Exploiting Generative AI to Scale Up Intelligent Tutoring Systems

Miguel Soares Ferreira
Master in Computer Science
Department of Computer Science
Faculty of Sciences of the University of Porto
2026

Exploiting Generative AI to Scale Up Intelligent Tutoring Systems

Miguel Soares Ferreira

Dissertation carried out as part of the Master in Computer Science

Department of Computer Science

2026

Supervisor
José Paulo Leal, Assistant Professor, DCC - FCUP

Acknowledgements

I would like to express my sincere gratitude to my supervisor, José Paulo Leal. His guidance has been fundamental from the very beginning of this research. Through his thoughtful feedback and constant availability, he helped me surpass the challenges in this dissertation and learn from him.

I am grateful for the opportunity I was given to carry out this research. Being trusted with this project allowed me to explore a very discussed topic in current times with a lot of possibilities. This experience allowed me to develop new technical and personal skills.

My sincere appreciation also goes to the Department of Computer Science of the Faculty of Science of the University of Porto. The resources, support, and atmosphere created within the faculty played an essential role in making this work possible and in shaping my overall academic experience.

I would like to extend my gratitude to all participants who took part in the questionnaires for this dissertation. Their willingness to contribute their time and opinions was essential to the evaluation of the proposed system. Their responses supported the validation of the work and improved the depth of the research with their different perspectives. Without their input, this work would not have achieved the same level of completeness.

Finally, I would like to thank everyone who influenced and supported me along the way.

Resumo

A investigação descrita nesta dissertação explora a integração da Inteligência Artificial Generativa (GenAI) em Sistemas Tutores Inteligentes (ITS), com o objetivo de melhorar o ensino da programação. Especificamente, estende o Agni, uma plataforma web de aprendizagem de programação, através da adição de funcionalidades ITS e da automatização da construção de componentes-chave dos ITS que são tradicionalmente desenvolvidos de forma manual. A metodologia recorre a Modelos de Linguagem de Grande Dimensão (LLMs) para extrair conceitos de programação a partir de materiais educativos, mapear as suas relações através de grafos de conceitos e diagnosticar lacunas específicas no conhecimento dos estudantes. Adicionalmente, a GenAI fornece dicas contextuais em tempo real durante exercícios de programação, simulando o apoio personalizado tipicamente oferecido por um tutor humano. O trabalho avalia se a GenAI pode substituir ou complementar eficazmente a criação manual de modelos de domínio, preservando os benefícios pedagógicos dos ITS. Os resultados obtidos sugerem o potencial da combinação de IA com ITS para melhorar a escalabilidade e reduzir a carga de trabalho manual.

Palavras-chave: Sistemas Tutores Inteligentes, Modelos de Linguagem de Grande Dimensão

Abstract

The research described in this dissertation 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 adding ITS features and 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 knowledge 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.

Keywords: Intelligent Tutoring Systems, Large language Models

Table of Contents

List of Figures

List of Abbreviations

AI Artificial Intelligence. 4– 6, 8, 12, 13, 17, 19– 25, 29– 31, 36– 39, 41– 45, 47– 49

APIs Application Interfaces. 6, 17, 20– 22, 29, 38, 39, 41, 47, 49

BKT Bayesian Knowledge Tracing. 4, 9, 10, 15, 23, 24, 27, 35, 38, 39, 46

CMS Content Management System. 18, 19

DOM Document Object Model. 20

GenAI Generative Artificial Intelligence. 1, 3– 6, 8, 11, 12, 14– 16, 21, 39, 40, 43– 48

ITS Intelligent Tutoring Systems. 1– 6, 8– 10, 12, 13, 15, 16, 24, 39, 45– 48

KCs Knowledge Components. 1, 3, 4, 9, 10, 16

LLMs Large Language Models. 1, 3, 11, 13– 15, 20, 21, 27, 44, 46, 47

UI User Interface. 17, 20

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.

The Intelligent Tutoring Systems (ITS) have traditionally addressed this need by providing adaptive feedback and personalized instruction. These systems are capable of modeling domain knowledge, tracking student progress and dynamically adjusting learning paths.

They have been applied successfully across many domains, such as mathematics [1], programming (SQL Tutor) [2], genetics [3] and conceptual modeling (COLLECT-UML) [4].

However, a major limitation of classical ITS lies in their reliance on manually engineered components, such as concept graphs/maps or production rules. In well-structured domains, knowledge can be represented as concept graphs, where nodes correspond to concepts and edges encode prerequisite or dependency relationships between them [5]. In procedural domains, rule-based systems can be used, where problem-solving steps are represented as production rules and student actions are compared against expected solution paths [6]. Alternatively, constraint-based modeling represents domain knowledge as a set of principles that solutions must satisfy, this allows for multiple solution strategies without requiring explicit enumeration of all possible paths [4]. Constructing these components requires significant effort from tutors which limited the deployment of these systems [7].

Recent advances in Generative Artificial Intelligence (GenAI), particularly Large Language Models (LLMs), offer new opportunities to overcome these limitations. Generative models are capable of producing new content with natural language input by learning from large datasets. In the context of programming education, these models can generate code, explain concepts and assist in creating learning materials. More importantly, they introduce the possibility of automatically generating the key components of an ITS, like the Knowledge Components (KCs).

1.1. Context

The main idea behind classical ITS is relatively straightforward. Most classical ITS architectures are composed of four major components: the Domain Model, the Student Model, the Tutoring Model and the Interface Model. Each component performs a specific function that contributes to the overall adaptive behavior of the system.

The Domain Model contains the knowledge that will be taught to the learner. This knowledge is commonly represented using structures such as rules, maps or graphs. The purpose of the domain model is to define the concepts within the subject area and explain how these concepts are related. A simple example can be observed in an ITS designed for basic mathematics. In such a system, the domain model may include concepts such as addition, subtraction, equations, variables and fractions. Where solving equations requires an understanding of variables, while fractions may be necessary before introducing algebraic division. These relationships can be represented through a concept graph that illustrates learning dependencies between topics. For instance, addition may be necessary before learning equations. Such a graph helps the ITS determine the appropriate learning sequence for the student.

Another essential component is the Student Model, which represents what the learner currently knows or does not know. For example, if a student consistently solves multiplication problems correctly but struggles with division exercises, the system may record multiplication as mastered while identifying division as an area of weak understanding. The ITS can then adapt future exercises and instructional content accordingly.

The Tutoring Model is responsible for determining how the system should teach the learner. For example, if a student repeatedly fails to solve a problem, the tutoring model may provide materials that teach the concepts required to solve that problem.

Finally, the Interface Model manages the communication between the student and the ITS. This component includes all interaction mechanisms used by the learner.

1.2. Motivation

Learning to program requires the gradual acquisition of multiple interconnected concepts, where understanding advanced topics often depends on mastering prerequisite knowledge. For example, before a student can correctly implement recursive algorithms, they are generally expected to understand functions, variables, conditional statements and

control flow. When a student fails to fully understand one of these foundational concepts, difficulties may propagate into later topics, negatively affecting the learning process.

The development of ITS aims to provide adaptive support tailored to each student. Early ITS systems focused heavily on mathematics education, where prerequisite relationships between concepts are particularly evident. Systems such as the Cognitive Tutor for mathematics modeled student knowledge through explicit representations of skills and KCs, allowing the system to identify which specific mathematical operations or reasoning steps a student had not yet mastered [1].

Although effective, building these systems traditionally required significant manual effort. Developing the domain knowledge for an ITS often involves identifying all relevant knowledge, defining prerequisite relationships between them and maintaining these structures as course content evolves.

Since LLMs are capable of analyzing educational materials and generating structured outputs, they may assist in the construction of domain models, identification of student difficulties and generation of feedback. Instead of completely replacing teachers, these systems can reduce repetitive work while keeping the teacher in control of the generated knowledge structures.

This motivation led to the development of the work presented in this dissertation, exploring how GenAI can support and scale ITS development within education.

1.3. Goal and Objectives

Thus, the research question that drove this work was the following: **Can an LLM aid a domain expert in the repetitive and labor intensive tasks required by ITS?** By automating this repetitive work, tutors could more easily adopt ITS in real educational environments, making it more scalable.

The central idea of this dissertation is to extend Agni [8], a platform for learning programming described in detail in a later chapter, by leveraging GenAI to automate the construction of ITS components. Instead of relying solely on manually constructed KCs, generative models are used to extract programming concepts from learning materials, then generate relationships between them and support the process of giving feedback to students. This approach aims to automate the manual and repetitive process of creating these components, improving scalability while preserving the pedagogical effectiveness of ITS.

The main objectives of this dissertation 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.

1.4. Approach

The research question that drove the work led to the goal of extending Agni to support ITS. Agni is a web-based playground to learn programming [8]. It will be detailed in Chapter 3.

To achieve these objectives, the development focused on extending the existing Agni platform by integrating ITS features and Artificial Intelligence (AI) functionalities. This dissertation adopts a hybrid approach, where well-established ITS mechanisms, such as KCs based domain modeling and student modeling techniques such as Bayesian Knowledge Tracing (BKT), were preserved, and it builds upon them by automating the knowledge modeling process using GenAI.

First, was the implementation of the component that exploits AI models to analyze programming learning materials and extract the taught concepts. These concepts are then structured into a graph representation, capturing prerequisite relationships between them. This replaces the need for manually constructing KCs, which is traditionally a time-consuming task.

Then, the component that uses AI during student resolution of programming exercises in order to identify the failed concepts and the student modeling with BKT were implemented. These identified concepts are used to update the student model, enabling the system to maintain the student´s knowledge state. Based on this, Agni is able to suggest relevant learning materials associated with the failed concepts, supporting adaptive learning. In addition, a feature where it's possible for students to request AI generated contextual feedback based on their current solution during resolution was added.

To evaluate the effectiveness of the approach, the system components that use AI were evaluated through user studies. From these studies agreement with the AI generated components and inter-annotator agreement was measured to assess the quality of the outputs compared to human annotators.

1.5. Workplan

The dissertation followed a workplan divided into phases aligned with its goals. The process began with a literature review on ITS and GenAI, establishing the foundation and proposed solution for the work.

Following this, the implementation phase focused on extending the Agni platform with the proposed features, including ITS support, concept extraction, concept graph generation, student modeling and feedback generation.

After implementation, an evaluation phase was carried out through user studies to assess the quality of the components and their agreement with human annotators.

With the evaluation phase completed, a paper on the work was written.

Throughout these phases, the writing of the dissertation was conducted in parallel with the development activities.

The timeline of these phases is illustrated in Figure 1.1.

Figure 1.1: Gantt Chart.

1.6. Dissertation Content

The dissertation begins with a review of the relevant literature in Chapter 2. Section 2.1 introduces ITS, including their historical development and core components. Section 2.2 reviews GenAI and its capabilities, while Section 2.3 looks into current research applications in programming education and Virtual Learning Environments.

Chapter 3 presents the technological background of the system. It begins with Agni, the platform used for implementing the features, followed by Strapi, the headless content

management system responsible for the backend, and Vue, the framework used for the frontend development. The chapter also describes the AI Application Interfaces (APIs) integrated into the system, namely Iaedu, Groq and Gemini, and concludes with a summary of these components

Chapter 4 presents the proposed system. It begins with the Adaptive Learning component, describing how student knowledge is modeled, how personalized lesson recommendations are generated and how failed concepts are detected. It then introduces the Concept Extraction process. Next, the chapter details the Concept Graph Generation mechanism. The Feedback Generation component is also presented, where hints are provided during exercises. The chapter concludes with a summary of the system's main components.

Chapter 5 describes the implementation of the proposed system. It details how each component was developed and integrated into the Agni platform, covering both backend and frontend aspects. The chapter begins with the AI APIs. It then presents the implementation of Concept Extraction and Concept Graph Generation. The Adaptive Learning component is also detailed. This is followed by the Feedback Generation component. The chapter then introduces the extensions made to the data model to support these features, and concludes with a summary of the implementation.

Chapter 6 presents the validation of the proposed system. It describes the evaluation methodology used to assess the effectiveness and reliability of the implemented GenAI components through a set of structured questionnaires. The chapter details the participant profiles, evaluation tasks and metrics used. It then presents and discusses the results obtained for concept identification and validation, concept graph generation, failed concept detection and feedback generation. The chapter concludes with a summary of the main findings, highlighting the overall performance of the system and its alignment with human judgment. 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

Chapter 7 concludes the dissertation by summarizing the main contributions and findings of the work. It revisits the objectives of integrating GenAI into ITS and discusses how the proposed system addresses these goals. The chapter highlights the effectiveness of the implemented components, supported by the validation results. It also outlines the main contributions of the dissertation and discusses possible directions for future work.

