

Institute of Software Engineering
Software Quality and Architecture

University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Bachelorarbeit

AI Tutoring in Software Engineering Education

Moritz Dallinger

Course of Study: Informatik

Examiner: Prof. Dr.-Ing. Steffen Becker

Supervisor: Niklas Meißner, M.Sc.

Commenced: June 11, 2025

Completed: January 12, 2026

Abstract

Context. Software engineering education is a complicated topic that requires effective teaching methods to ensure students acquire the necessary skills and knowledge. Timely and effective feedback is crucial for students to achieve optimal learning outcomes.

Problem. Traditional teaching methods often fall short in providing feedback in a timely and individualized manner, leading to suboptimal learning outcomes. Often, students do not actively seek feedback, which further hinders their learning process.

Objective. The objective of this thesis is to develop an artificial intelligence (AI)-tutor that can provide personalized feedback and adapt to different learning styles, to enhance the learning experience for students in software engineering education. Additionally, we explore proactive feedback mechanisms to further support student learning.

Method. We conduct a requirements engineering process to gather insights into the needs of students and educational theory to design an AI-tutor that provides not only reactive but also proactive feedback, both tailored to different player types. The approach is implemented into the existing learning platform MEITREX and evaluated through an experimental study with students.

Result. We identified key insights on requirements for an AI-tutor in software engineering education, specifically on student perception of proactive feedback. Our evaluation indicates that students benefit from the personalized feedback provided by the AI-tutor, leading to improved motivation, but acceptance of proactive hints varies among students.

Conclusion. The developed AI-tutor enhances software engineering education by providing personalized feedback, tailored to the individual student through different player types and student skill. However, while the reactive feedback approach seems promising, further research is required to optimize the AI-tutor's proactive feedback approach.

Kurzfassung

Kontext. Das Erlernen von Software Technik ist ein kompliziertes Thema, das effektive Lehrmethoden erfordert, um sicherzustellen, dass Studierende die notwendigen Fähigkeiten und Kenntnisse erwerben können. Zeitnahes und effektives Feedback ist dabei entscheidend, damit Studierende optimale Lernergebnisse erzielen können.

Problem. Traditionelle Lehrmethoden sind oft unzureichend darin, Feedback zeitnah und individuell bereitzustellen, was zu suboptimalen Lernergebnissen führt. Oftmals suchen Studierende auch nicht aktiv nach Feedback, was ihren Lernprozess zusätzlich behindert.

Ziel. Ziel dieser Arbeit ist die Entwicklung eines künstliche Intelligenz (KI)-Tutors, der personalisiertes Feedback geben und sich an verschiedene Lernstile anpassen kann, um die Lernerfahrung von Studierenden im Erlernen von Software Technik zu verbessern. Zusätzlich untersuchen wir proaktive Feedback Mechanismen, um den Lernprozess der Studierenden weiter zu unterstützen.

Methode. Wir führen einen Requirements-Engineering-Prozess durch, um die Anforderungen der Studierenden sowie gängiger Literatur zu erarbeiten. Darauf basierend entwerfen wir einen KI-Tutor, der nicht nur reaktives, sondern auch proaktives Feedback liefert, welches jeweils auf verschiedene Spielertypen zugeschnitten ist. Der Ansatz wird in die bestehende Lernplattform MEITREX implementiert und mittels einer experimentellen Studie mit Studierenden evaluiert.

Ergebnis. Wir haben wichtige Anforderungen bezüglich eines KI-Tutors in Erlernen von Software Technik identifiziert, insbesondere im Hinblick auf die Wahrnehmung von proaktivem Feedback durch die Studierenden. Unsere Evaluation deutet darauf hin, dass Studierende von dem personalisierten Feedback des KI-Tutors profitieren, was zu einer gesteigerten Motivation führt, während die Akzeptanz proaktiver Hinweise jedoch unter den Studierenden variiert.

Fazit. Der entwickelte KI-Tutor verbessert das Erlernen von Software Technik durch die Bereitstellung von personalisiertem Feedback, das durch verschiedene Spielertypen und das aktuelle Fähigkeitsniveau auf den einzelnen Studierenden zugeschnitten ist. Während der Ansatz der Personalisierung des reaktiven Feedbacks vielversprechend erscheint, ist jedoch weitere Forschung von Nöten, damit das proaktive Feedback des KI-Tutors optimiert werden kann.

Contents

1	Introduction	1
2	Foundations	3
2.1	Feedback	3
2.2	Large Language Models	5
2.3	Retrieval Augmented Generation	6
2.4	Prompting and Prompt Engineering	6
2.5	AI Chatbot	6
2.6	HEXAD Player Types	7
2.7	Learning management systems	8
2.8	MEITREX	8
3	Related Work	11
3.1	Literature Research Methodology	11
3.2	Iris, AI-tutor in Artemis	11
3.3	CodeHelp	12
3.4	CS50 Duck	12
3.5	Adaptive Immediate Feedback System by Marwan et al.	12
3.6	Ask-Elle	13
3.7	Proactive Programming Assistant by Impiö	13
3.8	Context-Aware and Proactive Feedback in LLM-based ITS	13
4	Survey outline and goals	15
4.1	Motivation for conducting a survey	15
4.2	Theoretical motivation of the survey constructs	15
4.3	Study structure	16
4.4	Poll goals	21
4.5	Students interviewed	23
5	Survey results	25
5.1	A look into the students	25
5.2	AI-tutors compared to human tutors regarding trust (RQ4.1)	26
5.3	Help seeking behavior (RQ4.2)	29
5.4	Proactive Feedback (RQ4.3)	31
5.5	Motivational Support (RQ4.4)	34
5.6	No code submissions (RQ4.5)	35
6	Design of the AI-tutor	37
6.1	Design Goals	37
6.2	AI-tutor functionality and already existing workflows	38

6.3	Reactive Feedback	39
6.4	Proactive Feedback	45
6.5	Examples	50
7	Implementation of the AI-tutor additions	51
7.1	Gamification usage and prompt building	51
7.2	Implementation of the AI-tutor logic	55
7.3	Proactive Feedback	57
7.4	Communication between different microservices	57
7.5	Thresholds and constraints for evaluation	59
8	Evaluation	61
8.1	Study Design	61
8.2	Results	64
8.3	Discussion	70
8.4	Threats to Validity	73
9	Conclusion	75
9.1	Summary	75
9.2	Benefits	76
9.3	Limitations	76
9.4	Lessons Learned	76
9.5	Future Work	77
	Bibliography	79
A	Survey Appendix	85
A.1	Survey Questionnaire (translated to English)	85
A.2	Survey Questionnaire (original in German)	91
A.3	Raw Survey data	97
B	Evaluation Appendix	99
B.1	Evaluation Questionnaire	99
B.2	Raw Evaluation Results	101
C	Implementation Appendix	103
C.1	All Prompt Templates	103
D	Implementation Appendix	109
D.1	Some example Templates	109

List of Figures

2.1	Intrinsic HEXAD player types	7
2.2	Player type subtypes	8
2.3	Disruptor subtypes	8
5.1	Frequency of AI tool usage among the surveyed students	26
5.2	Trust in human tutors among the surveyed students	26
5.3	Trust in AI-tutors among the surveyed students	26
5.4	Frequency of actively seeking help when encountering programming problems	32
5.5	Student attitudes toward proactive explanations when the topic is already fully understood	32
5.6	Student reactions to unsolicited hints during independent problem solving	33
6.1	Student asks a question about lecture materials	40
6.2	Student-initiated code feedback scenario	41
6.3	Follow-up interaction flow	43
6.4	Assessment hints interaction scenario	45
6.5	Proactive code feedback after repeated test failures	46
6.6	Proactive feedback after test completion	47
6.7	Proactive interaction after lecture milestone completion	48
6.8	Proactive interaction during long idle periods	49
7.1	Component diagram showing the communication architecture	59
8.1	Frequency of AI tool usage among the evaluated students	64
8.2	Distribution of participants by course of study	64
8.3	Distribution of participants by semester	65
8.4	The AI-tutor’s coding feedback was clear and understandable	66
8.5	Willingness to use the AI-tutor again for coding exercises	66
8.6	The AI-tutor gave me too much information	67
8.7	The hints provided by the AI-tutor were fitting to my personal skill	67
8.8	Diverging stacked bar chart of proactive feedback ratings	69
8.9	Diverging stacked bar chart of general user experience ratings	71

List of Tables

5.1	Cross-tabulation of trust in human tutors versus trust in AI-tutors	27
5.2	AI usage frequency versus trust in AI-generated answers	28
5.3	Willingness to use an AI-tutor for code feedback	28
5.4	Reasons for contacting human tutors	29
5.5	Reasons for potentially using an AI-tutor	30
5.6	Ranking of preferred help sources	31
5.7	Cross-tabulation of attitudes toward motivational feedback	34
5.8	Student attitudes toward receiving stylistic feedback without exact code solutions	35
8.1	Evaluation of AI-tutor’s helpfulness in code improvement	66
8.2	Cross-tabulation of explanation complexity evaluation	68
8.3	Evaluation of proactive feedback features	69
8.4	General user experience evaluation	70
A.1	Distribution of students by course of study and study progress	97
A.2	Biggest programmed project by students	97

Listings

7.1	Prompt template for question classification	52
7.2	Skill level guidance generation for achiever player type	53
7.3	Player type-specific hint customization	53
7.4	Template arguments for prompt construction	54
7.5	Example prompt template for follow-up questions	56
C.1	Lecture question prompt template	104
C.2	Code feedback prompt template	105
C.3	Follow-up question prompt template	106
C.4	Generating hint prompt template	107
C.5	Proactive feedback prompt template	108
D.1	Prompt - Answer Example Question classification	110
D.2	Prompt - Answer Example Proactive Hint for Philanthropist	111
D.3	Prompt - Answer Example with chat history	112

Acronyms

AI artificial intelligence. iii

CI confidence interval, here with a confidence level of 95%. 64

ITS intelligent tutoring system. 13

KB knowledge base. 6

KI künstliche Intelligenz. v

LLM large language model. 1

LMS Learning Management System. 12

M median. 64

RAG retrieval-augmented generation. 6

SDv standard deviation. 64

1 Introduction

Software engineering education is crucial for preparing students to become proficient software developers. Feedback plays a vital role in the learning process. Not only does the content of feedback matter, but also the timing and frequency of feedback are important factors to consider [CB18].

In traditional education, feedback is often provided by human instructors. However, this approach has limitations, such as scalability issues and delays in feedback delivery. Often, human instructors do not notice when students who do not initiate the interaction need feedback, leading to missed learning opportunities. Additionally, students have different preferences, which can make it challenging to provide personalized feedback that meets their individual needs in a timely manner. While some students seek help from existing large language model (LLM)s, these models often lack the exact content of the course the student is enrolled in, leading to suboptimal feedback quality. Moreover, these models might provide feedback that is not aligned with research on effective feedback in education or *hallucinates* incorrect information, which can further hinder the learning process.

This thesis aims to develop an AI-tutor that can provide personalized feedback to students according to the individual hexad player type of the student as shown in Marczewski [Mar15] and the individual skill the student possesses. The developed AI-tutor should focus on software engineering education, which also includes the help of students in programming tasks. Furthermore, the AI-tutor should be able to provide feedback to students not actively requesting feedback, which is referred to as proactive feedback. To support the development of proactive feedback, this thesis aims to gather insights on how students perceive proactive feedback in the context of software engineering education in the requirements elicitation phase.

