expressions, and another identified Variables as an additional failed concept. In Exercise 2, Comparison was identified as the failed concept, with all participants in agreement, although one participant also selected Expressions. These results suggest that the system is generally capable of identifying the underlying concepts responsible for student errors, although some discrepancies may arise in some cases.

Feedback generation was evaluated based on participant ratings of the generated hints for two exercise attempts. Both hints received an average rating of 4.83 out of 5. Overall, the generated feedback was perceived as useful, indicating that the system can provide meaningful guidance to students.

5 Conclusion and Future Work

This paper presented an approach for integrating GenAI into ITS to automate the construction of domain knowledge and feedback mechanisms. One of the main contributions of this work is the exploration of whether GenAI can effectively support tasks such as concept extraction, concept graph generation and feedback provision, which have traditionally relied on manually engineered components. Another contribution is the developed system which is available online[5].

[5] `https://agni.dcc.fc.up.pt`

Across all evaluated components, the results indicate that the proposed approach produces outputs that are largely consistent with human judgment. While disagreements were observed, these are expected and limited by the teachers keeping full control of the domain. Together, these findings support the viability of the approach in assisting the teacher with domain modeling and feedback generation in programming education.

In future work, further improvements will be made to the quality and robustness of AI generated outputs, particularly through the refinement of the prompts and the prompting techniques. The evaluation will also be extended with larger-scale user studies to better assess the effectiveness of the tasks. Further research may also explore the integration of more advanced student modeling approaches.

References

1 Gökhan Akçapınar and Esra Sidan. Ai chatbots in programming education: Guiding success or encouraging plagiarism. *Discover Artificial Intelligence*, 4:87, 2024. doi:10.1007/s44163-024-00203-7.

2 John Anderson, Albert Corbett, Kenneth Koedinger, and Ray Pelletier. Cognitive tutors: Lessons learned. *Journal of the Learning Sciences*, 4:167–207, April 1995. doi:10.1207/s15327809jls0402_2.

3 Yannik Bauer, José Paulo Leal, and Ricardo Queirós. Improving teacher‘s user experience in a virtual learning environment. Master’s thesis, Universidade do Porto, 2023.

4 Albert T. Corbett and John R. Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4(4):253–278, 1994. doi:10.1007/BF01099821.

5 Jacob Doughty, Zipiao Wan, Anishka Bompelli, Jubahed Qayum, Taozhi Wang, Juran Zhang, Yujia Zheng, Aidan Doyle, Pragnya Sridhar, Arav Agarwal, Christopher Bogart, Eric Keylor, Can Kultur, Jaromir Savelka, and Majd Sakr. A comparative study of ai-generated (gpt-4) and human-crafted mcqs in programming education. In *Proceedings of the 26th Australasian Computing Education Conference*, ACE 2024, pages 114–123. ACM, January 2024. doi:10.1145/3636243.3636256.

6 Andre Gutierrez, Paul Denny, and Andrew Luxton-Reilly. Evaluating automatically generated contextualised programming exercises. In *Proceedings of the 2024 ACM Technical Symposium on Computer Science Education*, pages 289–295, March 2024. doi:10.1145/3626252.3630863.

7 Kilem Gwet. *Handbook of inter-rater reliability: The definitive guide to measuring the extent of agreement among raters*. Advanced Analytics, LLC, January 2012.

8 James Kulik and J. D. Fletcher. Effectiveness of intelligent tutoring systems: A meta-analytic review. *Review of Educational Research*, 86, April 2015. doi:10.3102/0034654315581420.

9 Yulia Kumar, Anjana Manikandan, J. Jenny Li, and Patricia Morreale. Optimizing large language models for auto-generation of programming quizzes. In *2024 IEEE Integrated STEM Education Conference (ISEC)*, pages 1–5, 2024. doi:10.1109/ISEC61299.2024.10665141.

10 Wei Ma, Olusola O. Adesope, John C. Nesbit, and Qiang Liu. Intelligent tutoring systems and learning outcomes: A meta-analysis. *Journal of Educational Psychology*, 106(4):901–918, 2014. doi:10.1037/a0037123.

11 Subhankar Maity and Aniket Deroy. Generative ai and its impact on personalized intelligent tutoring systems, 2024. doi:10.48550/arXiv.2410.10650.

12 Moritz Mueller, Corinna List, and Michael Kipp. The power of context: An llm-based programming tutor with focused and proactive feedback. *Proceedings of the 6th European Conference on Software Engineering Education*, 2025.

13 Maciej Pankiewicz and Ryan Baker. *Enhancing Student Focus and Problem-Solving with Real-Time LLM Feedback on Compiler Errors*, pages 412–426. Springer-Verlag, September 2025. doi:10.1007/978-3-032-03870-8_28.

14 Tung Phung, Mengyan Wu, Heeryung Choi, Gustavo Soares, Sumit Gulwani, Adish Singla, and Christopher Brooks. Bridging gaps between student and expert evaluations of ai-generated programming hints, 2025. doi:10.48550/arXiv.2509.03269.

15 Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications, 2025. doi:10.48550/arXiv.2402.07927.

16 Pramuditha Suraweera, Antonija Mitrovic, and Brent Martin. Widening the knowledge acquisition bottleneck for constraint-based tutors. *I. J. Artificial Intelligence in Education*, 20:137–173, January 2010. doi:10.3233/JAI-2010-0005.

17 KURT VanLEHN. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46(4):197–221, 2011. doi:10.1080/00461520.2011.611369.

18 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. arXiv:1706.03762.

19 Beverly Woolf. *Building Intelligent Interactive Tutors, Student-Centered Strategies for Revolutionizing E-Learning*. Elsevier and Morgan Kaufmann, January 2008.

20 Xin Zhou, Zhuoxu Zhang, Xike Xie, and Jiawei Zhang. Deep learning based knowledge tracing in intelligent tutoring systems. *Scientific Reports*, 15, July 2025. doi:10.1038/s41598-025-07422-7.