The work developed in this dissertation resulted in the publication of a scientific paper. Consequently, some sections of this dissertation are based on and adapted from material presented in that publication.

2. State of the Art

This chapter reviews the current state of the research on topics relevant to this dissertation and establishes a foundation for the research and development undertaken. The review was heavily focused on the exploration of the current applications of AI in the area of programming education, the techniques used and its applications on Virtual Learning Environments.

Starting with a review of ITS and its history in Section 2.1, then a look into GenAI and its current state in Section 2.2. Further in the chapter there are the different applications of GenAI in programming education and how other research is applying it to Virtual Learning Environments in Section 2.3. This chapter concludes with a summary of key findings during this research period in Section 2.4.

2.1. Intelligent Tutoring Systems

Intelligent Tutoring Systems 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.

K. Vanlehn [9] describes these systems as computer-based instructional environments that provide immediate, customized instruction and feedback to learners without requiring human intervention. Comprehensive analyses by W. Ma et al. [10], and J. Kulik and J. D. Fletcher, [11] demonstrate that ITS interventions significantly improve learning outcomes and knowledge retention when compared to traditional studying from static texts.

The ITS review will start with a look into the historical foundations and then a description of the core components of an ITS system.

2.1.1 Historical Foundations

Since the earliest days of computers, researchers have strived to develop computer tutors that are as effective as human tutors. The conceptual roots of ITS can be traced to early computer-assisted instruction systems developed in the 1960s and 1970s, such as PLATO, which introduced interactive learning environments and early forms of automated feedback [12].

A significant milestone was the development of cognitive tutors grounded in cognitive psychology and production rule modeling. The work conducted at Carnegie Mellon University led to the creation of the Cognitive Tutor, which operationalized the idea that expert problem-solving knowledge could be modeled computationally [6]. This approach introduced the notion of KCs and tracking student mastery of knowledge.

2.1.2 Core Components

B. Woolf [13] 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, which correspond to skills required to solve a problem [6].

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.

The tutoring model governs instructional decisions. It 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 [6].

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.

2.1.3 Student Modeling

Student modeling has been a central research focus within ITS. One of the most influential approaches is BKT, introduced by A. T. Corbett and J. R. Anderson [14]. 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.

Bayesian Knowledge Tracing 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, which uses recurrent neural networks to capture complex learning patterns, while it as shown effectiveness, it struggles with smaller datasets [15].

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.

2.1.4 Knowledge Representation

Knowledge representation has always been central to adaptive systems. For example, in programming, KCs might include skills such as conditional statements or variables. In this 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 [7].

Programming knowledge is highly structured, conceptually layered, and dependent on prerequisite mastery (e.g., variables -> control flow -> functions -> recursion). Concept graphs are well suited for modeling such domains.

2.2. Generative Artificial Intelligence

Generative Artificial Intelligence refers to a class of machine learning systems capable of producing new content such as text, code, images or audio based on patterns learned from large datasets.

Recent advances in deep learning have significantly improved the capabilities of generative models. In particular, the development of large-scale transformer architectures has enabled models to process and generate natural language with high levels of fluency and contextual understanding. These systems, commonly referred to as LLMs, are trained on a massive corpora of text and can perform a wide variety of tasks such as question answering, summarization and explanation generation.

Because programming education relies heavily on natural language explanations and iterative feedback, GenAI is particularly well-suited for integration into adaptive learning systems. These models can support learners by generating explanations, assisting with problem-solving and providing feedback during practice.

However, GenAI systems also present limitations. Because they rely on statistical patterns rather than explicit reasoning structures, their outputs may contain inaccuracies or hallucinated information [16]. Therefore ensuring reliability remains an important research challenge when integrating these systems into learning environments [16].

Ethical considerations also play a significant role in the use of GenAI. Because the content it generates is limited to the data it was trained on, the information it generates may not be up-to-date, or it may contain bias, in fact it has been linked to having gender, racial or cultural biases, as well as many more, on the other hand, plagiarism is a prevalent issue among university students, and large language models can exacerbate this problem [16].

To address some of these limitations, several techniques have been proposed to improve the reliability of GenAI systems. One common approach is prompt engineering, this refers to the design and structuring of input prompts in a way that guides the model toward producing more accurate responses. For example, zero-shot prompting is when the model is only given a task description and has to rely on its pre-trained knowledge to generate an answer [17]. In contrast, few-shot prompting provides examples of input–output pairs within the prompt, helping the model better understand the expected format and reasoning process [17].

More advanced techniques include chain-of-thought prompting, which encourages the model to generate intermediate reasoning steps before producing a final answer, improving performance on complex tasks [17]. Similarly, self-consistency prompting involves generating multiple responses and selecting the most consistent answer increasing robustness [17].

In addition, prompt design often incorporates strategies such as specifying output formats, constraining responses and providing contextual information to reduce ambiguity. These techniques are particularly important in educational settings, where the correctness and pedagogical usefulness of generated feedback are critical. As a result, prompt engineering plays a central role in enabling GenAI systems to be effectively integrated into ITS.

2.3. Applications of Generative AI in Programming Education

Recent literature highlights several uses of GenAI in programming education. Such as, generating explanations, producing code examples, answering questions and providing contextual feedback, all of which are central activities in the learning process.

This section reviews current research on the use of GenAI in programming education.

2.3.1 Feedback and Explanations

One of the possible applications is the generation of personalized feedback. Providing timely and personalized feedback to large numbers of students is a long-standing challenge in programming courses. Providing individualized feedback is difficult due to large class sizes and limited time for all of the students. ITS have historically attempted to address this problem.

Because GenAI can interpret both problem statements and student solutions, they should be able to produce contextual feedback that can resemble human explanations.

Diving into the current research on this application, M. Kwak et al. [18] developed an ITS that used GenAI to generate adaptive learning materials and feedback, they found that current AI capabilities allow the generation of detailed feedback for the student. T. Phung et al. [19] tested feedback generation by asking GPT-4 to generate a hint and explanation of the hint on a buggy program, and then compared the results to a real tutor with metrics like if the provided hint was useful for the student to resolve the bug, they found that on a scale of 0-100 for these metrics the tutor had a 92.0 performance while GPT-4 had a 66.0 performance, while there is still a gap, it also shows us that AI can generate quality hints a lot of the time.

P. Denny et al. [20] deployed an LLM-powered digital assistant in a programming course and collected student feedback, their results highlight students value such tools for their ability to provide instant, engaging support, particularly during peak times such as before assessment deadlines. Students reported wanting to understand the content and felt that

explanations that focused on process (how) and foundational knowledge (why) were key features, they did not want to be directly provided with the solutions to the problems.

Feedback on compile error was also an explored application in research. M. Pankiewicz and R. Baker [21] evaluated the impact of GPT-4o at generating real-time feedback for compiler errors. In this study, GPT-4o was used to automate feedback generation for non-compiling submissions and a survey was performed after. Results showed that students who received feedback from LLMs submitted fewer non-compiling code attempts. According to the survey responses, a significant majority of students (81%) rated the usefulness of feedback positively and only 8% did not find it beneficial.

T. Phung et al. [22] studied mismatches in perceived hint quality from students and experts perspectives based on the deployment of AI-generated hints in a Python programming course, with the intent of proving better hints, some of their suggestions are incorporating student history to the hints and also when a student is far from the solution proposing a plan to solve the problem.

M. Mueller et al. [23] introduced a LLM-based ITS where the code submission history is used when generating feedback for the student. They then compared feedback with submission history and without, the results showed that users preferred the version with the submission history 69% of the time.

Despite these possibilities, challenges remain regarding accuracy and reliability, these models may occasionally produce incorrect explanations or misleading information, which can negatively impact learners.

2.3.2 Exercise Generation

Another application is the generation of programming exercises and quizzes. Creating programming exercises takes time and effort from tutors, using AI to perform this task instead could be a time saver.

Artificial Intelligence has shown that it can automatically generate learning materials based on the learning materials provided by the tutor, and preliminary evidence suggests that the use of various versions of learning materials leads to an increase in students study time, especially for students who have not mastered the topic otherwise [24].

J. Doughty et al. [25] analyzed the capability of GPT-4 to produce multiple-choice questions aligned with specific learning objectives from Python programming classes in higher

education. They developed an LLM-powered (GPT-4) system for generation of these questions from high-level course context and module-level objectives, and found that GPT-4 was capable of producing them with clear language, a single correct choice, high-quality distractors, and well-aligned with the learning objectives.

A possible type of exercise generation is personalized contexts, previous research has suggested that the context used in the problem description for the exercises is likely to impact student engagement and motivation [26]. Furthermore, using a broad range of contexts or even allowing students to do the personalization of the exercises themselves, may help with a more diverse student population. A. Del Carpio Gutierrez et al. [26] used GPT-4 to generate 500 contextualized programming exercises and found that the quality of the generated exercises was very high.

Another possibility is the student generating revision exercises themselves for a topic. Y.-F. Lin et al. [27] introduced Athena, a GenAI programming mentor constructed to guide learners to think critically, the revision tab allows students to generate revision exercises on a topic. Most students gave a positive response, saying that their motivation to keep learning had increased.

E. K et al. [28] conducted several experimental approaches to generating programming exercises. They tried generating exercises with a determined topic and context, and also generating exercises with different contexts from an original exercise, they found that this was the one with better results and students approved of the generated exercises.

On the topic of quizzes, Y. Kumar et al. [29] analyzed the use of LLMs like ChatGPT in creating quizzes for Java programming courses. This research compares quizzes made by LLMs against human-created content to assess their quality, the error rate was around 10% which means the LLMs were generally good at quizz generation.

Overall, AI-driven exercise generation has the potential to reduce instructor workload while increasing the availability and variety of practice materials for students.

2.3.3 Other Applications

Beyond feedback and exercise generation, GenAI is being explored in several additional areas within programming education and virtual learning environments.

One other application is the generation of interactive concept maps. In this approach, GenAI was used to automatically create hierarchical concept maps based on inputs such

as course descriptions, topic lists, and other metadata associated with the discipline [30].

These maps organize knowledge into structured representations that support exploration and understanding. Importantly, this process incorporates a human-in-the-loop stage. Instructors review, modify and refine the generated concept maps to ensure correctness and pedagogical alignment. This step is essential to address potential inaccuracies in AI-generated content and to incorporate domain expertise. Once validated, the concept map is approved for use within the learning environment.

The resulting concept maps are used exclusively for visualization and navigation purposes. Through hierarchical drill-down interaction, students can explore topics from high-level concepts to more detailed subtopics enabling a clearer understanding of relationships within the domain.

2.4. Summary

This chapter reviewed the state of the art on ITS, GenAI and their applications in programming education.

Intelligent Tutoring Systems were presented as adaptive learning environments designed to approximate the benefits of one-on-one human tutoring.

Historically, systems such as PLATO and later Cognitive Tutors demonstrated that modeling expert knowledge and tracking student learning at the level of knowledge components can significantly improve learning outcomes.

The classical ITS architecture, composed of domain, student, tutoring, and interface models, remains a foundational framework for adaptive educational systems. In particular, student modeling techniques such as BKT provide an efficient and well-established mechanism for estimating learner mastery and supporting adaptive decision-making. However, a key limitation identified is the reliance on manually constructed domain models and knowledge representations, such as concept graphs, which are labor-intensive and difficult to scale.

The chapter then explored GenAI and its recent advances. LLMs, enabled by transformer architectures, have demonstrated strong capabilities in natural language understanding and generation, making them suitable for tasks such as feedback generation. These characteristics align well with the needs of programming education, where feedback and

conceptual explanations are essential. Nevertheless, generative models present important limitations, including hallucinations and ethical concerns such as bias and academic integrity issues. Techniques such as prompt engineering have been proposed to mitigate these challenges, with approaches like zero-shot, few-shot, and chain-of-thought prompting improving output quality and reliability.

The review of applications of GenAI in programming education highlighted several promising directions. The generation of feedback and explanations addresses a long-standing challenge in large-scale education: providing timely and personalized support to students. While traditional ITS rely on rule-based mechanisms, recent studies show that generative models can produce useful and engaging feedback, although still not matching the performance of human tutors in all cases. Students value these systems for their immediacy, particularly in high-demand situations such as assignment deadlines, and their use has shown positive effects in areas such as compiler error feedback.

Another important application is the automatic generation of exercises and learning materials. GenAI can reduce instructor workload while enabling the creation exercises with personalized contexts. Research shows that AI-generated materials can be aligned with learning objectives and maintain high quality, particularly in structured formats such as multiple-choice questions. Additionally, the use of contextualized exercises has been linked to increased student motivation.

Beyond these applications, GenAI has also been explored for tasks such as the automatic construction of concept maps. These approaches aim to reduce the manual effort required to build domain models, although they often still rely on human validation to ensure correctness and pedagogical alignment.