To achieve the objectives of this thesis, we follow a structured plan. First, we start with a literature review to establish the theoretical foundations of AI-tutors, feedback in education and player types. Secondly, we conduct a survey to gather requirements for an AI-tutor in software engineering education, focusing on code feedback and proactive feedback and building on top of the literature review. This requirements elicitation phase results in a design of a personalized AI-tutor. This design is then implemented into the existing learning platform MEITREX. Finally, we evaluate the implemented AI-tutor through an experimental study with 17 students to assess its effectiveness in providing personalized feedback and proactive feedback. The results of this thesis include key insights on requirements for an AI-tutor in software engineering education, as well as insights on student perception of proactive feedback. The evaluation of the implemented AI-tutor within MEITREX indicates that students benefit from the personalized feedback provided by the AI-tutor, which could lead to potentially improved learning outcomes.

The developed AI-tutor shows promise in enhancing software engineering education by providing personalized feedback tailored to different player types and student skill. While the initial results are promising, further research is required to fully optimize the proactive feedback approach of the AI-tutor.

Contributions

The contributions of this thesis are as follows: We gather key insights on requirements for an AI-tutor in software engineering education, focusing on code feedback and proactive feedback. Additionally, we design and implement an AI-tutor that provides personalized feedback based on the hexad player type and individual skill of students. Lastly, we evaluate the implemented AI-tutor through an experimental study, demonstrating its effectiveness in enhancing student learning outcomes.

We address the following research questions:

- RQ1.1 What are the requirements for an AI-tutor in software engineering education, especially regarding code feedback and proactive feedback?
- RQ1.2 Is proactive feedback perceived as helpful by students in software engineering education?
- RQ1.3 How can an AI-tutor be designed to provide personalized feedback based on player types and individual skill?

Thesis Structure

The structure of this thesis is as follows:

Chapter 2 – Foundations: First, we provide concepts relevant for the thesis.

Chapter 3 – Related Work: Here, we discuss related work and highlight the differences from our approach.

Chapter 4 – Survey outline and goals: This chapter outlines the methodology we used to conduct a survey to gather requirements for an AI-tutor.

Chapter 5 – Survey results: Here, we evaluate and discuss the results of the survey.

Chapter 6 – Design of the AI-tutor: This chapter presents the design of our approach based on the requirements gathered from the survey.

Chapter 7 – Implementation of the AI-tutor additions: Here, we describe the implementation of our approach.

Chapter 8 – Evaluation: Here, we provide an evaluation of our implementation based on an experimental study.

Chapter 9 – Conclusion: We conclude our thesis and give an overview of final insights as well as future work.

2 Foundations

This chapter provides an overview of the fundamental concepts and theories relevant to understand this thesis. Section 2.1 gives an overview of feedback in general with focus on programming education. Section 2.2, 2.4 and 2.5 introduces LLMs, prompt engineering and the concept of an AI-tutor. Section 2.6 introduces the HEXAD Player Types which are used in MEITREX. While Section 2.7 introduces learning management systems in general, Section 2.8 introduces MEITREX, the learning management system this thesis builds upon.

2.1 Feedback

Feedback is an important and basic part of general education and also software engineering education. It has been defined as “any of the numerous procedures that are used to tell a learner if an instructional response is right or wrong” [Kul77, p. 211]. Beyond simple verification, feedback serves to extend the learner’s knowledge state by providing specific information about correctness [JM07]. This challenges the receiver when new information contradicts prior understanding [CB18].

Crisp and Bonk [CB18] suggest that feedback should not be viewed as a single event but as a multidimensional construct.

2.1.1 Dimensions of effective Feedback

To evaluate the quality of the feedback experience, Crisp and Bonk [CB18] identify six critical dimensions: *timeliness*, *frequency*, *distribution*, *source*, *individualization*, and *content*.

Timeliness and Frequency

Crisp and Bonk [CB18] define timeliness as the time interval between a learner’s response and the receipt of feedback. Frequency refers to how often feedback is provided during the learning process [CB18]. There is a consensus in the literature that timely feedback is crucial for effective learning [CB18; GS05; Hat15; JM07]. It is crucial that feedback is received before the learner’s cognitive focus shifts away from the task [BCD89]. Merrill [Mer13] argues that in problem-based learning, timely feedback is essential when students encounter obstacles that hinder their process. Without this support, learners may struggle to progress effectively [Mer13].

Source

Source refers to who or what provides the feedback and how the learner is able to put trust in this specific source [CB18]. Learners must trust the source of the feedback to be effective [Fra16].

Content

Perhaps the most critical dimension is the content itself. Research indicates that *elaborated feedback* is significantly more effective than simple *knowledge of results*, which often produces no effect [Kul77]. As highlighted in the Crisp and Bonk (2018) framework, effective content should:

- Facilitate metacognition, helping learners analyze their own thinking processes [RML17].
- Extend beyond cognitive skills to address emotional and character development [RML17].
- Connect authentically to the learner's strengths and interests, potentially through mentoring approaches [WW17].

2.1.2 Feedback in Programming Education

In software engineering education, students learn how to program. However, effective feedback is critical during the learning progress [Shu08] and is important for software engineering education, because learning programming is hard [KJH16]. The problem is not only understanding basic programming concepts, but also combining and applying these concepts correctly [LAJ05]. Additionally, the large scale of programming courses makes individual teacher intervention time consuming [KJH16].

Taxonomies of Feedback

In educational theory, feedback can be categorized in four levels: *task*, *process*, *self-regulation*, and *self* [HT07]. However, automated feedback is often classified by the specific type of knowledge it conveys [KJH16]:

1. **Knowledge about Task Constraints:** Information regarding requirements or rules (e.g., specific constraints on loops or syntax)
2. **Knowledge about Concepts:** Explanations or examples related to the programming topics
3. **Knowledge about Mistakes:** Feedback derived from errors. This can include failed testcases, compiler errors, solution errors, style violations or performance issues
4. **Knowledge on How to Proceed:** Guidance to help a stuck learner move forward. Often in the form of hints or on what to do next

A systematic review of automated tools suggests that while feedback based on testing *knowledge about mistakes* is common and useful for pointing out errors, it varies significantly in depth [KJH16]. However, many current tools lack feedback for *knowledge on how to proceed*, which limits the system's ability to effectively support learning [KJH16].

2.1.3 Automated feedback and hybrid approaches

Additionally, there is research focusing on enhancing feedback through real-time analysis and generative AI [LKK25; WRZ+20]. Examples for automated feedback can be found in Related Work Chapter 3.

MacNeil et al. [MTH+23] found that students perceive LLM-generated code explanations as helpful. Additionally, there exist studies where students rate AI-generated code explanations as better than students code explanations [LDM+23].

In a hybrid feedback system, requests to the LLM can be combined with other information like for example results of testcases [LKK25]. Hybrid systems could lead to limited risks of AI hallucinations [LKK25]. This could lead to higher quality *knowledge of performance* [LKK25].

2.2 Large Language Models

LLMs are advanced artificial intelligence systems trained on massive textual datasets to understand, predict, and generate human-like language [VSP+17].

2.2.1 The Transformer Architecture

Most LLMs today are based on the Transformer architecture introduced by Vaswani et al. [VSP+17]. The core innovation of this architecture is the *Self-Attention Mechanism* Vaswani et al. [VSP+17].

In a Transformer, the input text is tokenized into numerical representations [VSP+17]. The LLM then weights which words in a sentence are most important to each other, even if those words are far apart [VSP+17]. This lets the model be able to process complex tasks such as summarization, translation, and question answering [BMR+20].

2.2.2 Training and Adaptation

The capabilities of an LLM are developed through a multi-stage process.

During *pre-training* the model is trained on a big corpus of text typically with *Next-Token Prediction* [BMR+20]. This training lets the model learn statistical patterns in the language by minimizing the loss between predicted token and the actual next token [BMR+20].

Afterwards, the model can be further refined through *fine-tuning* [OWJ+22]. In this stage, the model is trained on a more specific dataset to adapt it to particular tasks or domains [OWJ+22].

In-Context Learning

While fine-tuning is one way to adapt LLMs to specific tasks, it requires additional training and computational resources [BMR+20]. Alternatively, LLMs can perform *in-context learning* to adapt its behavior based on the input context [BMR+20].

2.2.3 LLMs in Software Engineering

In the context of software engineering, LLMs can help because source code shares many structural properties with natural language and is even more predictable [CSD19; HBG+16].

LLMs are capable of generating code fragments [Git24], explaining syntax [LDM+23] and identifying bugs or errors.

2.3 Retrieval Augmented Generation

retrieval-augmented generation (RAG) is a framework that combines pre-trained LLMs with an external knowledge base (KB) to enhance the models ability to generate contextually relevant responses [LPP+20]. RAG extends sequence-to-sequence models by integrating a differentiable neural retriever that queries a non parametric memory [LPP+20]. By doing so, there is no need to retrain the entire model to update its knowledge [LPP+20].

2.4 Prompting and Prompt Engineering

A prompt is the input given to a Large Language Model which generates an output based on that input while prompt engineering refers to the process of crafting and refining these inputs to achieve desired outputs [Gir23; Whi23]. By using prompts, it is possible to generate answers without having to retrain the model, to achieve *in-context learning* [BMR+20; LYF+23].

Different prompting strategies are often categorized by the context provided to the model [BMR+20]:

- **Few-shot prompting** adds a set of examples and corresponding completions before providing the final context on which the LLM should answer [BMR+20]
- **One-shot prompting** adds exactly one example and corresponding completion before providing the final context on which the LLM should answer [BMR+20]
- **Zero-shot prompting** does not provide examples but provides a task description in natural language on which the LLM should answer [BMR+20]

Additionally, it is possible to let a LLM act as a predefined persona to ensure the LLM to focus on specific details [Whi23].

2.5 AI Chatbot

An AI chatbot uses artificial intelligence to respond to user input. The chatbot is able to react to input and also keep context learned earlier in the conversation in mind. An AI chatbot can help with solving problems or also help with coding [VP18]. Additionally, AI chatbots can help the student to

find bugs or errors in provided code. GitHub's *CopilotChat*, for example, in a student's IDE is able to provide code suggestions, help with understanding code blocks and also creates unit tests for the user [Git24]. Often, students use AI chatbots for programming [GvRv24].

2.6 HEXAD Player Types

In gamification theory, different user types are often defined to categorize users based on their motivations and behaviors within gamified systems. One taxonomy is the Hexad Player Types by Marczewski, which identifies 6 distinct player types: *Achievers*, *Socializers*, *Free Spirits*, *Philanthropists*, *Players*, and *Disruptors* [Mar15].

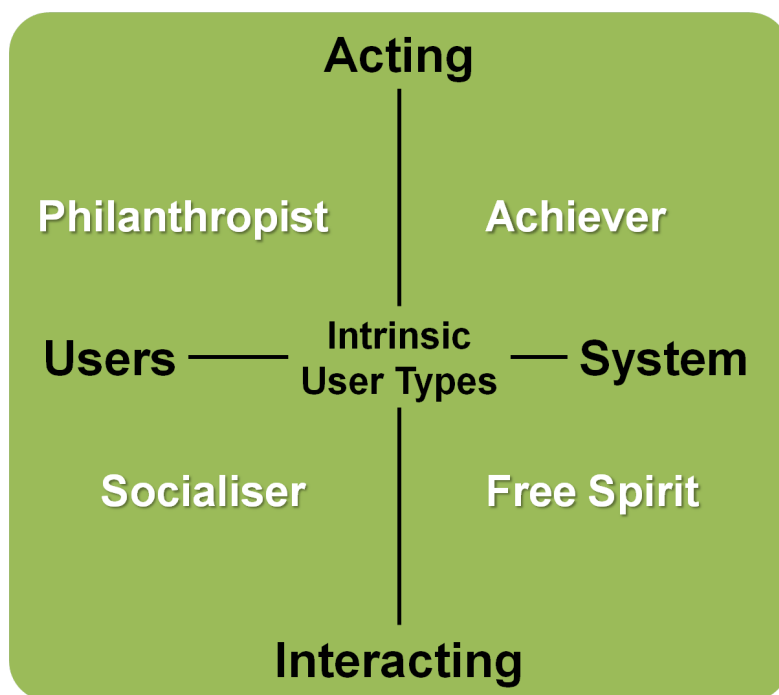


Figure 2.1: Intrinsic HEXAD player types [Mar15]

The intrinsic player types are:

- **Achievers** gain motivation through *mastery*. Usually, *achievers* want to complete challenges inside of the system and be the best for each [Mar15]
- **Socializers** gain motivation through the creation of social interactions. Usually, *socializers* want systems prompting *relatedness* [Mar15]
- **Free Spirits** gain motivation through *autonomy*. Usually, *free spirits* want systems supporting exploration or creativity [Mar15]
- **Philanthropists** gain motivation through *purpose and meaning*. Some *philanthropists* seek this in altruism while others simply have a desire to help others and improve their lives without expecting anything in return [Mar15]

These player types are visualised in graphic 2.1.