Overall, the literature reveals a clear opportunity at the intersection of ITS and GenAI. While ITS provide robust pedagogical frameworks and adaptive mechanisms, they are limited by the cost of building the KCs. Conversely, GenAI offers scalable content generation but lacks reliability and structured reasoning.

Having reviewed the research associated with ITS, GenAI and their applications in education, the next chapter shifts focus to the platforms and tools used to implement the proposed approach.

3. Background

This chapter presents the main systems employed in this dissertation. Each system is examined in detail, outlining its purpose, key features, advantages and possible limitations, in order to clarify its role within the overall context of the work. The chapter begins with Agni, a platform for learning programming in Section 3.1. This is followed by Strapi, a headless content management system in Subsection 3.1.1 and Vue, a framework for User Interface (UI) development Subsection 3.1.2. After that, Section 3.2 explains the AI APIs used in the work. Finally, the chapter concludes with a Summary of the discussed topics in Section 3.3.

3.1. Agni

Agni is a web-based platform designed for learning programming. The frontend was developed using Vue 2 and consists of two main interfaces: a student interface as illustrated in Figure 3.1 and a teacher interface seen in Figure 3.2. The teacher interface allows instructors to manage course content, including lessons, materials and related information. In contrast, the student interface provides an interactive learning environment, enabling learners to access instructional materials such as PDFs, as well as engage in practical activities like quizzes and programming exercises.

The underlying data structure of the platform before the work changes is illustrated in Appendix A. This structure is centered around the concept of a Course, which is composed of multiple Modules, each containing a sequence of Lessons. Lessons are further divided into two primary types of content: Expositive materials, which deliver instructional content, and Evaluative components, which assess student understanding.

Expositive content includes learning resources such as PDFs and videos. Evaluative components consist of quizzes or programming exercises. Quizzes are made up of questions and possible answers, while programming exercises include problem descriptions, contextual information and associated test cases used to validate solutions.

The system also models the interaction between students and course content through entities such as Class and Student. A class represents a group of students enrolled in a specific occurrence.

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

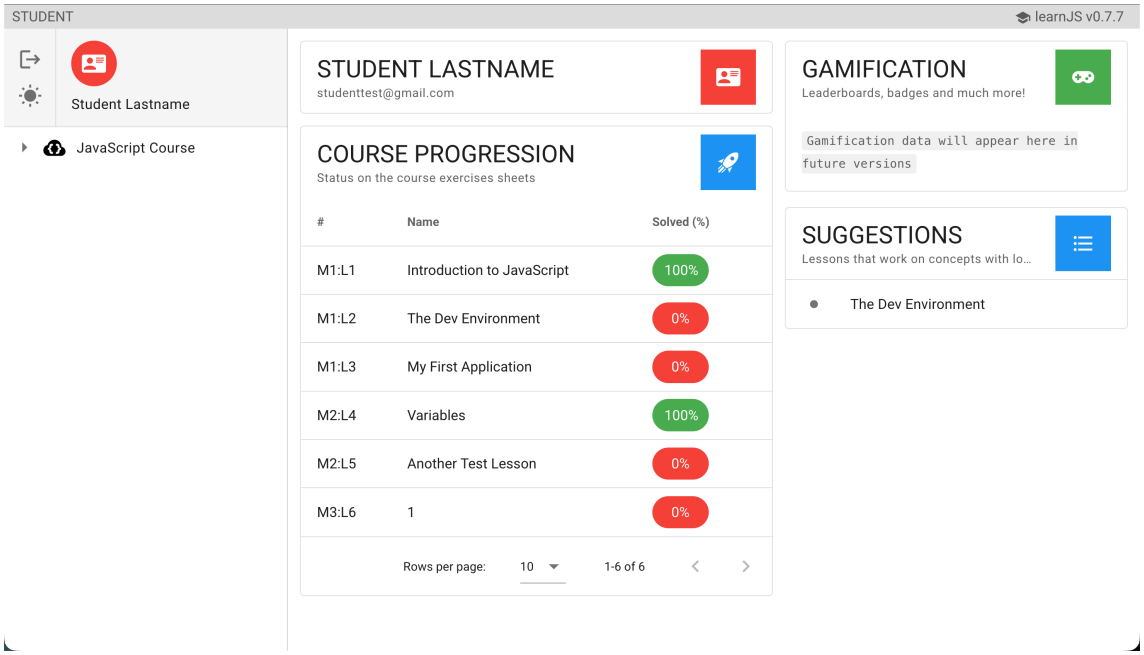

Figure 3.1: Student interface.

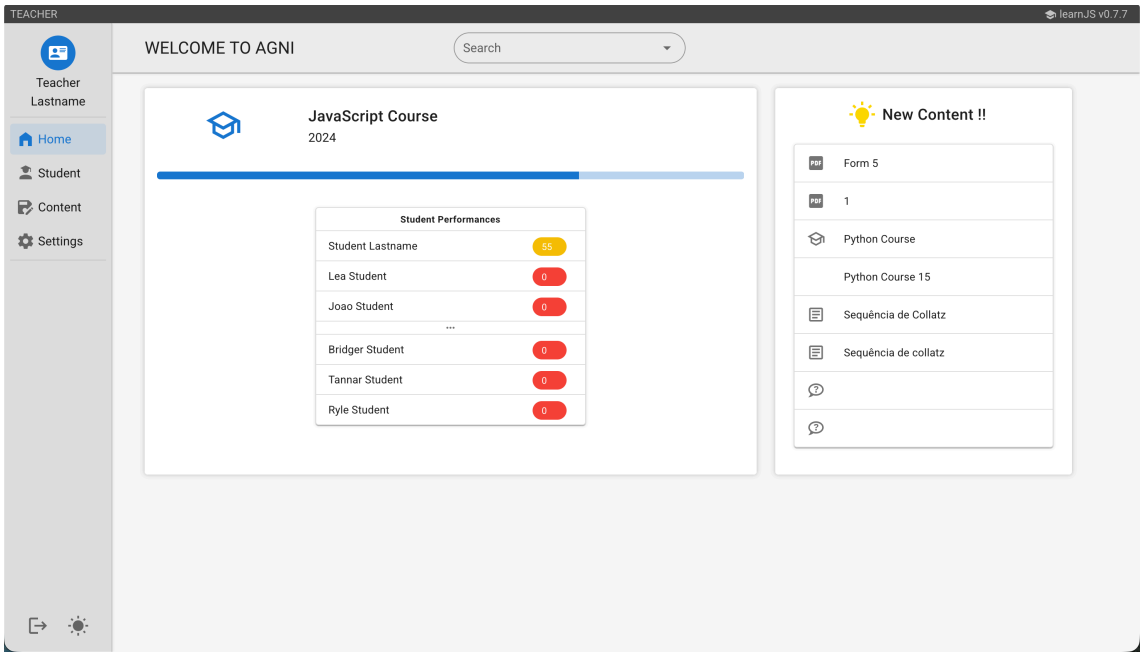

Figure 3.2: Teacher interface.

3.1.1 Strapi

Strapi is a headless Content Management System (CMS). While traditional CMS software allows users to create and modify digital content on websites without extensive programming knowledge. It typically tightly connects the frontend, where content is displayed,

with the backend, where content is managed and stored. This coupling can limit flexibility, especially when updating the frontend or using the content across various platforms. In contrast, as suggested by its name, a headless CMS focuses exclusively on the backend. The frontend is handled independently. The backend usually incorporates an API to interface with the frontend. This separation allows changes to the frontend without disrupting the core content. Moreover, it facilitates content integration across multiple platforms via the API.

Built on NodeJS, Strapi is an open-source platform with a user-friendly interface. This interface facilitates the construction of content structures and content creation. Users can define fields for various content types, including Collection types, Single types, and Components. Collection and Single types represent content structures with an API endpoint. These can be created and managed independently.

On the other hand, Components are reusable structures that can be used within a Collection or a Single type. The REST API supports the common CRUD operations, create, read, update and delete, with filtering through various parameters.

To further enhance its capabilities, Strapi allows for customizing API responses and actions through hooks. These include Collection hooks, which modify response behavior, and Webhooks, which can trigger subsequent requests based on predefined criteria. However, such configurations necessitate manual coding.

Its open-source nature encourages the community to develop and integrate additional features, expanding its capabilities. Strapi defaults to SQLite for data access, allowing switching to more scalable databases such as PostgreSQL, MySQL or MariaSQL. Additionally, Strapi facilitates the integration of unit tests to ensure system reliability.

In this dissertation, Strapi is used as the backend system. Its data model was extended and new services were implemented to support the integration of AI functionalities.

3.1.2 Vue

Vue is a component based JavaScript framework used for building user interfaces and single-page applications. It was created by Evan You, a former Google engineer, in 2014, and has since become widely adopted within the development community.

One of Vue's main advantages is its approachable learning curve, allowing developers to

quickly develop simple applications. Its clear structure and comprehensive documentation make it suitable for both beginners and experienced developers. Like other modern frameworks such as React and Angular, Vue follows a component-based architecture. This approach enables developers to create small, self-contained and reusable components that can be combined to form complex user interfaces. Each component encapsulates its own state, template and styling, which improves maintainability and testability in large applications.

Vue also provides a powerful directive system that allows developers to declaratively apply behavior to the Document Object Model (DOM). For instance, the v-if directive enables conditional rendering, v-for allows iteration over lists and v-model facilitates two-way data binding in form inputs.

In terms of performance, Vue leverages a virtual DOM. When the application state changes, Vue updates a virtual representation of the DOM, compares it with the actual DOM, and applies only the necessary changes. This update mechanism significantly enhances performance.

Although Vue offers many built-in features, additional functionality is often achieved through external libraries. Commonly used extensions include Vuex, Vue Router and Vuetify. Vuex provides centralized state management, ensuring consistent handling of shared data across components. Vue Router enables navigation between different views without reloading the page. Vuetify is a Material Design framework that offers a wide range of prebuilt UI components, like navigation bars, footers, icons and buttons, helping developers create responsive and visually consistent interfaces.

Vue was used to implement the UI of the new features within the Agni frontend, enabling interaction between users and the AI components of the system.

3.2. AI Application Interfaces

The system integrates multiple AI APIs to support its functionalities. These APIs provide access to the LLMs used for concept extraction, graph generation, feedback generation and failed concept detection. The integrated APIs are Iaedu, Groq and Gemini.

3.2.1 Iaedu

Iaedu* 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 LLMs, with the goal of improving productivity and avoiding inequality in academic contexts. However, at the time of this work, only the GPT-4o model is available through API.

The main advantage of using Iaedu is the absence of strict usage limits, making it suitable for applications with high token consumption. This characteristic makes it a practical choice for experimentation and continuous use within the Agni platform.

3.2.2 Groq

Groq† 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, reducing latency during interactions.

The use of the Groq API involves considerations similar to other AI services, like usage limits. Requests are typically measured in tokens, and system performance may depend on factors such as request frequency and model size.

3.2.3 Gemini

Gemini‡ is a LLM, developed by Google, designed to support tasks involving natural language understanding and multimodal reasoning. Similar to other LLM APIs, Gemini enables applications to generate contextually relevant responses based on input prompts.

Using the Gemini API involves costs associated with the selected model and the volume of data processed, typically measured in tokens. The service also enforces rate limits on the number of requests and the amount of data that can be processed within a given time frame.

*https://iaedu.pt
†https://groq.com
‡https://gemini.google.com

3.3. Summary

This chapter presented the key systems underlying this dissertation. Starting with Agni, the platform to support programming education and where the proposed AI features were implemented. Strapi was introduced as the backend system responsible for content management and API services, while Vue was described as the framework used for the user interface.

Finally, the AI APIs were presented, including Iaedu, Groq and Gemini. These services provide the generative capabilities that enable the intelligent features of the system, such as concept extraction, feedback generation and adaptive support. Together, these components form the technological foundation for the work developed in this dissertation.

After outlining the foundational systems in this chapter, the basis is set to explain the proposed system, starting with the system design in Chapter 4.

4. Proposed System

Following the review of the state of the art and the technologies relevant to this dissertation, this chapter presents the proposed system. The chapter begins with Adaptive Learning in Section 4.1, describing how student knowledge is modeled using BKT and how recommendations are generated. This is followed by Concept Extraction in Section 4.2, which details the process of identifying programming concepts from educational materials. Section 4.3 introduces Concept Graph Generation, explaining how relationships between concepts are established. After that, Section 4.4 presents the Feedback Generation component, which provides AI generated hints during exercises. Finally, the chapter concludes with a Summary of the discussed topics in Section 4.5.

4.1. Adaptive Learning

To track student knowledge, the system uses BKT, enabling a probabilistic representation of student mastery of individual concepts.

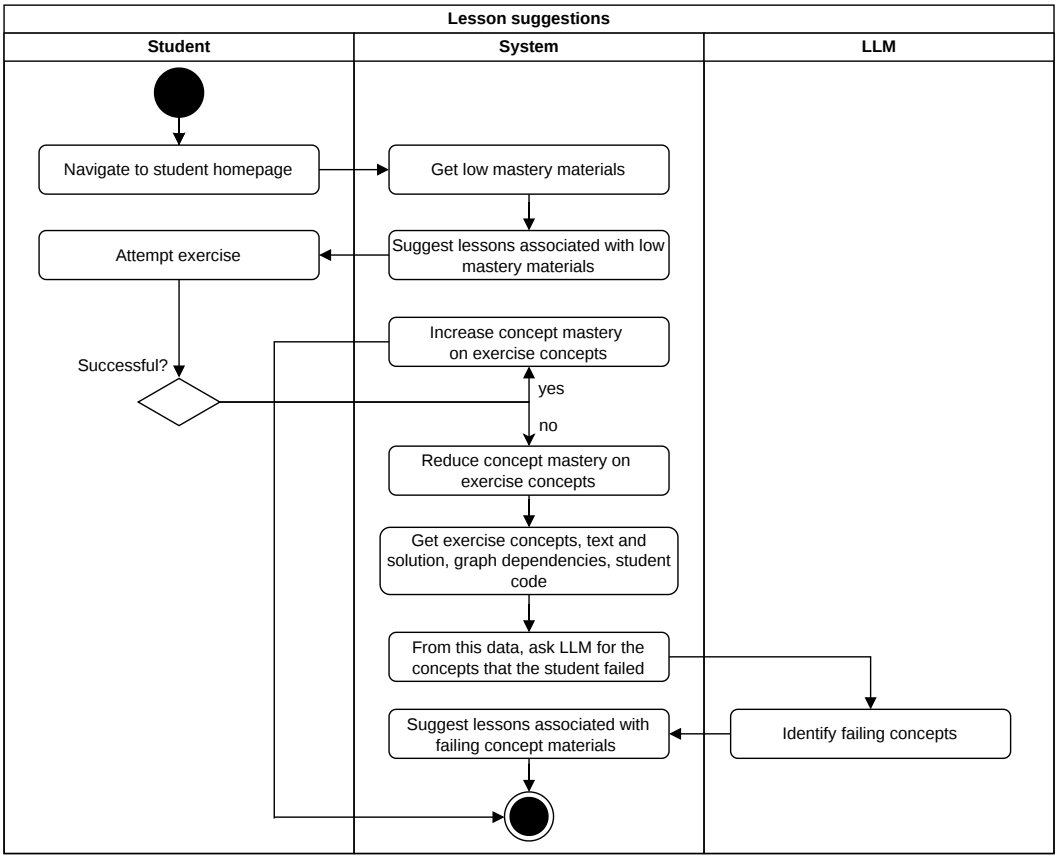

Figure 4.1: Lesson suggestions activity diagram.

As illustrated in Figure 4.1, 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. 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.

4.2. Concept Extraction

A key component of the system is the automatic extraction of programming concepts from educational materials. As illustrated in Figure 4.2, this process involves the teacher, the system and a LLM.

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.

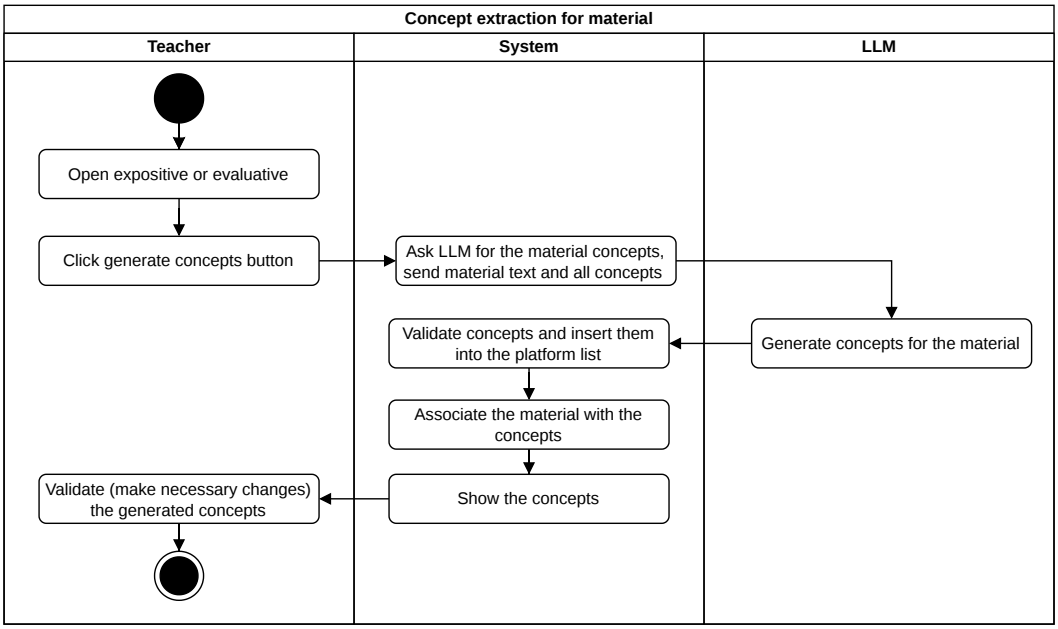

Figure 4.2: Concept extraction activity diagram.

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.

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 approach balances AI results with user control.

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.

4.3. 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 4.3, this process involves the teacher, the system and a LLM. The teacher provides high-level validation, while the LLM assists in identifying and structuring relationships between concepts.

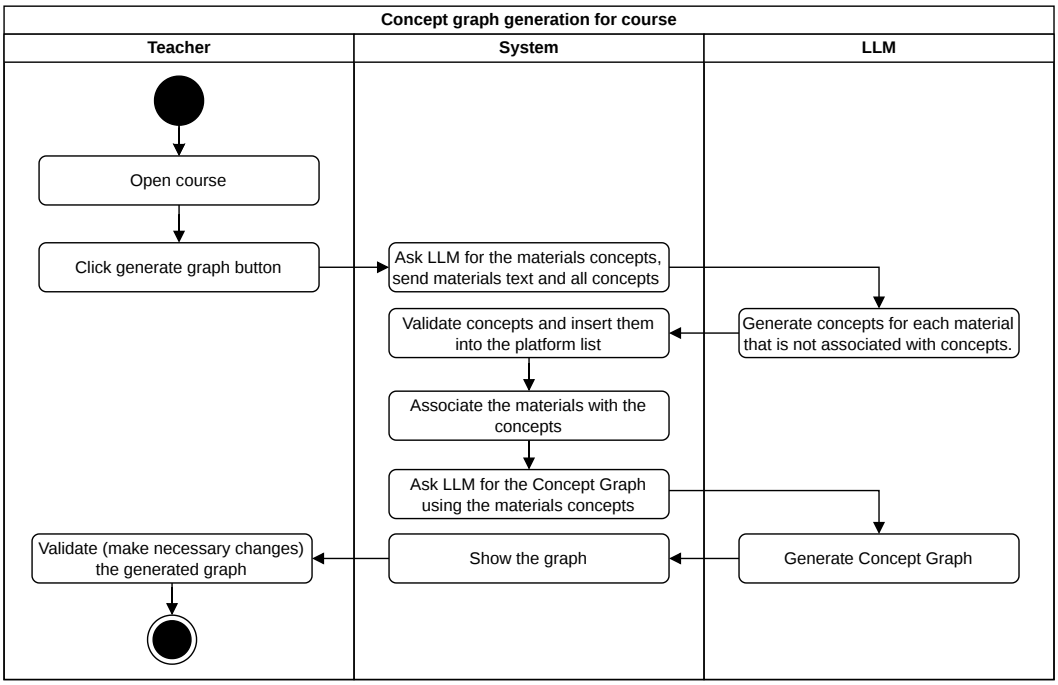

Figure 4.3: Graph generation activity diagram.

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 4.2.

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.

The generated graph is then presented to the teacher through Agni. 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.

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.

4.4. 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. 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.

4.5. Summary

This chapter presented the proposed system, focusing on its core components and how they interact.

The system leverages BKT to model student knowledge and provide lesson recommendations based on exercise attempt outcomes or concept mastery levels.

It incorporates an automated concept extraction process using LLMs, allowing educational materials to be tagged with the explicitly taught programming concepts while keeping the teacher in control.

Building on these concepts, the system can automatically generate concept graphs that represent the prerequisite knowledge relationships seen in programming courses.

Additionally, the generated contextual feedback provides hints during exercises without revealing solutions, simulating human tutor feedback.

After outlining the system in this chapter, the basis is set to explain the implementation in Chapter 5.

5. Implementation

After the high level overview of the proposed components, this chapter explains the implementation process of each component. It details how the system was developed and integrated into the Agni platform, covering both backend and frontend aspects.

The chapter begins with the AI APIs in Section 5.1, describing the implementation of the layer responsible for handling communication with multiple AI providers. This is followed by Concept Extraction in Section 5.2, detailing how programming concepts are automatically identified and managed within the system. Section 5.3 presents the Concept Graph Generation component, explaining how relationships between concepts are generated and visualized. Section 5.4 describes the Adaptive Learning mechanisms, including student modeling and recommendation logic. Section 5.5 introduces the Feedback Generation component, which provides hints during exercise resolution. Section 5.6 describes the Data Model extensions required to support these features. Finally, the chapter concludes with a summary of the implemented components in Section 5.7.

5.1. AI Application Interfaces

A new AI manager service was created on the Strapi backend to centralize all the AI related functionality. This service is responsible for handling communication with external AI providers, managing prompt building and provider selection.

Initially, the component was developed with support for a single provider (Iaedu). During the integration of Iaedu, an issue was encountered when requesting structured outputs in JSON format. The response generated by the model were frequently wrapped in Markdown code block delimiters like, “‘json“‘, which prevented direct parsing using JSON parsers. A preprocessing step was introduced to sanitize the response before parsing, this fixed the problem. This issue highlights the importance of defensive processing when working with LLM outputs, as models may not strictly adhere to formatting constraints.

However, relying on a single API introduces limitations in terms of flexibility and availability. To address this, an abstraction layer was introduced, allowing the system to support multiple AI providers in a uniform way. This design ensures that the system is decoupled from any specific provider and can dynamically switch between them based on the configuration defined by the teacher.

Each provider (Iaedu, Groq, and Gemini) is implemented as a separate class adhering to a shared interface, responsible for handling request formatting and response parsing. All providers extend a common `BaseProvider` class. Listing 5.1 shows an example implementation for the Google provider.

```
class GoogleProvider extends BaseProvider {
  async generate(prompt) {
    const { GoogleGenAI } = await import('@google/genai');
    const ai = new GoogleGenAI({ apiKey: this.apiKey });

    const response = await ai.models.generateContent({
      model: this.model,
      contents: prompt,
    });

    return this.safeParse(response.text);
  }
}
```

Listing 5.1: Google provider implementation

In this implementation, the `generate` method encapsulates all provider-specific logic, including request construction and response handling. The use of dynamic import ensures that the Google SDK is only loaded when required, reducing unnecessary overhead. The method `safeParse`, defined in the base class, is responsible for validating and parsing the model output.

When a request is made, the AI manager dynamically selects the appropriate provider based on user configuration stored in the database. This includes the selected provider, model and corresponding API key (`aiProvider`, `aiModel` and `aiApiKey` fields in the User collection).

It is responsible for constructing and sending requests to the selected AI provider, as well as handling the responses. The abstraction layer ensures that all providers can be used interchangeably without affecting downstream components.

Upon receiving a response, the service performs a normalization step to convert the output into a consistent internal format. By standardizing the output, the system ensures that the components can consume AI responses without being aware of the underlying provider.

Having centralized the AI interactions within a single service, the system becomes more scalable. New providers can be integrated with minimal changes to the existing codebase, and updates to API communication logic are isolated from the rest of the platform.

5.2. Concept Extraction

The implementation of the concept extraction feature required changes to both the frontend and backend of Agni.

On the frontend side, using Vue, several components were developed to support concept management and visualization. In the content tab of the teacher interface, a new `Concept` table was introduced, displaying all concepts currently stored in the platform database. This table allows teachers to create, edit and delete concepts, providing full control over the concept repository.

To manage the association between concepts and learning materials, a reusable `Concepts` component was implemented. This component displays the concepts associated with a given material as tags. It allows teachers to add concepts from the global list and remove them individually through the interface. This ensures that teachers remain in control of the concept assignment process, even when AI generated concepts are used.