Other player types:

- **Players** gain motivation through *extrinsic rewards*. However, there are some subtypes *self-seeker, consumer, networker* and *exploiter* each behaving similar to the intrinsic player types [Mar15]. Player subtypes are visualised in graphic 2.2
- **Disruptors** gain motivation through *change* [Mar15]. Usually, only a small amount of people have the *disruptor* player type [Mar15; MBSB25]. *Disruptors* can be split into different subtypes: *griefer, destroyer, influencer* and *improver* [Mar15]. Disruptor subtypes are visualised in graphic 2.3

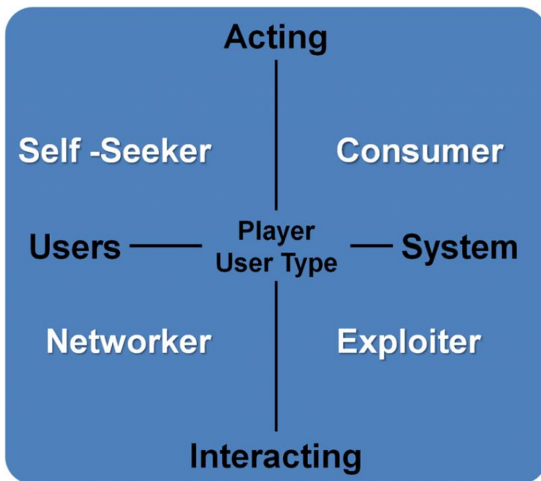


Figure 2.2: Player type subtypes [Mar15]

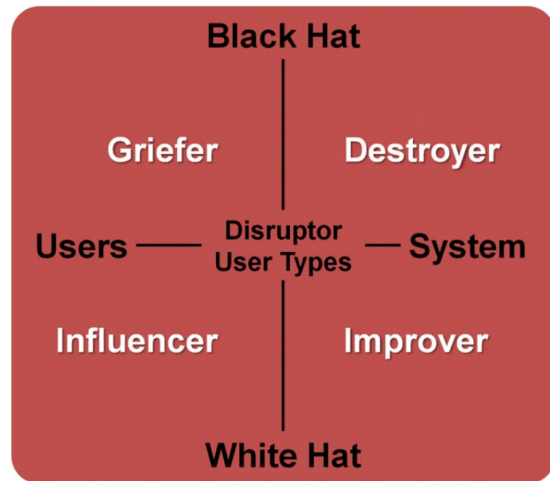


Figure 2.3: Disruptor subtypes [Mar15]

2.7 Learning management systems

A learning management system (LMS) is an e-learning tool designed to improve the learning experience for students [Bra21]. In these it is possible to offer different courses. Furthermore, the content that the student has already worked on can be recorded in each course. Examples are Ilias, Moodle, Schoologie, ATutor and many more. LMS are wide spread at universities and are therefore often used in software engineering education.

2.8 MEITREX

MEITREX is a learning management system with focus on software engineering education [Mei24]. To enhance student motivation and engagement, MEITREX uses *adaptive gamification framework* [MTS+26]. MEITREX tailors the learning experience based on three dimensions: the student's *HEXAD player type*, their current *course progress*, and their *competence level* aligned with the CS2023 standard [MTS+26]. This way, MEITREX ensures every student is supported as efficiently

as possible [Mar15; Mei24]. Following the adaptive gamification approach, widgets change accordingly to the individual student [MTS+26]. Additionally, MEITREX provides *adaptive content* that adjusts in real-time to the learner's progress and preferences [MTS+26].

MEITREX integrates a *DocProcAI* pipeline. This makes it possible to segment course materials like lecture videos, slides, and documents, into smaller units. It is then possible to utilize semantic search on the lecture materials which can be used by an AI-tutor to base answers on lecture materials. Currently, the *DocProcAI* is developed by students and not yet published.

Current implementation of MEITREX can be found at <https://github.com/MEITREX> [MEI25].

3 Related Work

This section gives an overview of existing AI-tutoring systems and proactive feedback mechanisms in software engineering education.

3.1 Literature Research Methodology

To identify related work, we used Google Scholar ¹ as a search engine. We used the following search queries:

- *AI tutoring in software engineering education*
- *proactive feedback in software engineering education using LLMs*

For each query, we selected the first 10 results and included only relevant papers that discussed AI-tutoring systems or proactive feedback mechanisms in software engineering education. Additionally, some papers were provided by our supervisor, while others were found by snowballing from the references of already identified papers. However, there is a lot of related work in the area of AI-tutoring systems, so we focused on a few that are closely related to our approach.

3.2 Iris, AI-tutor in Artemis

Artemis is an Automated Programming Assessment System designed for large-scale programming courses [FSB+24] with an ai-driven virtual tutor called Iris [BFK24]. It currently uses GPT-3.5-Turbo by OpenAI to automatically give AI-Feedback inside the code editor [BFK24; FSB+24]. Iris employs a *Chain-of-Thought* prompting strategy combined with few-shot learning to process students requests effectively [BFK24]. Iris chooses relevant parts of the repository of the student as context to a question [BFK24]. However, Iris also processes the exercise description, automated feedback from tests as well as the previous chat history [BFK24]. The tool focusses to never provide code in the answer and act more as an *excellent tutor* [BFK24].

Iris utilizes a relevance score to filter out questions that are not relevant to the current topic [BFK24]. Before returning the final answer to the student, Iris performs a self-evaluation step to check if the answer is appropriate [BFK24]. If the answer is not appropriate, Iris tries to improve the answer by refining the response until it is satisfactory [BFK24].

However, Iris only answers questions that the student asks reactively. Our approach also explores proactive behaviour, which is currently not supported by Iris.

¹<https://scholar.google.com/>

3.3 CodeHelp

CodeHelp is a web based interface to offer students help with programming exercises through a help request form [LSSD24]. The focus is on creating robust *guardrails* that do not reveal solutions to the student, but only provide assistance [LSSD24]. Additionally, CodeHelp allows instructors to define an *avoid set* to prevent the AI from suggesting concepts not yet taught in the curriculum [LSSD24]. Students indicate the language they are programming in, the relevant code block, possibly an error message and the question they have [LSSD24]. The system utilizes a multi prompt workflow to first verify the sufficiency of the students query, to eventually display a clarification request if the query is insufficient [LSSD24]. Otherwise, the system generates two responses for the main prompt in parallel and picks the the higher scoring response [LSSD24]. If there are still code blocks in the response, a third prompt is used to remove any remaining code blocks [LSSD24].

The difference to our approach lies in the fact that CodeHelp requires students to actively request help through a form [LSSD24], while our approach also explores proactive feedback mechanisms. Additionally, in our approach, the AI-tutor is able to gather relevant code snippets automatically from the learning platform MEITREX, while CodeHelp requires students to manually input relevant code snippets [LSSD24]. Moreover, our approach integrates course materials as context for feedback generation, which is not supported by CodeHelp [LSSD24].

3.4 CS50 Duck

The Learning Management System (LMS) CS50 got an inbuilt duck with reference to *rubber duck debugging* as an AI tool to help students [LZL+24]. It currently uses GPT-4.0 to provide feedback [LZL+24]. The duck is accessed through a web browser or a separate VS Code extension [LZL+24]. The student is able to interact with the chatbot through writing, as well as getting side by side comparisons from their code and different code with a better style guide [LZL+24]. The system is designed to provide assistance in a pedagogically good manner [LZL+24]. To reduce *hallucinations*, the system uses RAG to base the responses on course context [LZL+24]. Additionally, the system prevents overuse by limiting the number of requests a student can make through a *hearts* system, where each student gets a new request once every 3 minutes banking up to 10 *hearts* [LZL+24]. Main functionality of the duck is to explain highlightes code snippets to the student, as well as option to evaluate code style and the chatbot to answer course related questions [LZL+24].

Unlike our approach, the duck does also not act proactively to the student. Additionally, the duck is not utilizing student player types to further personalize feedback.

3.5 Adaptive Immediate Feedback System by Marwan et al.

The adapted immediate feedback system by Marwan et al. provides personalized, real-time feedback to novice programmers [MGF+20]. The system uses *objective detectors* to monitor student progress and trigger feedback based on sub-goal task completion [MGF+20]. Additionally, the system includes a visual progress panel and pop-up messages that offer encouragement or reinforcement [MGF+20].

Unlike our approach, this system does not utilize LLMs for feedback generation and only interacts through pre defined feedback messages [MGF+20].

3.6 Ask-Elle

Ask-Elle is an intelligent tutoring system (ITS) supporting the learning of the strongly-typed functional programming language Haskell [GHJv17]. The system supports the incremental, stepwise development of programs by verifying incomplete solutions and providing automated hints [GHJv17]. Teachers can define exercises along with model solutions and properties that solutions should satisfy [GHJv17]. Ask-Elle achieves this functionality by a combination of strategy-based model tracing and property-based testing, which are used to generate feedback [GHJv17].

However, Ask-Elle does not utilize LLMs for feedback generation and does not provide proactive feedback to students. Additionally, Ask-Elle is specifically designed for Haskell programming, which limits its applicability to other programming languages and broader software engineering education contexts.

3.7 Proactive Programming Assistant by Impiö

Impiö developed a proof-of-concept proactive programming assistant that provides real-time, personalized feedback to learners during coding activities [Imp25]. In this master thesis, proactive feedback is triggered based on following triggers: static code analysis finds multiple errors or warnings in student code, test cases the student initiated fail or the submitted code did not pass automated assessments [Imp25]. Additionally, if the student has been idle for a certain period of time, the system triggers proactive feedback [Imp25]. The LLM chatGPT-4o is used to generate personalized hints and suggestions [Imp25]. This work also provides code examples while providing a hint on what to do next [Imp25].

However, the work of Impiö does not integrate course material as context for feedback generation, while also not allowing students to interact with the LLM directly [Imp25]. This is where our approach differs, as our AI-tutor integrates course materials and allows students to interact with the LLM chatbot directly.

3.8 Context-Aware and Proactive Feedback in LLM-based ITS

The system describes an LLM-based ITS for introductory programming [MLK25]. The ITS utilizes code history of students to generate contextualized feedback [MLK25]. To address proactive support, a neural network is trained on user interaction data to predict optimal feedback phases [MLK25]. The student is still able to request feedback directly from the system with the press of a button [MLK25].

3 Related Work

However, the work of Mueller et al. does not include course materials as context for feedback generation [MLK25], unlike our approach. Additionally, while still utilizing user interaction history, this work does not individualize feedback according to the individual students player type [MLK25], which is a important aspect of our approach.