Additionally, a Generate Concepts button was integrated into both Expositive and Evaluative pages. This triggers the automatic concept extraction process. Communication with the backend is handled through HTTP requests implemented using Axios.

On the backend, new endpoints were created to support concept management, including retrieving, creating, updating and deleting concepts. Furthermore, a dedicated endpoint was implemented to extract concepts from a given material.

When this endpoint is invoked, the system retrieves the material text, the list of existing platform concepts, and the AI configuration (provider, model and API key). The material text is extracted using the pdf-extraction JavaScript library. This information is used to construct structured prompts that are sent to the GenAI.

Initially, the extraction process followed a single-step pipeline where within a single request the LLM had to identify the concepts in the material text and then align them with the existing platform vocabulary. This resulted in the extracted concept quality being less consistent, with more hallucinations and concepts that represent the same knowledge. Because of this, the process was divided into multiple prompts. The prompts are illustrated in Appendix B.

After these changes, the extraction process now follows a multi-step pipeline. First, candidate concepts are extracted from the material. Then, a normalization step merges similar

concepts into a consistent representation. Finally, the concepts are aligned with the existing platform vocabulary to avoid duplication and ensure consistency.

To reduce variability in the generated output, the entire process can be executed five times through a user selection on the frontend. The resulting concepts are counted, and only those that appear at least three times are retained. This majority-based filtering improves the robustness of the results.

After receiving the concepts, the system performs a validation step to ensure alignment with the existing concept repository. This includes matching generated concepts with previously defined ones and avoiding duplicates. The validated concepts are then associated with the corresponding material and stored in the database.

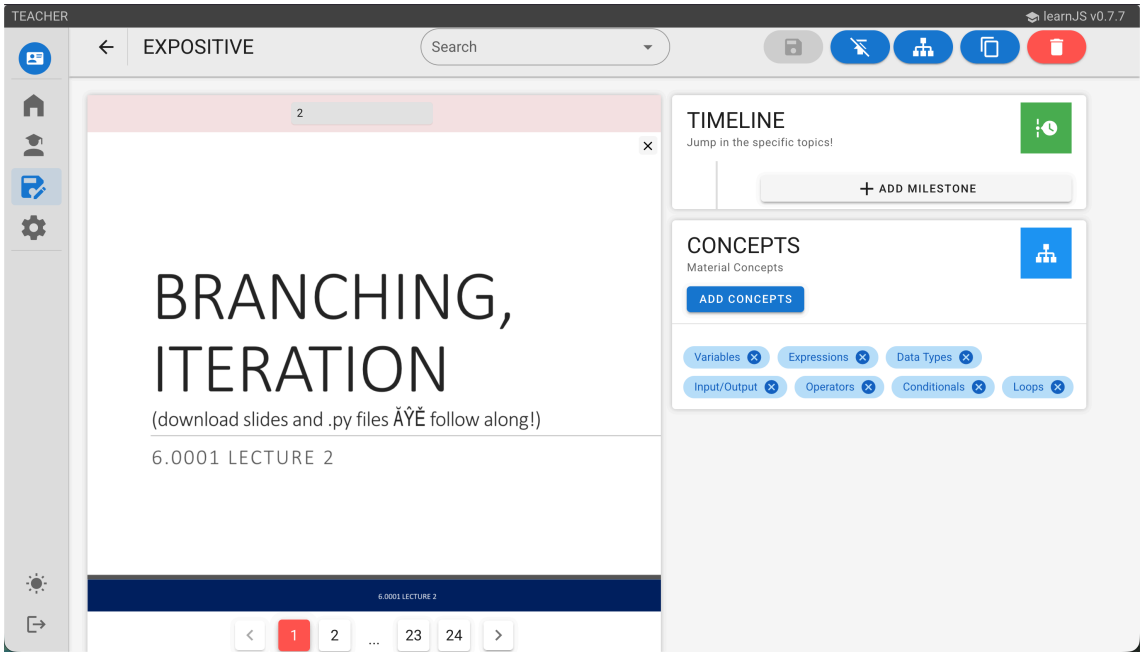

Figure 5.1: Extracted Concepts.

Finally, the resulting concepts are returned to the frontend, where they are displayed to the teacher for review, illustrated in Figure 5.1. The teacher can then refine the list by adding or removing concepts, maintaining a human-in-the-loop approach.

5.3. Concept Graph Generation

The implementation of the concept graph generation feature required modifications to both the frontend and backend of Agni.

On the frontend, a ConceptGraph component was developed using Vue and the vis-network* JavaScript library for graph visualization. This component displays the concept dependencies associated with a course in the form of a directed graph, this can be seen in Figure 5.2. It allows teachers to manually add or remove concepts (nodes) and define or delete prerequisite relationships (edges), ensuring full control over the generated structure.

Additionally, a Generate Concept Graph button was integrated into the Course page. This button triggers the automatic graph generation process.

On the backend, a dedicated endpoint was implemented to handle graph generation requests. When invoked, the system first retrieves all concepts of the materials associated with the selected course. For materials that do not yet have associated concepts, the concept extraction process is executed for each material.

Once all materials are associated with concepts, the system aggregates the complete set of concepts for the course. The generation of the concept graph is driven by a structured prompt (Listing 5.2) that encodes the notion of prerequisite relationships between concepts. Specifically, a dependency is defined as a relationship where the understanding of one concept is required before another can be meaningfully learned.

```
You are building a COURSE-SPECIFIC concept dependency graph.

DEFINITION:
Dependency means:
"Understanding concept A is REQUIRED before concept B can be meaningfully learned."

RULES:
- Concepts are already validated.
- Use ONLY the concepts provided.
- Do NOT invent new concepts.
- Output directed dependencies.
- If concept A must be learned before B, output A -> B.
- If B requires understanding multiple concepts, output multiple edges.
- If the concept doesnt have any dependencies, it doesnt need an edge.
- BUT most of the time concepts WILL have dependencies.
- The goal is to connect the graph, WHEN POSSIBLE.
- Avoid unconnected graphs.
- Some concepts may depend from several concepts.

CONCEPTS:
<<<
${JSON.stringify(conceptLabels, null, 2)}
```

*https://www.npmjs.com/package/vis-network

```
>>>

OUTPUT FORMAT (JSON ONLY):
{
  "edges": [
    { "from": "A", "to": "B" },
    { "from": "E", "to": "B" },
    { "from": "B", "to": "C" },
    { "from": "B", "to": "D" }
  ]
}

Output ONLY JSON, no markdown, no commentary
```

Listing 5.2: Concept graph generation prompt

The prompt enforces strict constraints to ensure that the generated graph remains consistent with the course domain. In particular, it restricts the model to using only the provided concepts and requires the output to follow a structured JSON format representing directed edges.

Additionally, the prompt encourages the generation of connected graphs whenever possible, improving the usefulness of the resulting structure for modeling learning progression. This was added after previous results were showing unconnected graphs, which would not be useful for the intended system.

The prompt is forwarded to the selected LLM. The model then returns a set of relationships representing prerequisite dependencies between concepts. These relationships are then parsed and transformed into a graph structure, where nodes correspond to concepts and edges represent dependencies.

After generation, the graph is stored in the database and returned to the frontend for visualization, illustrated in Figure 5.2. The teacher can then review and modify the graph as needed, maintaining a human-in-the-loop approach.

This implementation reduces the manual effort required to construct concept graphs while preserving teacher control and ensuring consistency across the domain model.

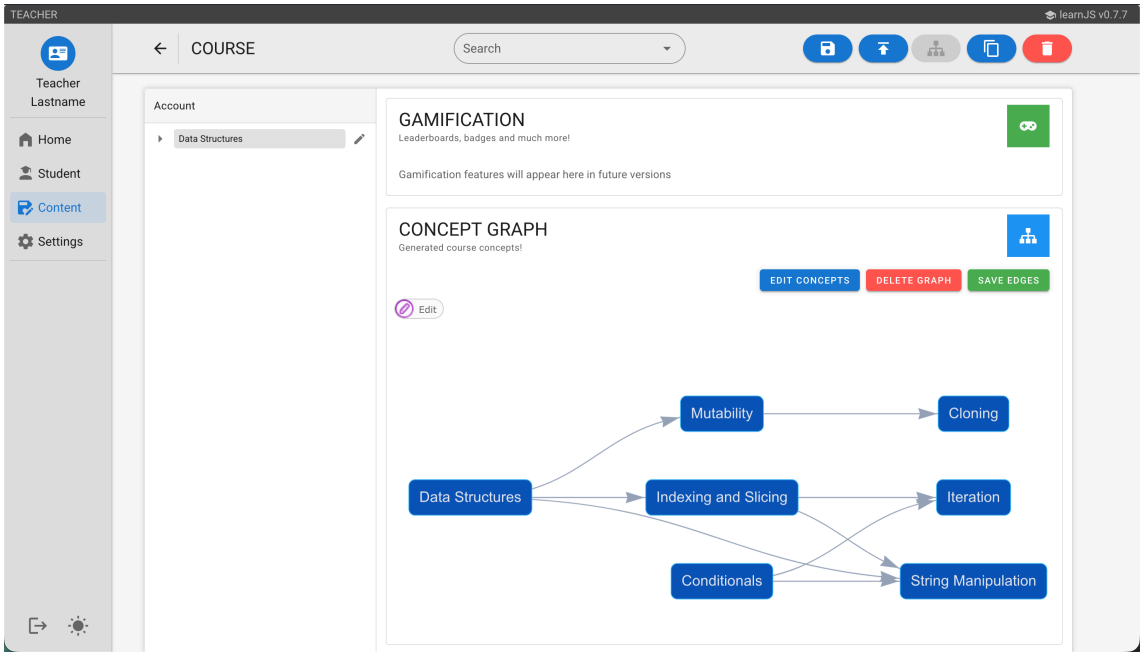

Figure 5.2: Generated Graph.

5.4. Adaptive Learning

The implementation of the adaptive learning component integrates student modeling, failure analysis and recommendation mechanisms into the Agni platform.

On the frontend, a suggestions panel, seen in Figure 5.3, was implemented where students can view recommended materials based on their performance and estimated current knowledge state.

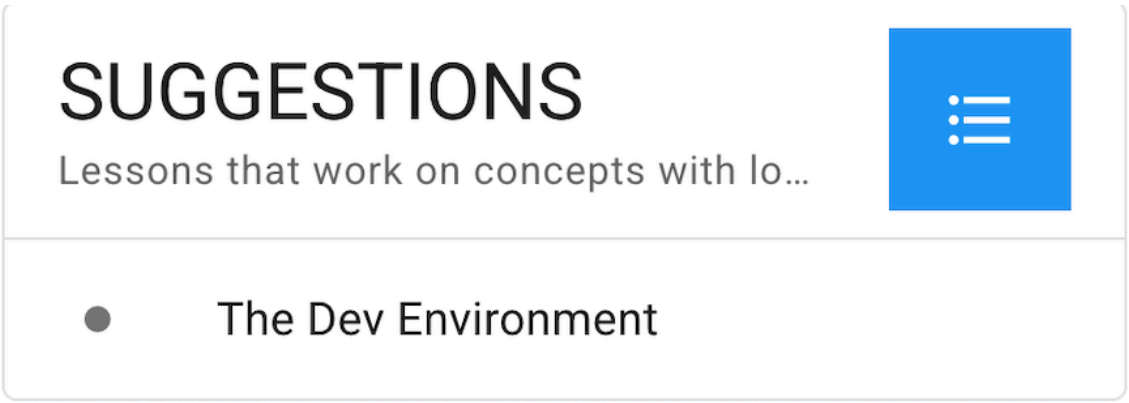

Figure 5.3: Suggestions Panel.

On the backend, BKT was implemented to model student knowledge. Each concept is associated with a mastery probability that represents the likelihood that a student has understood that concept.

Whenever a student submits an evaluative exercise, their performance is used to update the mastery of all associated concepts. The update process is implemented in the `concept-mastery` controller, as shown in Listing 5.3.

```
function updateBKT(oldMastery, isCorrect) {
  let posterior;

  if (isCorrect) {
    posterior =
      (oldMastery * (1 - S)) /
      (oldMastery * (1 - S) + (1 - oldMastery) * G);
  } else {
    posterior =
      (oldMastery * S) /
      (oldMastery * S + (1 - oldMastery) * (1 - G));
  }

  return posterior + (1 - posterior) * T;
}
```

Listing 5.3: BKT mastery update implementation

The parameters $P(L_0)$, T, G, and S represent the initial mastery, learning rate, guess probability, and slip probability, respectively. This probabilistic model allows the system to continuously refine its estimation of student knowledge over time.