4 Survey outline and goals

To understand more about the requirements students have to an AI-tutor, we conducted a study here at the University of Stuttgart. This chapter outlines the structure of the survey, the goals of the survey as well as how the students got reached. This survey will be continued in chapter 5 where the survey gets evaluated.

4.1 Motivation for conducting a survey

To better understand the requirements software engineering students have to an AI-tutor, this survey investigates how software engineering students currently seek help when they have problems with programming tasks. Requirements for feedback students might have while solving software engineering related tasks might differ compared to non software engineering students. Human Tutors might traditionally provide the needed support, it is unclear if and in which form AI-tutors can compare to humans.

By investigating students help seeking behaviour, their trust in AI and possible use-cases for an AI-tutor, this study aims to identify expectations and constraints software engineering students have for an AI-tutor. Understanding these requirements is crucial for offering a well accepted AI-tutor.

4.2 Theoretical motivation of the survey constructs

The design of the survey is based on the theoretical foundations of effective feedback outlined in Chapter 2. To ensure the developed AI-tutor meets theoretical standards, the survey investigates specific dimensions of feedback identified by Crisp and Bonk, specifically focusing on source, timeliness and content.

As noted in the feedback dimensions by Crisp and Bonk, the learner must trust the source of feedback for it to be effective [Fra16]. It is therefore essential to understand how students perceive the credibility of an AI-tutor compared to human tutors.

Literature emphasizes that feedback must be timely to be effective [CB18]. Without timely feedback, learners might struggle to progress effectively [Mer13]. Therefore, we investigate students actively seeking help and their perception of proactive feedback from an AI-tutor.

Sometimes, *knowledge on how to proceed* is needed for effective learning and is often overlooked by automated feedback systems [KJH16]. Additionally, it is important to understand how students perceive no code submissions from an AI-tutor, to ensure that the AI-tutor encourages learning rather than providing direct solutions. This claim is not only supported by literature [KJH16; Kul77] but also used in other related work like Bassner et al. or Liffiton et al.

While gamification theories like the HEXAD player types [Mar15] suggest different motivational drivers for users, it is necessary to understand the general student sentiment toward motivational feedback from an AI-tutor.

4.3 Study structure

The survey was organized into four sections to systematically address the research objectives. The first section collected participants demographic information. The second section examined general tutoring practices and experiences. The third section investigated attitudes toward AI-tutors. The fourth section consisted of open-ended questions, providing participants the opportunity to elaborate on their perspectives regarding AI-tutoring.

The study contained 18 questions and 3 open feedback sections. The survey was conducted in German, as the target group was students of the University of Stuttgart. However, the questions have been translated into English for this thesis. The original questions in German and the translated questions in English can be found in the Appendix A. The poll was created and carried out using the online survey tool LamaPoll.

The survey was piloted with one member of the SQA institute as well as my supervisor to ensure clarity and relevance of the questions. Additionally, I tested the survey with two computer science students to confirm that the questions were understandable.

4.3.1 Demographic and educational background

This part of the survey collects basic demographic and educational background data (e.g., study program, prior programming experience, AI usage habits). The answers to these questions help contextualise the later responses and enable comparisons between different student profiles. They also ensure that differences in attitudes toward AI-tutors can be interpreted in light of the participants' experience levels.

This part of the survey consists of following 6 question topics:

- Course of study
- Progress in studies
- Computer science at school before starting university?
- Largest project programmed to date?
- AI usage frequency
- Experience as tutor

Exact wording of the questions and answer options can be found in the Appendix A.1.1.

Course of study

This question collects the students' course of study. The options were based on the different software engineering related courses of study at University of Stuttgart. Additionally, students had the option to write down their own course of study in case it was not listed.

Progress in studies

This question collects the students progress in their studies. The answer options were chosen to distinguish between someone who is in the beginning of their studies (Bachelor Semester 1-3), someone who is more advanced in their studies (Bachelor Semester 4+) and someone who is already doing a Master. The options were chosen this way to ensure that participants of the study can not be identified personally based on their progress in studies and course of study.

Computer science at school before starting university?

This question collects if the student had computer science at school before starting university. This might give an indication of how much prior experience the student has with computer science before starting university. Students who had computer science at school may have more experience with programming and computer science concepts than students who did not have computer science at school.

Largest project programmed to date?

It might be possible that students with more programming experience have different requirements to an AI-tutor than students with less programming experience. To determine the programming experience of the students in a quick manner, we asked them about the largest project they have ever programmed to date. The answer options were chosen to distinguish between small scripts, medium sized projects and large projects. However, more lines of code do not necessarily indicate more experience in programming but rather the complexity and quality of the code. Accurately estimating the programming experience within a short time frame can be difficult, particularly for students who are new to programming or are self-taught. Therefore, this question only gives a rough indication of the programming experience of the students.

AI usage frequency

This question collects how often the student uses ai tools in their daily life. The answer options were chosen to distinguish between daily usage, multiple times a week, once a week and rarely. They give an indication of how familiar the student is with ai tools in general.

Experience as tutor

This questions simply asks if the student has experience as a tutor. It might give an interesting insight if students with tutoring experience have different requirements to an AI-tutor than students without tutoring experience.

Not asked questions

The study did not contain any questions about the player type of the students. At the time of conducting the survey, the consensus was to investigate the AI-tutor outside of the player types and gamification done inside of MEITREX.

Additionally, no questions about gender or age were asked. This was done to keep the survey as short as possible and to not intimidate students from answering the survey. Given the small number of potential participants, it was therefore ensured that no personal information was collected that could identify the students.

4.3.2 About Tutoring in general

The second section explores students existing help-seeking behaviour, their trust in human tutors, and their preferred sources of support. These questions establish a baseline for understanding how students currently solve programming problems and form the comparative foundation for later items about AI-tutors.

It consists of following 4 questions:

- Trust in human tutors
- Preferred sources of help for solving programming tasks
- Seeking active help
- Reasons for contacting a human tutor

Exact wording of the questions and answer options can be found in the Appendix A.1.2.

Trust in human tutors

This question collects how much the student trusts their human tutor. This gives a baseline for later comparison with the AI-tutor. The scale was chosen as a 5 point likert scale from do not trust at all to fully trust.

Preferred sources of help for solving programming tasks

This question lets the student rank different sources of help for solving programming tasks. The options were chosen to include different possible sources of help. This list is not exhaustive but covers the most common sources of help. However, it gives a good overview of the most common sources of help for programming tasks.

Seeking active help

This question collects if the student actively seeks help when they have problems with programming tasks. The scale was chosen as a 5 point likert scale from never to very often.

Reasons for contacting a human tutor

This question lets the student select different reasons for contacting a human tutor. The options were chosen to include different possible reasons for contacting a human tutor. This list is not exhaustive but covers the most common reasons for contacting a human tutor. Therefore, an *other* option was included to let the student write down their own reason for contacting a human tutor.

4.3.3 About AI-tutors

This section focuses on attitudes toward AI-tutoring systems. It includes questions about trust in automated feedback, preferred forms of assistance, emotional responses to motivational messages, and preferences regarding stylistic feedback versus complete solutions. These items address the main research questions regarding the design requirements for an AI-tutor.

It consists of following 8 questions:

- Trust in referenced AI-generated answers
- Willingness to use AI for code analysis
- Reasons for contacting an AI-tutor
- Feelings about receiving redundant hints
- Feelings about receiving unsolicited help
- Reaction to motivational AI feedback
- Reaction to lack of motivational AI feedback
- Perception of stylistic feedback without concrete code solutions

Exact wording of the questions and answer options can be found in the Appendix A.1.3.

Trust in referenced AI-generated answers

This question collects how much the student trusts referenced ai-generated answers. The scale was chosen as a 5 point likert scale from do not trust at all to fully trust.

Willingness to use AI for code analysis

This question collects if the student would generally use an AI-tutor for code analysis. The scale was chosen as a 5 point likert scale from never to very often.

Reasons for contacting an AI-tutor

This question lets the student select different reasons for contacting an AI-tutor. The options were chosen to include the same possible reasons for contacting an AI-tutor as for contacting a human tutor in the previous section 4.3.2. However, an option for the student to write down their own reason for contacting an AI-tutor was not included. Reason for this is that this question will be asked in more detail later again in the open feedback section.

Feelings about receiving redundant hints

This question collects how the student would feel about receiving redundant hints from an AI-tutor. The scale was chosen as a 5 point likert scale from rejection to helpful and supportive.

Feelings about receiving unsolicited help

This question collects how the student would feel if the tutor would give them a hint which solves a key part of their problem without them asking for it. The scale was chosen as a 5 point likert scale from very disturbing to very helpful.

Reaction to (lack of) motivational AI feedback

These two questions collect how the student would feel about receiving motivational feedback from an AI-tutor. This is done with two similar questions in the Kano model [BBB+93]. Therefore, it is possible to determine if motivational feedback is a must-have, performance or excitement requirement for the AI-tutor. The scale(s) were chosen as a 5 point likert scale from bad to excellent.

Perception of stylistic feedback without concrete code solutions

This question collects how the student would feel about receiving stylistic feedback from an AI-tutor instead of exact solutions. The scale was chosen as a 5 point likert scale from very bad to very good.

4.3.4 Open feedback

The final section offers participants the opportunity to freely articulate their expectations, concerns, and suggestions concerning AI-based tutoring. These qualitative responses complement the quantitative data, providing additional insights and highlighting potential requirements or considerations that may not have been captured by the predefined answer options.

This section begins with a brief introduction outlining the objectives of the thesis and describing the AI-tutor. Participants were then invited to provide open-ended feedback on their experiences and perspectives regarding the AI-tutor. They got following 3 questions:

- What should I perhaps implement?
- What is important to you in this regard?
- What would you find particularly bad?

Exact wording of the questions can be found in the Appendix A.1.4.

The students were able to write down their thoughts without any restrictions.

4.4 Poll goals

The goal of the poll was to understand what requirements computer science students have to an AI-tutor. Moreover, we wanted to understand how computer science students would use an AI-tutor and what tasks they would like to be supported with it. But this study only gives a theoretical overview of the requirements. To understand the practical use of an AI-tutor, a prototype has to be built and evaluated with students afterwards.

There were multiple questions we wanted to answer with this study.

4.4.1 General interest

The primary objective was to determine the level of general interest in an AI-tutor at all. This includes if students would generally trust an AI-tutor and if so do they trust them more or less than a human tutor. Furthermore, we wanted to understand for what tasks students would like to use an AI-tutor for. Does an AI-tutor have more use cases than a human tutor?

I measure this with multiple questions. With question 7 cf. A.1.2 we wanted to understand how much students trust their human tutor. This gives a baseline for later comparison with the AI-tutor in question 11 cf. A.1.3. With question 8 cf. A.1.2 we wanted to understand how students currently seek help when they have problems with programming tasks. This gives an indication of how students currently solve problems and what role an AI-tutor could play in this. To measure if students would generally use an AI-tutor, we asked question 12 cf. A.1.3. Question 10 cf. A.1.3 then measures the baseline of interactions done with a human tutor. Finally, with question 13 cf. A.1.2 we wanted to understand for what reasons students would then ask an AI-tutor for help.

This would then generally give an overview if students would like to use an AI-tutor and for what tasks they would like to use it for.

4.4.2 Proactive

Additionally, we wanted to understand if students would actively ask the human tutor for help when they have problems with the course or programming tasks. With students who completed their school career during the Covid-19 pandemic, it is possible that students encounter greater barriers to communication and experience increased transactional distance [GG22]. The AI-tutor could therefore also be triggered proactively when the student has problems, without the student actively asking for help. After the student finished an assessment, the AI-tutor could for example analyse the results and proactively provide help to the student in topics where the student has problems. Or when the student is programming and the AI-tutor recognises that the student is stuck, the AI-tutor could proactively provide help or repeat important concepts.