Based on the estimated mastery levels, the system identifies concepts where the student exhibits low proficiency. A threshold-based approach is used to filter concepts with mastery below a predefined value obtained from literature, in this case 0.85.

Candidate lessons are then selected by matching their associated concepts with the identified low-mastery concepts. Each lesson is scored according to its relevance, giving higher weight to lessons that address weaker concepts. Lessons are then ranked by score and the top 10 results are presented to the student.

In addition to mastery-based recommendations, the system incorporates a failure analysis mechanism. When a student fails an exercise, their submitted code is analyzed using the AI manager service, with the prompt illustrated in Appendix C, to identify the most likely underlying conceptual misunderstandings.

This process considers:

- The exercise description

- The associated exercise concepts
- The student's code
- The reference solution
- The course concept graph

The AI model returns a set of failing concepts, which are then used to identify relevant lessons covering those concepts. This allows the system to provide targeted remediation based on the student's specific mistakes.

The adaptive learning component enables the platform to dynamically adjust learning content based on student performance. Through probabilistic modeling and AI-assisted diagnosis, the system delivers targeted recommendations that address both persistent knowledge gaps and immediate learning difficulties.

5.5. Feedback Generation

The feedback generation component provides students with contextual hints during the resolution of programming exercises, supporting incremental learning without revealing the solution.

On the frontend, this functionality is exposed through a Get Hint button available during an exercise attempt. When triggered, a request is sent to the backend, and the generated hint is displayed in a dedicated panel within the interface. This allows students to request assistance on demand while maintaining focus on the exercise.

On the backend, a dedicated endpoint handles hint generation requests. When invoked, the system retrieves the problem description and the student's current code submission. This information is used to construct a structured prompt in the AI manager service.

The prompt is designed to enforce pedagogical constraints, ensuring that the generated feedback remains helpful without directly providing the solution. In particular, it restricts the response to identifying a single issue, avoids code generation, and encourages a concise and Socratic style of guidance.

Once the response is received, the system extracts the generated hint and returns it to the frontend for display. The student can then use this guidance to refine their solution.

5.6. Data Model

To support the new features, the Strapi data model was extended.

For concept generation, a new Concept collection was created. This collection includes a `label` field representing the concept name. Additionally, a `conceptTags` field was added to both the Expositive and Evaluative collections, these collections represent materials that teach knowledge and exercises to practice the knowledge, respectively. The added field establishes a relation to the Concept collection and is used for associating the created concepts with the material that teaches them.

The data model was further expanded with the introduction of ConceptGraph and ConceptEdge components. The ConceptGraph component includes a `concept` field and an `edges` field, which is a list of ConceptEdge components. Each ConceptEdge represents a graph edge and contains `from` and `to` fields, both defined as relations to the Concept collection. The existing Course collection was updated with a `ConceptGraph` field, enabling each course to be associated with a graph.

To model student knowledge, a `conceptMasteries` field was added to the Student collection. This field is related to a new ConceptMastery collection, which represents a student´s mastery of a given concept. The ConceptMastery collection includes a `mastery field` (expressed as a percentage), as well as `student` and `concept` fields, which are relations to Student and Concept collections, respectively. This structure enables the modeling of student mastery for each concept using BKT.

Finally, to enable the AI features, the User collection was updated with three new fields: `aiProvider`, `aiModel` and `aiApiKey`. These fields store the AI provider, the model and corresponding API key. Only teachers are permitted to configure these fields, and when a student uses an AI feature, the configuration of the teacher who created the course is used.

5.7. Summary

This chapter presented the implementation of the proposed system, detailing how each component was developed and integrated into the Agni platform.

The AI APIs introduced the layer that enables communication with multiple AI providers.

The concept extraction component demonstrated how programming concepts can be automatically identified from educational materials using GenAI, while maintaining teacher control through an interactive interface.

Building on these concepts, the concept graph generation component enabled the automatic creation of prerequisite relationships between concepts, supporting the representation of domain knowledge.

The adaptive learning component leveraged BKT and failure analysis with GenAI to model student knowledge and provide personalized learning recommendations.

Additionally, the feedback generation component provided contextual hints during exercise resolution, guiding students without revealing solutions and promoting active learning.

Finally, the data model was extended to support ITS and these features, enabling the representation of concepts, graph relationships, student mastery and the integration of AI APIs within the platform.

Overall, the implementation successfully integrates GenAI in Agni, resulting in a system capable of ITS features and automating key tasks while preserving user control.

6. Validation

The evaluation is a fundamental component of this work, as it aims to assess the effectiveness 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 Chapter is divided into three sections. First, the methodology section 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 section presents and analyzes the findings obtained from these evaluations, covering concept identification, concept validation, concept graph validation, failed concept detection and feedback rating. The chapter concludes with a summary of the validation results.

6.1. 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 [31]. 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*, 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.

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.

6.2. 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 for material 1 26.6% and 52% for material 2, while with the AI participants

*Introduction to Computer Science and Programming in Python

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 6.1 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.

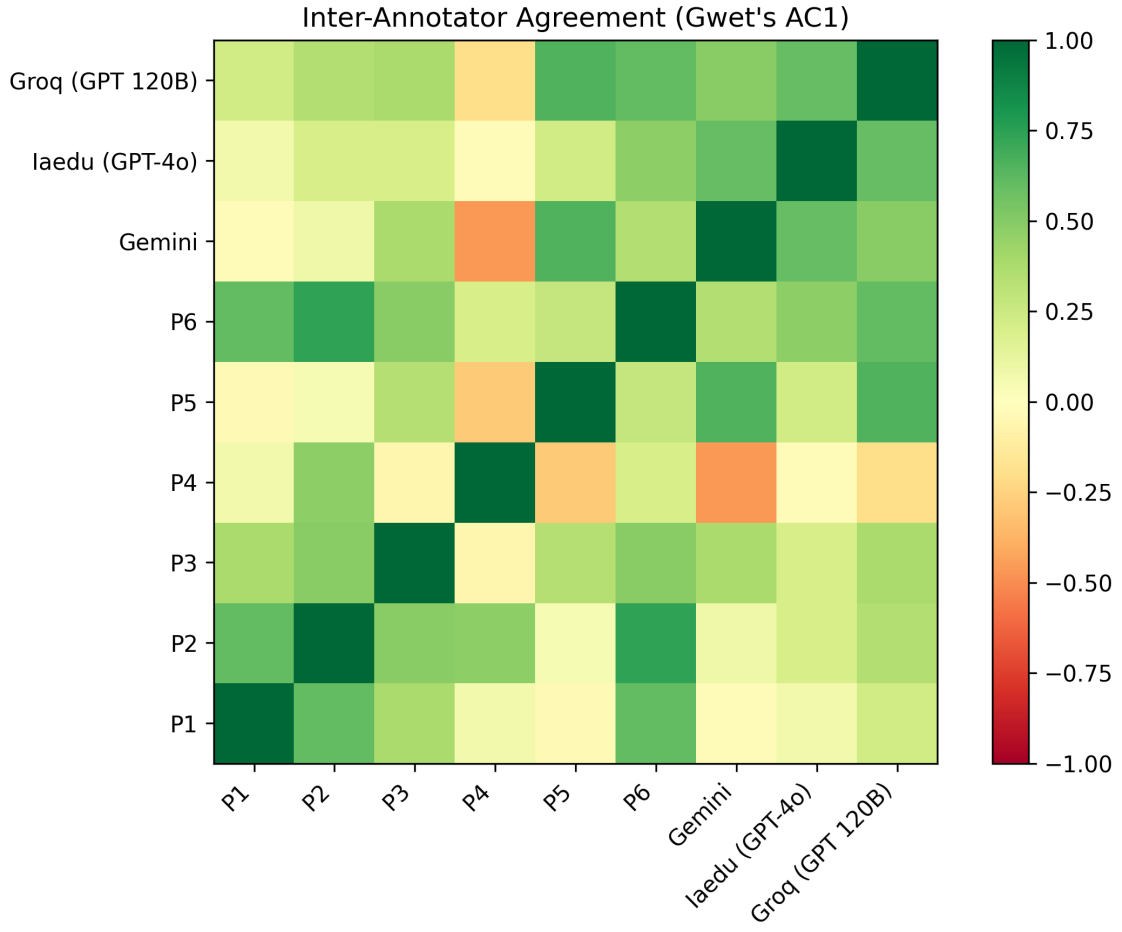

Figure 6.1: Inter-annotator agreement between pairs for identification (Material 1).

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 6.2. 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 6.1 and 6.2.

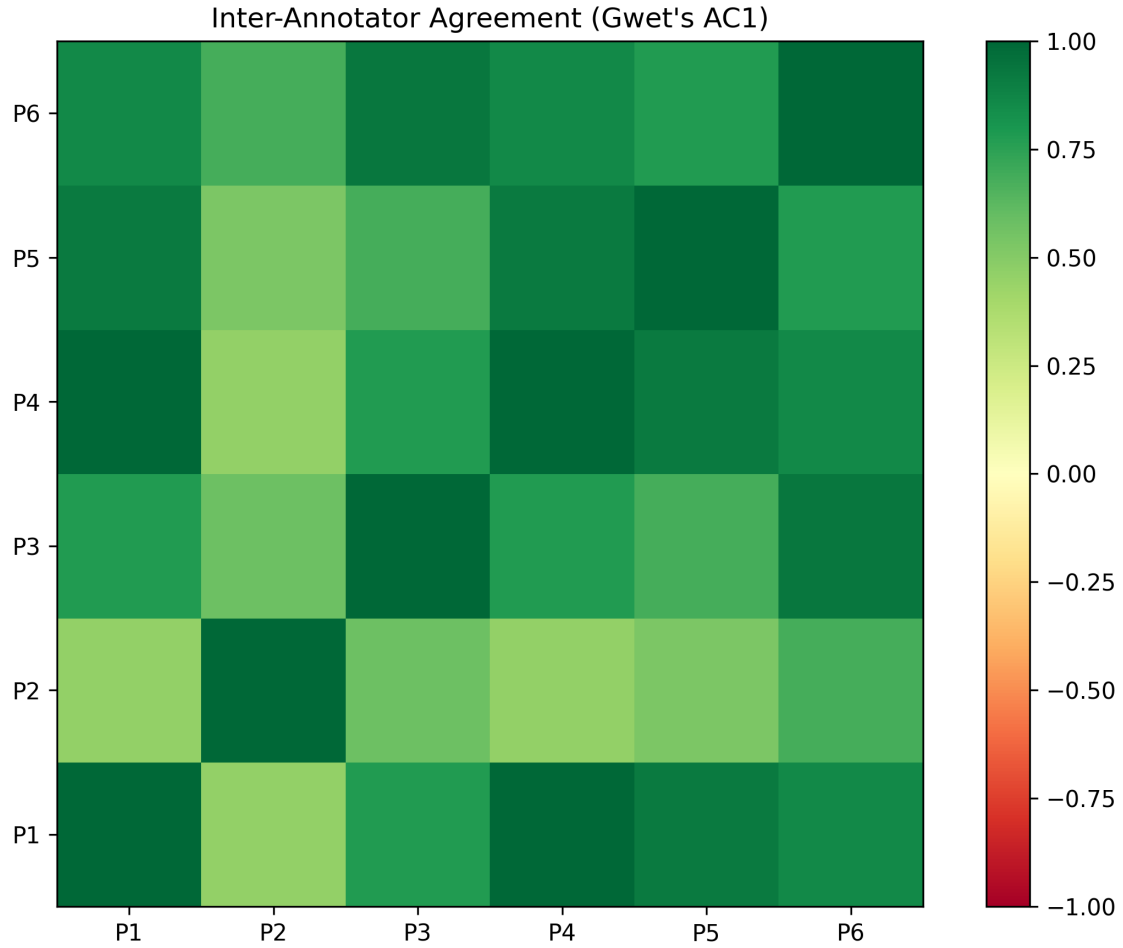

Figure 6.2: Inter-annotator agreement between pairs for validation (Material 1).

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.

6.3. Threats to Validity

Although the validation results indicate that the proposed approach can produce outputs largely aligned with human judgment, several threats to validity must be considered.

One limitation of this evaluation is the relatively small number of participants involved in the questionnaires. While the participants had backgrounds related to computer science and programming, a larger and more diverse sample could provide more robust and generalizable results.

Another limitation is that the evaluation focused primarily on comparing AI generated outputs with human judgment rather than evaluating the willingness and acceptance of students and teachers to use the proposed system in real educational scenarios. Due to time constraints, it was not possible to conduct a long-term study involving students and teachers actively interacting with the platform in educational settings. Such studies would provide stronger evidence regarding usability, trust in the generated outputs and the practical impact of the system on student engagement and teacher workload.