Therefore, we wanted to understand to what extent students seek help on their own and how much they would appreciate proactive support from the AI-tutor. Furthermore, we wanted to understand if some student may be annoyed by proactive help from the AI-tutor. The goal was to understand how to design the AI-tutor in this regard.

We measure this with different questions. With question 9 cf. A.1.2 we wanted to understand if students actively ask for help when they have problems. This would give an indication if students would actively ask or if they would rather have proactive help from the AI-tutor. Not asking for help would indicate that students would rather have proactive help from the AI-tutor. However, there might be a risk that students get annoyed by proactive help from the AI-tutor. With question 14 and question 15 cf. A.1.3 we wanted to understand exactly this. Negative responses to these questions would indicate that students do not like too much proactive help from the AI-tutor. Thus, the AI-tutor should rather wait for the student to ask for help instead of proactively providing help. Positive responses however would indicate that students like proactive help from the AI-tutor. Thus, the AI-tutor should proactively provide help to the student when it recognises that the student has problems.

4.4.3 Empathy

Another goal of the study was to understand how important empathy and motivational feedback is to students when they get help from an AI-tutor. When students are stuck on a programming task for a long time, they may get frustrated and demotivated. An AI-tutor could provide motivational feedback to the student in this situation to keep the student motivated. However, it is also possible that students do not like to get motivational feedback from an AI-tutor. Therefore, we wanted to understand how students would feel about receiving motivational feedback from an AI-tutor.

This got measured with question 16 and question 17 cf. A.1.3. They are similar questions, with the only difference being if the student receives motivational feedback or not. Therefore, we can check with the Kano model [BBB+93] how important motivational feedback is to students.

4.4.4 No code feedback

Lastly, we wanted to understand how students would feel about receiving stylistic feedback from an AI-tutor instead of exact solutions. When students are programming, they may write code that is working but not stylistically good. As shown in chapter 2, providing exact solutions to students may lead to students not learning the concepts properly. Therefore, we wanted to understand how students would feel about receiving stylistic feedback from an AI-tutor instead of exact solutions.

To verify this, we asked question 18 cf. A.1.3.

4.4.5 Research Questions

To translate the goals of this study into a concrete empirical investigation, the following research questions were defined. Each question corresponds to one of the thematic areas of the survey.

RQ4.1 To what extent are computer science students interested in using an AI-tutor, and how much do they trust its technical accuracy compared to a human tutor?

RQ4.2 How do students currently seek help when working on programming tasks, and what role could an AI-tutor play within their existing help-seeking strategies?

RQ4.3 How do students evaluate proactive assistance from an AI-tutor, particularly in situations of frustration or difficulty?

RQ4.4 How do students feel about receiving empathetic or motivational responses from an AI-tutor?

RQ4.5 How do students judge stylistic or conceptual feedback that does not include exact code solutions?

4.5 Students interviewed

The study got answered by 34 students of University of Stuttgart. This is a good number of answersets to get a good overview of the requirements computer science students have to an AI-tutor.

4.5.1 How did the students get reached?

The study was online between 16.07.2025 and 01.08.2025. In this time, students were able to go to the website and answer the question. During this time, we provided the link to students studying inside V38, which is the Computer Science Building at Campus Vaihingen of University of Stuttgart. We reached these students by actively asking them if they would like to participate in the survey.

Additionally, an E-Mail was sent to the mailinglist of the SQA institute by my supervisor Niklas Meißner. This mailinglist contains students who are writing their bachelor or master thesis at the SQA institute or did so in the past. However, participants did not get any incentive to participate in the survey.

limitations

Sadly, this limits the generalizability of the results, since only students studying at University of Stuttgart and actively studying with multiple people could participate in the survey. Students studying at home alone could not be reached this way. Students had no incentive to participate in the survey, other than helping me with my bachelor thesis. Therefore, it is possible that only students generally willing to help with others participated in the survey. This could have biased the results of the survey.

Additionally, the survey was conducted at a time where the thesis was intended to not include any gamification elements or the HEXAD player type. This got changed later on during the thesis after the survey was already conducted and we were in the design or implementation phase of the AI-tutor.

5 Survey results

This chapter evaluates the results of the conducted survey with respect to the defined research questions. It is focusing on students trust in AI-tutors, help-seeking behavior, attitudes toward proactive assistance, and preferences regarding empathetic and stylistic feedback. Full results of the survey can be found as part of our research dataset [DM26b].

5.1 A look into the students

This section provides an overview of the demographic and educational background of the participants included in the analysis. The presented results serve to contextualize the subsequent findings and do not yet address the research questions of this study.

In total, 34 complete response sets were included in the evaluation. All participants were students enrolled at the University of Stuttgart in software engineering related study programs. The majority (24/34) of respondents were enrolled in a Bachelor program, with 10 students within the first 3 semesters, with 14 students from higher Bachelor semesters and 10 students enrolled in a Masters program. Exact values of semester and course of study are shown in Figure A.1.

Regarding prior exposure to computer science, a substantial portion of participants reported having had computer science courses before entering university. This indicates that many respondents did not encounter programming for the first time during their academic studies.

Programming experience was additionally measured by asking participants to estimate the largest project they had implemented to date. Most students reported experience beyond small scripts, indicating familiarity with medium-sized or larger programming tasks. While this question does not directly capture coding skill, it provides a rough indication of practical programming experience among the surveyed students. Figure A.2 provides a detailed breakdown of the distribution of reported project sizes by study progress.

The majority of participants indicated that they use AI-based tools regularly, ranging from multiple times per week to daily usage. This suggests a generally high level of familiarity with AI systems among the surveyed students. Exact usage frequencies are depicted in Figure 5.1.

Finally, 13 respondents reported prior experience as tutors. This aspect is relevant for later analysis, as tutoring experience may influence expectations toward feedback quality, explanation style, and pedagogical behavior of an AI-tutor.

Overall, the sample primarily consists of students with prior programming experience and regular exposure to AI tools.

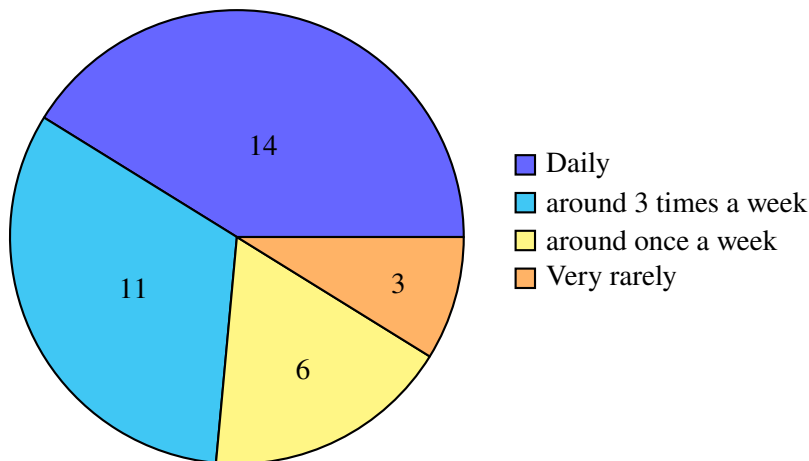


Figure 5.1: Frequency of AI tool usage among the surveyed students (n=34)

5.2 AI-tutors compared to human tutors regarding trust (RQ4.1)

This section analyzes the trust levels students have in AI-tutors compared to human tutors to answer research question RQ4.1: *To what extent are computer science students interested in using an AI-tutor, and how much do they trust its technical accuracy compared to a human tutor?*

5.2.1 Trust in human tutors

Participants were first asked to indicate how much they generally trust the technical accuracy of their human tutors answers to create a baseline to compare trust in the AI-tutor later. The results show a high level of trust in human tutors overall. Out of the 34 evaluated response sets, 24 participants indicated that they tend to trust their tutors answers, while 7 participants stated that they completely trust their tutor. Only 3 participants reported being undecided, and no participant expressed explicit distrust in the technical accuracy of human tutors. Results are visualized in Figure 5.2.

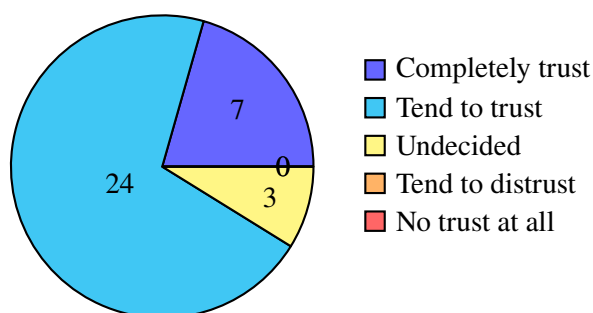


Figure 5.2: Baseline: Trust in human tutors among the surveyed students (n=34)

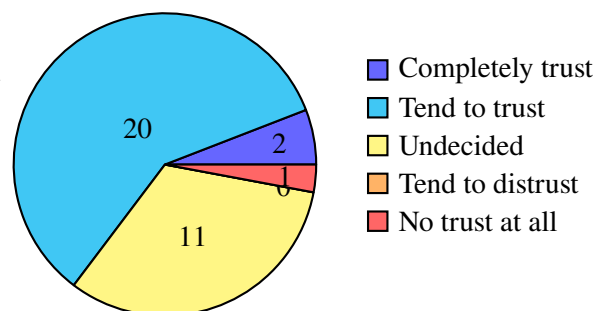


Figure 5.3: Trust in AI-tutors among the surveyed students (n=34)

These results suggest that human tutors are perceived as a highly reliable source of guidance by the surveyed students. Surveyed students generally have confidence in the technical accuracy of the assistance provided by their human tutors. These findings establish a benchmark for evaluating trust in AI-tutors in the subsequent section.

5.2.2 Trust in AI-tutors

To assess trust in AI-tutors under realistic usage conditions, participants were asked how much they would trust an automatically generated answer when it is supported by references to lecture slides or exercise solutions. This qualification was explicitly included to account for source transparency.

The responses reveal a more differentiated picture in trust levels than for human tutors. The majority of participants (20 out of 34) indicated that they would tend to trust such AI-generated answers. A smaller subset (2 participants) reported that they would completely trust the AI-generated response. At the same time, 11 participants stated that they were undecided, and 1 participant reported that they would not trust the answer at all. Results are visualized in Figure 5.3.

5.2.3 Correlation of trust levels between AI-tutors and human tutors

Comparing the trust levels in AI-tutors to those in human tutors reveals notable differences. Trust in human tutors is characterized by a strong concentration of positive responses and a lack of explicit distrust. In contrast, trust in AI-tutors appears more cautious, with a substantial portion of participants expressing uncertainty.

The results do not indicate a general rejection of AI-tutors. Instead, they suggest that trust in AI-tutors is present but more tentative compared to human tutors. The fact that most participants selected *tend to trust* rather than *completely trust* for AI-generated answers further supports the interpretation that AI-tutors are currently viewed as supportive but not authoritative sources of technical knowledge. Table 5.1 provides a cross-tabulation of trust in human tutors (rows) versus trust in AI-tutors (columns).

Table 5.1: Cross-tabulation of trust in human tutors (rows) versus trust in AI-tutors (columns)

Trust in Human Tutor (rows) / AI (columns)	Completely trust	Tend to trust	Undecided / Don't know	Tend to distrust	No trust at all	Total
Completely trust	1	4	2	0	0	7
Tend to trust	1	15	8	0	0	24
Undecided / Don't know	0	1	1	0	1	3
Tend to distrust	0	0	0	0	0	0
No trust at all	0	0	0	0	0	0
Total	2	20	11	0	1	34

Table 5.2 shows the relationship between the frequency of AI tool usage and trust in AI-generated answers. Interestingly, higher usage frequency appears to correlate with increased trust in AI-generated answers. For instance, among daily users of AI tools, 14 out of 19 participants indicated

that they would either completely trust or tend to trust AI-generated answers. In contrast, among those who reported very rare usage of AI tools, only 1 out of 3 participants expressed trust in AI-generated answers, while 2 participants were undecided or distrusted the answers.

Table 5.2: AI usage frequency versus trust in AI-generated answers

Trust in AI (rows) / Usage frequency (columns)	Daily	~3x/week	~1x/week	Very rarely	Total
Completely trust	1	1	0	0	2
Tend to trust	13	6	1	0	20
Undecided / Don't know	5	2	2	2	11
Tend to distrust	0	0	0	0	0
No trust at all	0	0	0	1	1
Total	19	9	3	3	34

5.2.4 Usage of AI-tutor for code feedback

To understand students' willingness to use an AI-tutor for code feedback, participants were asked how likely they would be to use such a tool if it were available. Table 5.3 shows the distribution of responses to this question.

Table 5.3: Willingness to use an AI-tutor for code feedback (n=34)

Response	Count	Percentage
Yes, definitely	19	55.88%
Probably yes	10	29.41%
Undecided / Don't know	2	5.88%
Probably no	2	5.88%
No, definitely not	1	2.94%
Total	34	100.00%

29 of the 34 participants indicated that they would be likely or very likely to use an AI-tutor for code feedback. This suggests a strong interest in leveraging AI-based assistance for programming tasks among the surveyed students.

5.2.5 Implications for AI-tutor design

The findings regarding trust in AI-tutors have several implications for the design of AI-based tutoring systems. First and foremost, the results show that there is no general distrust toward AI-tutors among software engineering students. Therefore a well-designed AI-tutor has the potential to be accepted as a supportive tool by students.

An AI-tutor are generally accepted as a supportive tool by students. AI-tutors should therefore focus on building and maintaining trust through transparency, reliability, and alignment with student expectations. Providing references to credible sources, such as lecture materials, can enhance trustworthiness.

5.3 Help seeking behavior (RQ4.2)

This section analyzes the help-seeking behavior of students to answer research question RQ4.2: *How do students currently seek help when working on programming tasks, and what role could an AI-tutor play within their existing help-seeking strategies?*

5.3.1 Reasons students seek help from tutors

Table 5.4 presents the reasons students reported for contacting human tutors. The results show that comprehension questions about content and feedback requests were the most common reasons, with 61.76% and 58.82% of participants respectively indicating these motivations. Notably, no participants reported contacting tutors for motivational problems, while 17.65% indicated that they never contacted a human tutor at all.

Table 5.4: Reasons for contacting human tutors (n=34)

Reason for human tutor consultation	Count	Frequency by Participant
I had comprehension questions about content	21	61.76%
I needed help with a task	9	26.47%
I wanted feedback on a solution / thought process	20	58.82%
I was uncertain about the content of a submission	10	29.41%
I was uncertain how to start	4	11.76%
I had motivational problems	0	0.00%
I never contacted a tutor	6	17.65%
Other:	5	14.71%

Additional written reasons

Besides the answers from Table 5.4 students were able to add additional reasons why they have already interacted with human tutors. 5 students wrote additional reasons. These included 2 answers that they asked for more information about a topic or tips with an assignment. One student once contacted the human tutor so he can clarify the assignment task. One student was unsure why he did not get points for the assignment while another one submitted the assignment too late. While an AI-tutor might be able to help with explaining why points were not given, it is unlikely that an AI-tutor can help with deadline issues since these are administrative problems.

5.3.2 Reasons students seek help from AI-tutors

Table 5.5 presents the reasons students reported for potentially using an AI-tutor. Comparing these results to human tutor consultations reveals notable differences. While comprehension questions remain equally common, students indicated increased willingness to seek help from an AI-tutor

5 Survey results

for tasks (+14 responses), feedback requests (+3 responses), and uncertainty about how to start (+12 responses). Interestingly, a small portion of students indicated they might use an AI-tutor for motivational problems, whereas no students reported contacting human tutors for this reason.

Compared to human tutors, AI-tutors appear to be considered particularly suitable for early problem solving. This suggests that students may perceive AI-tutors as more accessible or less intimidating for initial guidance.

Table 5.5: Reasons for potentially using an AI-tutor (n=34)

Reason for AI-tutor consultation	Count	difference compared to human	Frequency by Participant
I have comprehension questions about content	21	=	61.76%
I need help with a task	23	+14	67.65%
I want feedback on a solution / thought process	23	+3	67.65%
I am uncertain about the content of a submission	18	+8	52.94%
I am uncertain how to start	16	+12	47.06%
I have motivational problems	3	+3	8.82%
I would never use an AI-tutor	4	-2	11.76%

5.3.3 Preferred help sources ranking

Students working on programming tasks often have multiple help sources available. To understand how students currently use existing AI tools within their guidance seeking strategies, participants were asked to rank four help sources based on their preference when encountering difficulties with programming tasks. Table 5.6 shows the results of this ranking.

Consulting lecture materials and independent problem solving ranked highest overall with an average rank of 2.118. 13 participants placed this help source in the first position while only 4 students placed it last, indicating a strong preference for self-directed learning. Asking peers / fellow students ranked slightly higher than AI chatbots, with an average rank of 2.382 and 2.441 respectively. Human tutors however ranked lowest on average with an average rank of 3.059, with 20 participants placing tutors in the fourth position.

This indicates that human tutors are currently not the first choice for students when seeking help with programming tasks even though they are highly trusted as shown in Section 5.2.

Notably, AI chatbots ranked in the middle (2.441 with 2.5 as total average). This indicates that AI-based help is already well-integrated into students' help-seeking strategies and could be further leveraged by AI-tutors.

5.3.4 Implications for AI-tutor design

Compared to human tutors, AI-tutors appear suited for early problem solving, such as getting started or validating partial progress. This implies that the AI-tutor should offer guidance for understanding code assignments and provide feedback on solutions or thought processes. The relatively high

Table 5.6: Ranking of preferred help sources (n=34)

Help Source	Average Rank	Overall Rank	1st	2nd	3rd	4th
Consult lecture materials and try independently	2.118	1	38.24%	23.53%	26.47%	11.76%
Peers / fellow students	2.382	2	26.47%	17.65%	47.06%	8.82%
AI chatbot (e.g., ChatGPT)	2.441	3	17.65%	41.18%	20.59%	20.59%
Human tutor	3.059	4	17.65%	17.65%	5.88%	58.82%

ranking of AI chatbots among help sources indicates that students are already comfortable using AI tools for assistance. Integrating an AI-tutor could therefore help to support students in their guide seeking strategies effectively.

5.4 Proactive Feedback (RQ4.3)

This section analyzes how students would perceive proactive feedback to answer research question RQ4.3: *How do students evaluate proactive assistance from an AI-tutor, particularly in situations of frustration or difficulty?*

5.4.1 Students actively asking for help

To understand the need for proactive help from an AI-tutor, students were first asked how often they actively seek help when working on programming tasks. Results indicate that a big portion of students (16 out of 34) only rarely or never seek help when facing difficulties. A slightly smaller portion (13 out of 34) reported that they often or always seek help while 5 students indicated that they sometimes seek help. These results suggest that a significant number of students may not actively seek assistance even when they encounter challenges. Figure 5.4 visualizes the distribution of help-seeking frequency among the surveyed students.

5.4.2 Reaction to proactive explanations from an AI-tutor

To evaluate proactive assistance from an AI-tutor it is also important to look at potential risks of such a feature. Students were therefore asked how they would feel if an AI-tutor would proactively offer explanations when they have already understood the topic.

Only a small portion of students (6) answered they would find such proactive explanations annoying or patronizing (4) or would prefer not to receive them at all (2). The majority of students indicated they would either find such explanations helpful and supportive (5), take note of them but not be bothered (9), or find them unnecessary but not problematic (14). Figure 5.5 visualizes these results.

These findings suggest that unnecessary proactive explanations when the student already understands the topic are generally tolerated by students. Therefore, AI-tutors could consider implementing proactive assistance features without a high risk of negative reception from students.

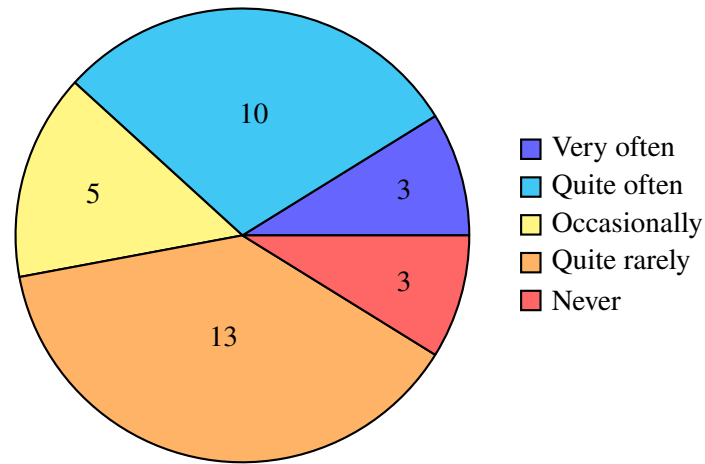


Figure 5.4: Frequency of actively seeking help when encountering programming problems (n=34)

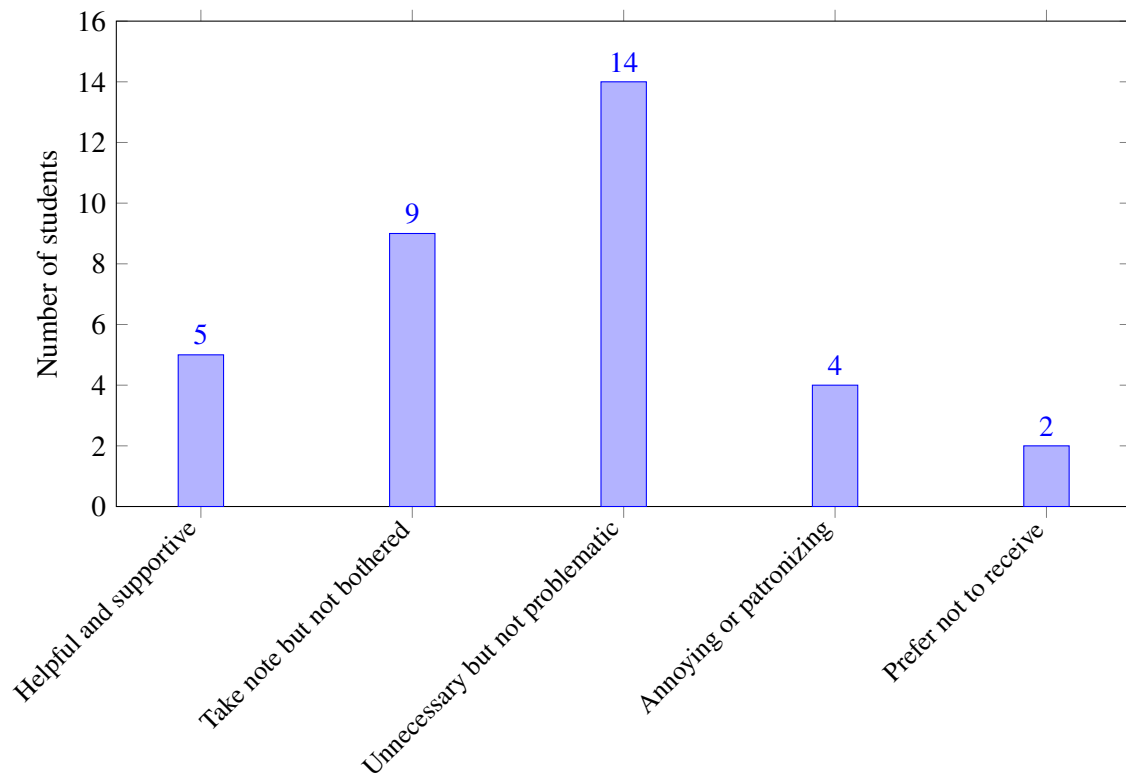


Figure 5.5: Student attitudes toward proactive explanations when the topic is already fully understood (n=34)

5.4.3 Proactive tips from an AI-tutor

To evaluate proactive assistance from an AI-tutor it is also important to look at potential risks of such a feature. Students were therefore asked how they would feel if an AI-tutor would proactively give a tip or hint which anticipate a central part of a task they are working on independently.