Another threat is related to the non-deterministic nature of GenAI systems. Since LLMs may generate different outputs for the same input, the obtained results can vary depending on the execution. Although techniques such as prompt engineering and majority voting were employed to reduce variability, complete consistency cannot be guaranteed.

Finally, while the proposed system keeps the teacher in the loop and allows manual validation of generated outputs, the quality of the generated concepts, graphs and feedback

still depends on the capabilities and limitations of the selected AI models. Improvements or changes in future models may affect the behavior and performance of the system.

6.4. Summary

This chapter presented the evaluation of the proposed system, focusing on assessing the quality of the GenAI components through questionnaires.

Overall, the results suggest that the proposed approach is capable of generating outputs that are largely aligned with human judgment across the evaluated components. The strongest results were observed in concept validation, concept graph generation and feedback generation, where participants showed high levels of agreement with the AI-generated outputs. Although lower agreement values were observed in the open-ended concept identification task, this appears to be mainly related to differences in concept naming and abstraction levels rather than disagreement about the underlying knowledge itself.

The validation process also highlighted some challenges associated with evaluating GenAI systems in educational contexts. Since AI generated outputs are inherently non-deterministic, meaning that repeated executions may produce different results. These characteristics make evaluation more complex and reinforce the importance of keeping the teacher in the loop to review and refine the generated outputs when necessary.

It is also important to note that the objective of this validation was not to evaluate the effectiveness of ITS themselves, as they have already been extensively studied in the literature. Instead, the focus of this evaluation was on assessing the quality of the AI generated outputs in comparison with human judgment. Nevertheless, evaluating the proposed system in a real educational environment would be an important direction for future work, as it would allow the assessment of how the implemented features in the Agni perform in practice when supporting students and teachers over longer periods of use.

7. Conclusion

This dissertation focused on exploiting GenAI to scale ITS. Specifically, it updated Agni to support ITS features and integrated GenAI to automate the construction of the domain knowledge and feedback mechanisms. The work began with a literature review of ITS in Section 2.1 and GenAI in Section 2.2, 2.3. After that, Chapter 4 presents the proposed system, where the architecture and the ITS and GenAI features are described. Subsequently, Chapter 5 details the implementation process, outlining the thought process and encountered challenges. To validate the effectiveness of the components, questionnaires with the intent of evaluating each specific component were conducted and the results of these questionnaires were discussed in Chapter 6.

Across all evaluated components, the results indicate that the proposed approach can produce 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.

The system integrated multiple components to support adaptive programming education within the Agni platform. It is capable of automatically extracting programming concepts from educational materials, generating concept graphs that represent prerequisite relationships, modeling student knowledge using BKT, providing personalized lesson recommendations based on student performance, identifying failed concepts and giving contextual feedback during exercise attempts.

These features reduce the need for manual construction of domain models and provide support for learners, while maintaining teacher control over the learning process. By automating repetitive and time-consuming ITS tasks, the proposed approach lowers some of the barriers associated with maintaining ITS environments.

Additionally, the work contributes to the growing body of research exploring the intersection between GenAI and education. While recent advances in LLMs have shown strong capabilities in code generation and conversational interaction, this dissertation explored how these models can support classical ITS tasks such as domain modeling and student diagnosis. The obtained results suggest that GenAI can effectively assist in these tasks while still benefiting from teacher supervision and validation.

Overall, this work suggests that GenAI can effectively support and partially automate key components of ITS, providing an initial answer to the question posed in the introduction.

7.1. Discussion of Objectives

The first objective, Automatic Concept Extraction, was achieved through the development of a GenAI pipeline capable of identifying programming concepts from educational materials. The implemented system extracts concepts from Expositive and Evaluative materials, aligns them with existing platform concepts and allows teachers to validate the generated outputs. Overall, the validation showed good participant agreement with the AI generated concepts across two learning materials, supporting the effectiveness of the approach.

The second objective, Concept Graph Generation, was accomplished through the automatic generation of prerequisite relationships between programming concepts at the course level using LLMs. These generated graphs are visualized and editable within Agni, allowing teachers to refine the produced structures. Participants indicated strong alignment with the AI generated relationships between concepts.

The third objective, Failed Concepts Diagnosis, was addressed by integrating a failure analysis mechanism that uses GenAI to analyze incorrect student submissions and identify the concepts most likely responsible for the failure. Agreement between participants was evaluated across two programming exercises. The alignment between AI and participants in the results suggested that the system is capable of identifying the failed concepts.

The fourth objective, Feedback, was successfully implemented through a contextual hint generation mechanism integrated into exercise attempts. The generated hints were designed to guide students without revealing solutions directly. Evaluation participants considered the generated feedback as useful.

Beyond the individual objectives, the work also achieved the broader goal of integrating ITS support into the Agni platform. This included extending the frontend and backend to support concepts, concept graphs, student mastery modeling and content recommendations. Additionally, the introduction of a provider abstraction layer enabled support for multiple AI APIs.

Although the efficacy of the proposed approach was validated, its efficiency was not. In other words, while the system successfully achieved its objectives, it remains unclear whether this approach enables users to use ITS more quickly or encourages greater ITS adoption.

Overall, the objectives defined for this dissertation were achieved, resulting in a version of Agni that supports ITS features and is capable of leveraging GenAI to automate ITS components and support adaptive learning features.

7.2. Contributions

The major contributions of this dissertation were:

- The developed system, which is available online* for use, allowing teachers and students to interact with the implemented ITS and GenAI functionalities in a real environment.
- The publication† of a scientific paper with the title: "Exploiting Generative AI to Scale Up Intelligent Tutoring Systems", detailing the development process and obtained results of the system.
- The creation of a Docker image‡ containing the complete Agni platform, including backend and frontend components, simplifying deployment for anyone.
- An approach for automating the domain modeling and feedback mechanisms of ITS with the use of GenAI.

7.3. Future Work

While the proposed system shows good results for integrating GenAI into ITS, several directions for future work can further improve its effectiveness.

One important area is the evaluation of the system. Larger-scale user studies should be conducted to better assess the effectiveness of the implemented components. Additionally, evaluating the system in real educational settings over longer periods of time would provide deeper insights into how students and teachers interact with these components.

Another direction is improving the reliability and consistency of AI generated outputs. Although techniques such as prompt engineering and majority voting were employed, GenAI models remain non-deterministic. Future work may explore more advanced prompting techniques and other strategies to improve consistency and reduce hallucinations.

In terms of user interaction, further improvements could be made to the feedback generation component. For example, adapting the level of detail of the hints based on the

*https://agni.dcc.fc.up.pt
†https://doi.org/10.4230/OASIcs.ICPEC.2026.2
‡https://github.com/rqueiros/agni/tree/feat/ai-features

student's proficiency could provide a more personalized experience. Additionally, the integration of more advanced student modeling approaches could be explored.

Another possible improvement is the expansion of the adaptive learning capabilities. Currently, recommendations are primarily based on concept mastery and failed concept detection. Future work could incorporate additional factors such as interaction history and exercise difficulty to provide more accurate personalization.

From a system perspective, additional AI providers and local open-source models could be integrated into the abstraction layer. This would improve flexibility, reduce dependency on external APIs and potentially lower operational costs. Exploring self-hosted models may also address privacy and latency concerns.

Another relevant direction is the integration of learning analytics dashboards for teachers. Such dashboards could provide visual insights into student progress, concept mastery distribution and common learning difficulties, supporting data-driven pedagogical decisions.

Finally, the approach could be extended beyond programming education to other domains where structured knowledge representation and adaptive learning are required.

References

[1] K. R. Koedinger and A. T. Corbett, “Cognitive tutors : technology bringing learning science to the classroom,” in *The Cambridge Handbook of the Learning Sciences*, ser. Cambridge Handbooks in Psychology, R. K. Sawyer, Ed. Cambridge University Press, 2006, pp. 61–77. [Online]. Available: https://hal.science/hal-00699796 [Cited on pages 1 and 3.]

[2] A. Mitrovic, “An intelligent sql tutor on the web,” *Int. J. Artif. Intell. Ed.*, vol. 13, no. 2–4, p. 173–197, Apr. 2003. [Cited on page 1.]

[3] A. Corbett, L. Kauffman, B. Maclaren, A. Wagner, and E. Jones, “A cognitive tutor for genetics problem solving: Learning gains and student modeling,” *Journal of Educational Computing Research - J EDUC COMPUT RES*, vol. 42, pp. 219–239, 03 2010. [Cited on page 1.]

[4] N. Baghaei, A. Mitrovic, and W. Irwin, “Supporting collaborative learning and problem-solving in a constraint-based cscl environment for uml class diagrams,” *I. J. Computer-Supported Collaborative Learning*, vol. 2, pp. 159–190, 09 2007. [Cited on page 1.]

[5] W. Villegas, D. Buenano-Fernandez, A. Navarro, and A. Mera-Navarrete, “Adaptive intelligent tutoring systems for stem education: analysis of the learning impact and effectiveness of personalized feedback,” *Smart Learning Environments*, vol. 12, 06 2025. [Cited on page 1.]

[6] J. Anderson, A. Corbett, K. Koedinger, and R. Pelletier, “Cognitive tutors: Lessons learned,” *Journal of the Learning Sciences*, vol. 4, pp. 167–207, 04 1995. [Cited on pages 1 and 9.]

[7] P. Suraweera, A. Mitrovic, and B. Martin, “Widening the knowledge acquisition bottleneck for constraint-based tutors,” *I. J. Artificial Intelligence in Education*, vol. 20, pp. 137–173, 01 2010. [Cited on pages 1 and 10.]

[8] Y. Bauer, J. P. Leal, and R. Queirós, “Improving teacher‘s user experience in a virtual learning environment,” Master’s thesis, Universidade do Porto, 2023. [Cited on pages 3 and 4.]

[9] K. VanLEHN, “The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems,” *Educational Psychologist*, vol. 46, no. 4, pp. 197–221, 2011. [Cited on page 8.]

[10] W. Ma, O. O. Adesope, J. C. Nesbit, and Q. Liu, “Intelligent tutoring systems and learning outcomes: A meta-analysis,” *Journal of Educational Psychology*, vol. 106, no. 4, pp. 901–918, 2014. [Cited on page 8.]

[11] J. Kulik and J. D. Fletcher, “Effectiveness of intelligent tutoring systems: A meta-analytic review,” *Review of Educational Research*, vol. 86, 04 2015. [Cited on page 8.]

[12] S. G. Smith and B. A. Sherwood, “Educational uses of the plato computer system,” *Science*, vol. 192, no. 4237, pp. 344–352, 1976. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.769165 [Cited on page 8.]

[13] B. Woolf, *Building Intelligent Interactive Tutors, Student-Centered Strategies for Revolutionizing E-Learning*. Elsevier and Morgan Kaufmann, 01 2008. [Cited on page 9.]

[14] A. T. Corbett and J. R. Anderson, “Knowledge tracing: Modeling the acquisition of procedural knowledge,” *User Modeling and User-Adapted Interaction*, vol. 4, no. 4, pp. 253–278, 1994. [Cited on page 9.]

[15] X. Zhou, Z. Zhang, X. Xie, and J. Zhang, “Deep learning based knowledge tracing in intelligent tutoring systems,” *Scientific Reports*, vol. 15, 07 2025. [Cited on page 10.]

[16] G. Akçapınar and E. Sidan, “Ai chatbots in programming education: guiding success or encouraging plagiarism,” *Discover Artificial Intelligence*, vol. 4, 11 2024. [Cited on page 11.]

[17] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, “A systematic survey of prompt engineering in large language models: Techniques and applications,” 2025. [Online]. Available: https://arxiv.org/abs/2402.07927 [Cited on page 11.]

[18] M. Kwak, J. Jenkins, and J. Kim, “Adaptive programming language learning system based on generative ai,” *Issues In Information Systems*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:263198606 [Cited on page 12.]

[19] T. Phung, V.-A. Pădurean, J. Cambronero, S. Gulwani, T. Kohn, R. Majumdar, A. Singla, and G. Soares, "Generative ai for programming education: Benchmarking chatgpt, gpt-4, and human tutors," 2023. [Online]. Available: https://arxiv.org/abs/2306.17156 [Cited on page 12.]

[20] P. Denny, S. MacNeil, J. Savelka, L. Porter, and A. Luxton-Reilly, "Desirable characteristics for ai teaching assistants in programming education," in *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ser. ITiCSE 2024. ACM, 2024, p. 408–414. [Online]. Available: http://dx.doi.org/10.1145/3649217.3653574 [Cited on page 12.]

[21] M. Pankiewicz and R. Baker, *Enhancing Student Focus and Problem-Solving with Real-Time LLM Feedback on Compiler Errors*. Springer-Verlag, 09 2025, pp. 412–426. [Cited on page 13.]