Students reactions were quite clear on this topic. Only a small portion of students (8) answered they would find such proactive hints helpful (2) or rather helpful (6). The majority of students indicated they would either be neutral (3), find such hints rather disturbing (16), or very disturbing (7).

Figure 5.6 visualizes these results.

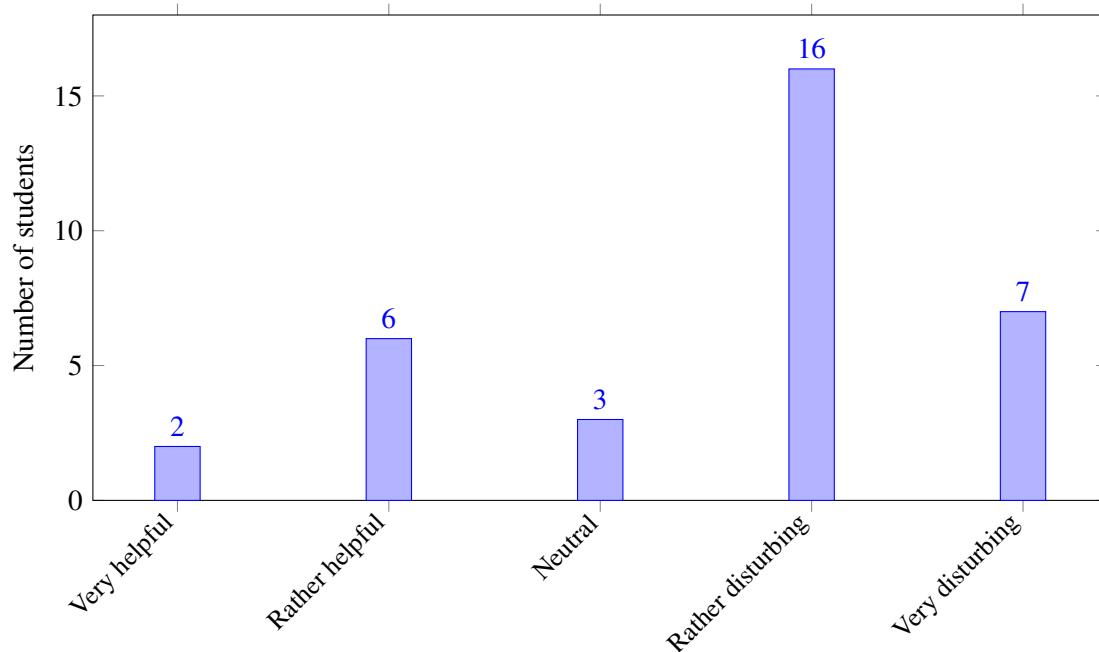


Figure 5.6: Student reactions to unsolicited hints during independent problem solving (n=34)

These results indicate that most students prefer to maintain autonomy when working on programming tasks. Hints that preemptively reveal key aspects of a task are generally perceived as disruptive. This suggests that AI-tutors should not provide unsolicited hints too easily during independent problem solving to avoid negatively impacting the learning experience.

5.4.4 Implications for AI-tutor design

First and foremost, the results indicate that a significant portion of students do not actively seek help when facing difficulties. This could result in more students benefiting from proactive assistance features from an AI-tutor. However, it is also crucial to consider the potential negative reception of proactive feedback. While proactive explanations when the student already understands the topic are generally tolerated, giving unsolicited hints could be perceived as disruptive. AI-tutors should therefore be designed to balance proactive feedback carefully to avoid negatively impacting the learning experience of the student.

5.5 Motivational Support (RQ4.4)

This section analyzes how students would perceive motivational messages or messages with motivational touch to answer research question RQ4.4: *How do students feel about receiving empathetic or motivational responses from an AI-tutor?*

To understand students preferences regarding motivational feedback inside AI-tutors responses, a Kano Model-style analysis was conducted. Question 16 presented a scenario where the AI-tutor's response begins with motivational feedback while Question 17 presented the same scenario but without such motivational elements.

Table 5.7 presents a cross-tabulation of responses to these two questions, enabling a Kano Model-style analysis of student preferences. The rows represent responses to Question 16 (with motivational feedback), and the columns represent responses to Question 17 (without motivational feedback).

Table 5.7: Cross-tabulation of attitudes toward motivational feedback: with motivational feedback (Q16, rows) versus without motivational feedback (Q17, columns)

With motivational feedback (Q16) rows / Without (Q17) columns	Excellent	Expect that	Acceptable	Can accept that	Bad	Total
I think that is excellent	0	0	7	3	1	11
I expect that	0	1	0	1	0	2
I think that is acceptable	0	0	7	1	0	8
I can accept that	0	1	3	1	1	6
I think that is bad	1	2	0	0	4	7
Total	1	4	17	6	6	34

Analyzing this table reveals a few peaks.

First, 4 students were unhappy with motivational feedback, rating it as bad while rating the non-motivational response as bad as well. This indicates a contradiction in their preferences.

Another peak can be seen rating motivational feedback as excellent while rating non-motivational feedback as acceptable (7 students). This indicates that motivational feedback might be an excitement factor for these students.

Negative reactions to motivational feedback are also present, with 3 students rating motivational feedback as bad while rating non-motivational feedback as acceptable or better.

In total, 11 students rated motivational feedback as excellent while only 1 student rated non-motivational feedback as excellent. To conclude, motivational feedback seems to be more often seen as a positive addition rather than a negative one.

5.5.1 Implications for AI-tutor design

These findings suggest that motivational feedback should be considered as a potential feature in AI-tutor design. It might enhance the user experience for students who appreciate such elements. However, it should also be possible for a student to disable motivational feedback if they find it distracting or unhelpful.

5.6 No code submissions (RQ4.5)

This section analyzes how students would perceive if the AI-tutor does not answer with code at all and only helps with guidance what to improve inside the code to answer RQ4.5: *How do students judge stylistic or conceptual feedback that does not include exact code solutions?*

Research suggests that students receiving solutions for programming tasks might not learn as effectively as students who receive guidance on improving their code without being given exact solutions as shown in chapter 2. To ensure that an AI-tutor never provides exact code solutions, it might be possible to force the AI-tutor to only give stylistic or conceptual feedback without providing code snippets. To evaluate how students perceive this concept, students were asked how they would feel if an AI-tutor never provided exact code solutions but only guidance on how to improve their code. Table 5.8 provides a detailed breakdown of these responses.

Table 5.8: Student attitudes toward receiving stylistic feedback without exact code solutions (n=34)

Response	Count	Frequency
Very good - I would like to work out the solution myself	8	23.53%
Rather good - a hint is sufficient for me	7	20.59%
Neutral - it depends on the context	14	41.18%
Rather poor - I would prefer an example illustrating such a style	5	14.71%
Very poor - I want to see the complete solution	0	0.00%
Total	34	100.00%

The results indicate generally positive reactions towards receiving stylistic feedback without exact code solutions. While 15 students expressed positive attitudes (very good or rather good), only 5 students expressed negative attitudes (rather poor) and no student expressed very poor attitudes. The largest portion of students (14) expressed to be neutral in this regard.

5.6.1 Implications for AI-tutor design

The lack of negative reactions suggest that it is not perceived as bad if an AI-tutor never provides exact code solutions. Therefore, forcing an AI-tutor to never under any circumstances provide exact code solutions could be a viable design choice to enhance learning outcomes without significantly impacting student satisfaction.

6 Design of the AI-tutor

This chapter presents the design of the AI tutoring component developed for the existing gamified learning platform MEITREX. The goal of the system is to support students in learning software engineering concepts more effectively and to enhance their motivation through adaptive interaction and personalized gamification.

The design process focused on integrating the AI-tutor seamlessly into the learning workflow and aligning its behavior with the motivational needs of different player types. To explore these aspects, eight distinct interaction scenarios were designed, each representing a specific learning situation and the corresponding tutor behavior. These scenarios are illustrated in the diagrams presented in this section.

6.1 Design Goals

Chapter 2 and chapter 5 established several constraints and design choices. This section summarizes them briefly.

Section 5.2 and section 5.3 showed that students are generally open to receiving feedback from an AI-tutor. Receiving feedback about written code was also seen as helpful by most students (section 5.2). Providing references to lecture material and sources was also seen as helpful (section 5.2). Depending on the user query, it might be possible to receive many fitting sources from the lecture materials. While using every source might be beneficial for the AI-tutor to give a better and correct answer, it could also lead to overwhelming the student with too many sources. Therefore, the AI-tutor should provide references to lecture materials and sources, but limit the number of sources to a maximum value per answer.

Proactive feedback can be an option to improve the learning experience for students (section 5.4). However, while giving explanations to students proactively seems to be generally accepted even if the student already understood the topic (section 5.4), giving hints could be perceived as disruptive (section 5.4). The AI-tutor should give proactive feedback, but should not trigger too often.

Motivational and empathetic feedback is a feature enhancing the user experience for students (section 5.4). Literature research 2 indicates if a student receives solutions for an assignment, it might negatively impact the learning progress of the student. The AI-tutor should never give exact solutions to an assignment even if the student asks directly for it. One possible way to enforce this is if the AI-tutor never answers with code like *Iris, the AI-tutor in Artemis* [BFK24] does. RQ4.5 in section 5.6 indicates that students do not perceive never getting any code inside of the feedback negatively for student satisfaction.

Additionally, it should be possible for the AI-tutor to work on follow-up questions by the student. Since context aware dialogue is essential in human interaction, an AI should maintain context to appear humanlike [Tur07]. Therefore, it should be implemented to give the AI-tutor information about the last few question response pairs to increase student satisfaction.

6.2 AI-tutor functionality and already existing workflows

6.2.1 Existing architecture

Before starting this thesis, an AI-tutor prototype already existed inside of MEITREX.

However, the AI-tutor was only able to answer questions and generate hints. Additionally, the AI-tutor did not adapt to the student personality or skill level nor was able to use context from previous interactions with the student. Consequently, the previous AI-tutor could neither answer follow-up questions correctly nor provide proactive feedback.

***DocProcAI* Automated Course Content Processing**

DocProcAI is a pipeline designed to connect generic LLMs with course-specific knowledge. This is achieved by structuring different teaching materials like lecture slides, lecture videos or other type of content into a structured knowledge base. The goal is to link educational resources to enable an AI-tutor to provide answers according to the lecture and to provide these sources to the student after answering.