[22] T. Phung, M. Wu, H. Choi, G. Soares, S. Gulwani, A. Singla, and C. Brooks, "Bridging gaps between student and expert evaluations of ai-generated programming hints," 2025. [Online]. Available: https://arxiv.org/abs/2509.03269 [Cited on page 13.]

[23] M. Mueller, C. List, and M. Kipp, "The power of context: An llm-based programming tutor with focused and proactive feedback," in *Proceedings of the 6th European Conference on Software Engineering Education*, ser. ECSEE '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1–10. [Online]. Available: https://doi.org/10.1145/3723010.3723034 [Cited on page 13.]

[24] I. Pesovski, R. Santos, R. Henriques, and V. Trajkovik, "Generative ai for customizable learning experiences," *Sustainability*, vol. 16, p. 3034, 04 2024. [Cited on page 13.]

[25] J. Doughty, Z. Wan, A. Bompelli, J. Qayum, T. Wang, J. Zhang, Y. Zheng, A. Doyle, P. Sridhar, A. Agarwal, C. Bogart, E. Keylor, C. Kultur, J. Savelka, and M. 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*, ser. ACE 2024. ACM, Jan. 2024, p. 114–123. [Online]. Available: http://dx.doi.org/10.1145/3636243.3636256 [Cited on page 13.]

[26] A. Del Carpio Gutierrez, P. Denny, and A. Luxton-Reilly, "Evaluating automatically generated contextualised programming exercises," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE 2024. New

York, NY, USA: Association for Computing Machinery, 2024, p. 289–295. [Online]. Available: https://doi.org/10.1145/3626252.3630863 [Cited on page 14.]

[27] Y.-F. Lin, M. Fahim, F. Khan, and K. Sakamura, "Athena: A genai-powered programming tutor based on open-source llm," *2025 1st International Conference on Consumer Technology (ICCT-Pacific)*, pp. 1–4, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:279002135 [Cited on page 14.]

[28] E. K. Juuso Rytilahti, Oshani Weerakoon, "Exploring ai-driven programming exercise generation," *Proceedings of the 20th International CDIO Conference*, 2024. [Cited on page 14.]

[29] Y. Kumar, A. Manikandan, J. J. Li, and P. Morreale, "Optimizing large language models for auto-generation of programming quizzes," *2024 IEEE Integrated STEM Education Conference (ISEC)*, pp. 1–5, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:272713943 [Cited on page 14.]

[30] S. Tytenko, Y. Papkovych, M. Zhak, S. Shkelebei, and O. Vasyliev, "Ai-driven digital concept maps in learning environments: Empirical insights from student use," *Realidad y Reflexión*, pp. 87–101, 12 2025. [Cited on page 15.]

[31] K. Gwet, *Handbook of inter-rater reliability: The definitive guide to measuring the extent of agreement among raters*. Advanced Analytics, LLC, 01 2012. [Cited on page 40.]

Appendix A

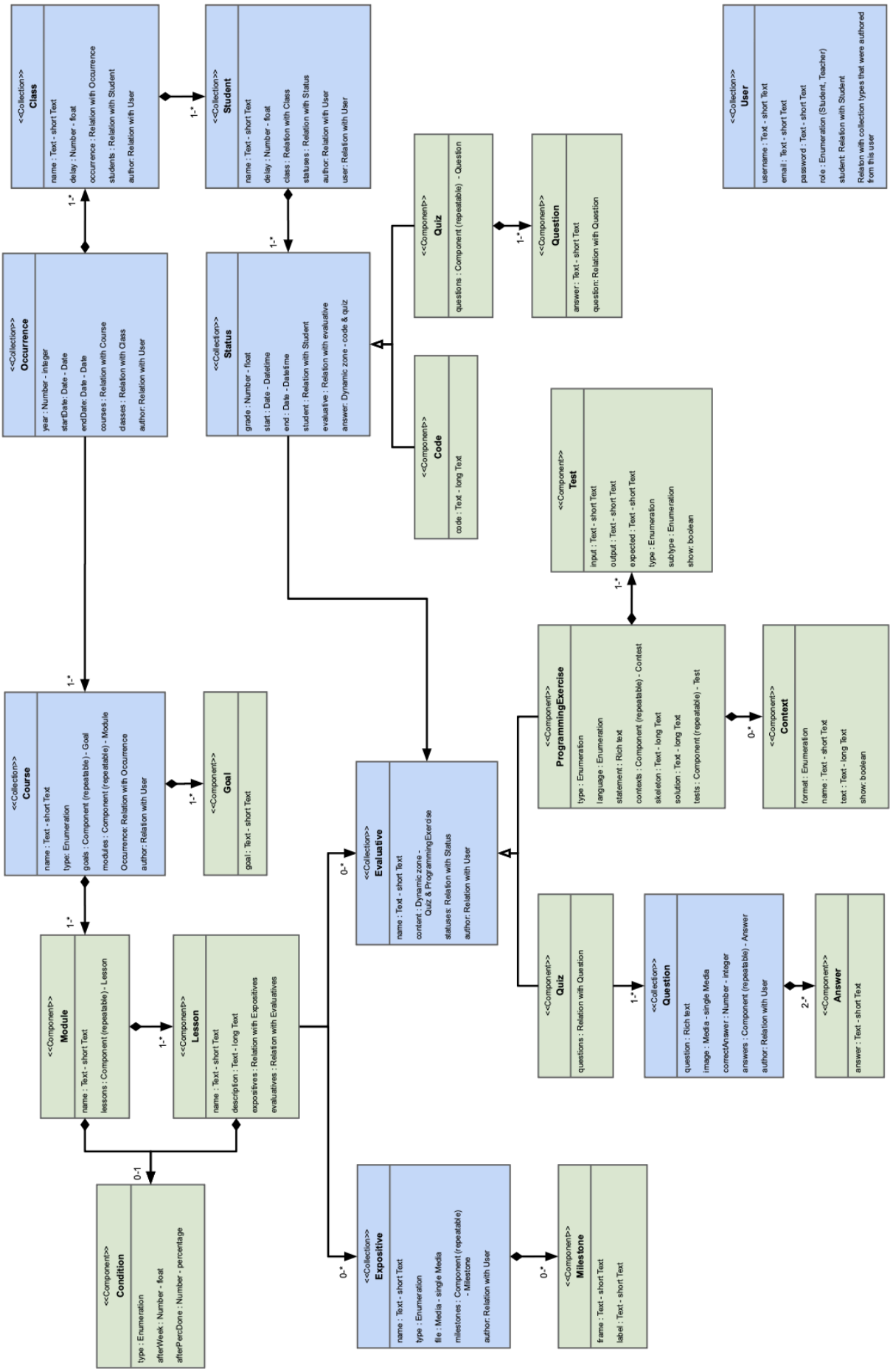

Figure 1: Original Data Model.

Appendix B

```
You are a Technical Curriculum Architect and Computer Science Professor.

Analyze the provided learning material and extract the DISTINCT PROGRAMMING CONCEPTS that are explicitly ta

IMPORTANT:
Create a NEW concept ONLY if the mechanism is fundamentally different from one already created.
Dont create new concepts when they represent the same underlying programming mechanism from already created

TEXT TO ANALYZE:
<<<
${data}
>>>

OBJECTIVE:
Identify the concrete programming mechanism taught in the text, at the level a curriculum would track.

CRITICAL RULES:

1. One Concept Per Mechanism
  Do NOT create separate concepts for:
    - Syntax variants
    - Sub-forms
    - Minor extensions
    - Special cases

Examples:
  - "if", "else if", "nested if" -> "Conditionals"
  - "for", "while", "do while" -> "Loops"

2. Language-Aware Naming
  - Prefer language-agnostic concepts unless the mechanism is language-specific.
  - Example:
    - "Closures"
    - "JavaScript Event Loop"

3. No Meta or Pedagogical Concepts
  FORBIDDEN:
    - "Programming basics"
    - "JavaScript overview"
    - "Logic"
    - "Syntax"

4. If a concept cannot be demonstrated with a short code snippet, it is NOT a valid programming concept.

5. Output Rules
```

```
  - Output ONLY valid JSON
  - No explanations
  - No duplicates

OUTPUT FORMAT:
{
  "concepts": [
    "Conditionals",
    "Variables",
    "Recursion"
  ]
}
Output ONLY JSON, no markdown, no commentary
```

Listing 1: Candidate concept extraction prompt

```
You are a Technical Curriculum Architect performing a NORMALIZATION pass.

Your task is to merge, generalize, and normalize the following candidate concepts into a minimal set of DIS

CANDIDATE CONCEPTS:
<<<
${JSON.stringify(candidates, null, 2)}
>>>

MANDATORY MECHANISM IDENTITY TEST:
Two concepts are the SAME if they:
  - Serve the same purpose
  - Can be grouped into a single concept
  - Differ only by syntax, structure, or specialization

RULES:

1. Canonical Naming
   - Prefer widely accepted curriculum terms
   - Use language-agnostic names unless impossible

2. If two concepts overlap, keep the more general one.

3. Prefer fewer, stronger concepts.

FEW-SHOT EXAMPLES:

INPUT:
[
  "If Statements",
  "Else If Conditions",
  "Nested Conditionals"
]
```

```
OUTPUT:
{
  "concepts": [
    "Conditionals"
  ]
}

INPUT:
[
  "For Loops",
  "While Loops",
  "Looping Over Arrays"
]

OUTPUT:
{
  "concepts": [
    "Loops"
  ]
}

OUTPUT FORMAT (JSON ONLY):
{
  "concepts": []
}

Output ONLY JSON, no markdown, no commentary
```

Listing 2: Concept extraction normalization prompt

```
You are a Knowledge Ontology Maintainer.

Your task is to reconcile newly normalized concepts with an existing controlled vocabulary (IF THE VOCABULA

NEW CONCEPTS:
<<<
${JSON.stringify(normalized, null, 2)}
>>>

CONTROLLED VOCABULARY (AUTHORITATIVE):
<<<
${JSON.stringify(allConcepts, null, 2)}
>>>

RULES:

1. Reuse Before Creating (ABSOLUTE)
   - If a new concept matches an existing one in meaning or mechanism,
```

```
     USE THE EXISTING LABEL EXACTLY.

2. Do NOT create:
   - Synonyms
   - Paraphrases
   - Slight renamings

3. Only keep a new concept if:
   - No existing concept represents the same mechanism.

4. Final list must be:
   - Minimal
   - Non-redundant
   - Curriculum-appropriate

FEW-SHOT EXAMPLE:

CONTROLLED:
["Conditionals"]

NEW:
["If Else Logic"]

OUTPUT FORMAT (JSON ONLY):
{
  "concepts": ["Conditionals"]
}

Output ONLY JSON, no markdown, no commentary
```

Listing 3: Concept extraction uniformization prompt

Appendix C

```
You are an educational diagnosis assistant.

Your task is to determine which provided learning concepts most likely caused a student to fail an exercise

You MUST:
  - Select ONLY concepts that appear in the provided concept list or in the prerequisite graph.
  - Consider BOTH:
    (a) Concepts explicitly associated with the exercise
    (b) Any of their prerequisite concepts from the graph
  - Use specific evidence from the student's code, the problem description and the solution code.
  - Identify the underlying misunderstanding, not just surface syntax mistakes.
  - Return AT LEAST one concept.

STEP 1 - Understand the task
Carefully read the problem text and the solution code and determine what skills are required.

STEP 2 - Analyze the student code
Identify:
  - Logical mistakes
  - Misuse of language features
  - Missing required structures
  - Incorrect algorithm design
  - Misunderstanding of control flow, data structures, etc.

STEP 3 - Map mistakes to concepts
  Match each mistake to the most fundamental concept that explains it.
  If a mistake is caused by misunderstanding a prerequisite concept, prefer selecting the prerequisite.

STEP 4 - Filter results
  Only return concepts that exist in:
    - The exercise concept list
    - OR their prerequisites from the graph

Do NOT invent concepts.
Do NOT explain your reasoning.
Do NOT output anything except valid JSON.

1. Exercise problem text
<<<
${problemText}
>>>

2. Student code
<<<
${code}
```

```
>>>

3. Solution code
<<<
${solution}
>>>

4. Concepts explicitly associated with this exercise
<<<
${JSON.stringify(conceptTags, null, 2)}
>>>

5. Course concept graph edges (A -> B means A is a prerequisite of B)
${JSON.stringify(conceptGraph, null, 2)}

Output format (JSON ONLY):

{
  "concepts": ["concept_name_1", "concept_name_2"]
}

Output ONLY JSON, no markdown, no commentary
```

Listing 4: Student diagnosis prompt