The pipeline begins by splitting up larger files into smaller parts. The audio of lecture videos gets transcribed and visual changes are detected. Each smaller part gets a fitting title and representative keyword tags. Finally, *DocProcAI* matches slide images to video timestamps using visual template matching, allowing students to jump to the exact moment a slide is discussed. Assessments are linked to relevant materials based on the semantic similarity of their text embeddings. The result is a structured semantic index that allows an AI-tutor to answer questions by retrieving and referencing specific parts, such as a video timestamp or slide number, from the lecture materials. This way, responses are verifiable and strictly tied to the lecture materials.

Previous workflow

There were 2 possible ways to interact with the AI-tutor: To write a question into a text field or to request a hint for an assignment.

The workflow for a question is as follows: First, the question gets classified into different categories. These include: A question about lecture materials, a question about the MEITREX system, a question labeled as *other* and an unrecognizable question. However, only questions about the lecture materials get progressed further.

Second, a semantic search gets conducted using the student request and the *DocProcAI* pipeline. Afterwards, a prompt gets engineered with the retrieved results and finally the result returned to the user.

The workflow for a hint request is similar, with the exception that no classification of the question is needed.

6.2.2 Overview of new AI-tutor functionality

The AI-tutor will provide feedback to students in different situations. These can be divided into two main categories: reactive feedback and proactive feedback.

Reactive feedback is triggered after something the student did with intent to interact with the AI-tutor. This includes asking a question to the AI-tutor or requesting a hint for an assessment, because the student actively wants to communicate with the AI-tutor. In contrast, proactive feedback is triggered by the backend without any user intent. However, the proactive feedback is still related to something the student recently did. The following sections will go into detail about each of these two main categories and their different scenarios.

6.3 Reactive Feedback

When a user writes a text to the AI-tutor, the message first gets classified. It should be differentiated between questions about the lecture, questions about written code and follow-up questions. A question about the MEITREX system is currently not in scope of this thesis and will therefore not be implemented and therefore not answered by the AI-tutor. An unrecognizable question will lead to a message to the student that the question could not be understood. However, if the student asks for assessment hints 6.3.4, the question does not be classified further, because a hint request is differentiable from other questions by the fact that the student requests a hint by pressing a button and not by writing a question himself.

Once classified, relevant materials get retrieved. Additionally, the AI-tutor changes its behavior accordingly to the student. However, the student is able to save preferences to get treated differently to optimise personalization. For reactive feedback, chat history will not only be used for follow-up questions 6.3.3, but also for questions about lecture materials 6.3.1 and questions about code feedback 6.3.2. Once everything is gathered, the prompt gets engineered, send to the LLM and the result returned to the student.

6.3.1 Student asks a question about lecture materials

Previously, the AI-tutor was able to answer questions about lecture materials, similar to this scenario. The addition to this scenario is that the AI-tutor will adapt its behavior according to the skill level of the student.

A very standard use-case for an AI-tutor would be the student having a question about course materials. These questions could be about anything regarding the course materials.

Figure 6.1 shows the workflow of the AI-tutor for such a comprehension question on context. Once the question gets classified as a question of this type, the required context to answer the question gets retrieved. To answer the question according to lecture materials, these materials need to be retrieved first according to the student question. The AI-tutor could need different sources for each

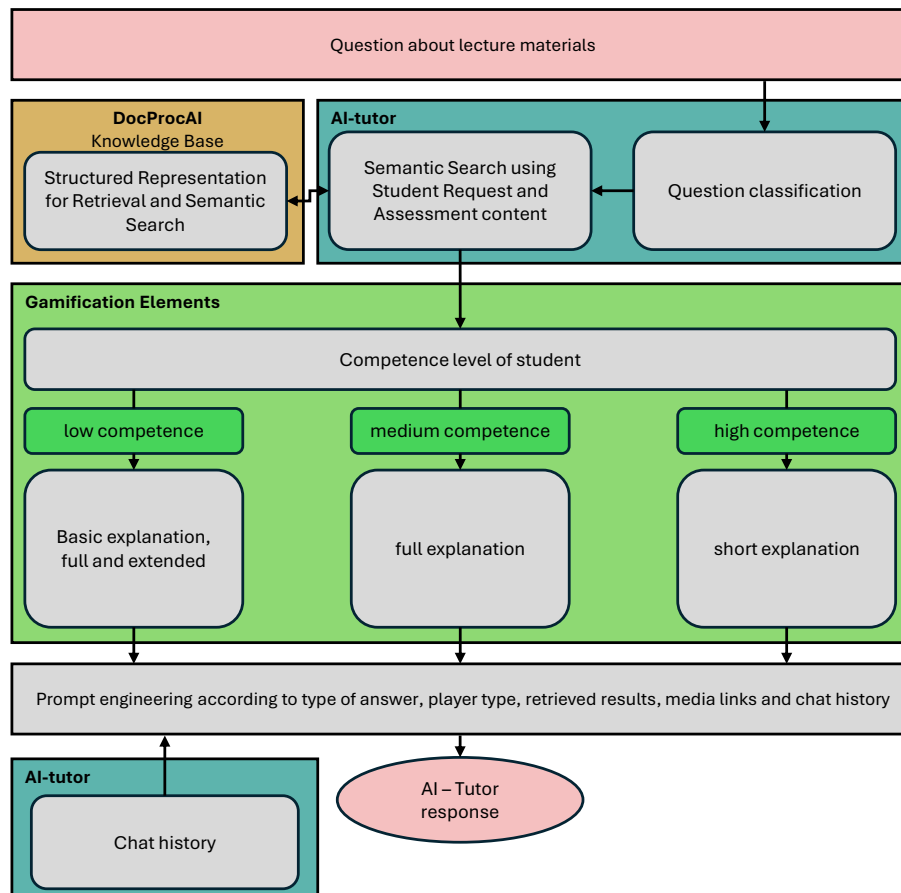


Figure 6.1: Student asks a question about lecture materials

different question. Therefore, the already existing *DocProcAI* knowledge base [MEI25] gets used to retrieve fitting sources from the course materials. This could include lecture slides, lecture videos or transcripts of videos.

To minimize the risk of giving an explanation with too much or too little information for the user, the prompt gets changed according to the competence level of the student. There are multiple ways to determine the skill of a student on a given topic. To calculate the competence level of a student, the average of each competence level of the student for each topic the question is about could be calculated and used. According to this competence level, the AI-tutor can react differently.

If the student is relatively new in the course and the understanding of a given topic is small, a basic explanation can be given with a full explanation. It will not be assumed the student got any prior knowledge about the topic of the question he asked. This way, the student does not get overwhelmed by an explanation.

If the student possesses substantial prior knowledge and experience in the respective topic area, it is beneficial to not explain too much to him he already understood 6.1. To ensure this, the AI-tutor will answer with a short but useful answer. If the student needs more context, he might ask again.

If the student is in between these two levels, a balanced explanation will be given. This explanation will not be too low level, but also not too complex.

Once the student specific prompt part is engineered, the final prompt will be created. However, the student might have a chat history with the AI-tutor already. If this is the case, the last few question-response pairs will be added to the prompt to give the AI-tutor more context about the current state of the context. Finally, the prompt will be engineered according to the retrieved sources, the student competence level, player type, the chat history (if available) and the student question. The final prompt will then be sent to the LLM to generate an answer which will be returned to the student.

6.3.2 Student asks for code feedback

Since the focus is on tutoring in software engineering education, it is important to offer feedback for code written by the student. Earlier versions of the AI-tutor did not provide feedback on written source code. This scenario builds upon the existing mechanism for retrieving course materials and enhances it by generating feedback that is adapted to the student's player type.

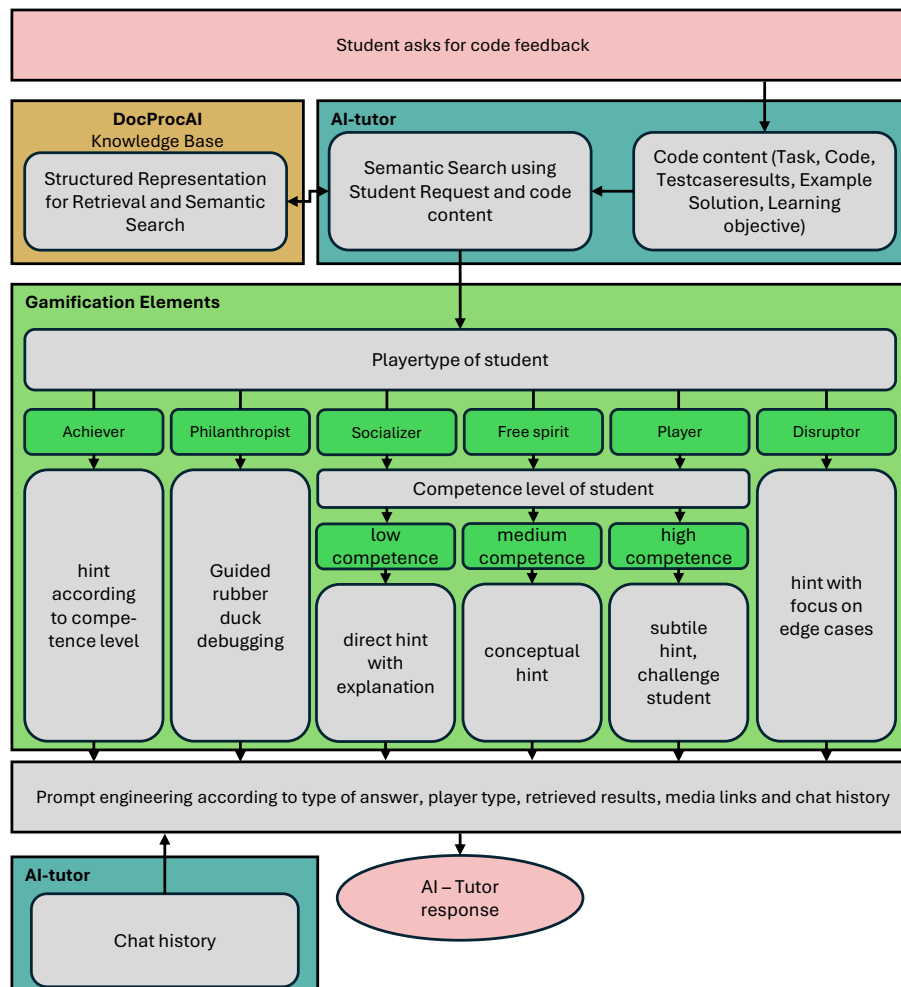


Figure 6.2: Student-initiated code feedback scenario

Figure 6.2 shows the workflow of the AI-tutor for a question where the student asks for feedback about written code. Once the question gets classified as a question of this type, the AI-tutor gathers information about the code the student is currently working on. The AI-tutor could need different sources for each different questions or written code. Therefore, the already existing *DocProcAI* knowledge base [MEI25] gets used to retrieve fitting sources from the course materials. This could include lecture slides, lecture videos, transcripts of videos or code style guides.

The player type of the student will also influence the behavior of the AI-tutor.

As an achiever is primarily concerned with mastery, the AI-tutor adapts its responses based on the user's skill level. If the student is relatively new in the course and the understanding of a given topic is small, a basic explanation can be given with a full explanation. It will not be assumed the student got any prior knowledge about the topic of the question he asked. This way, the student does not get overwhelmed by an explanation. If the student is relatively good and experienced in the topic, it is beneficial to avoid over-explaining concepts the student has already mastered 6.1. To ensure this, the AI-tutor will answer with a short but useful answer. If the student needs more context, he might ask again. If the student is in between these two levels, a balanced explanation will be given. This explanation will not be too low level, but also not too complex.

A philanthropist enjoys helping others. In software engineering, there is a way to debug code called *rubber duck debugging* [HT99]. In this type of debugging, the programmer will explain the code he wrote to someone else, especially to someone with no knowledge of programming. This way, it is possible for the programmer to notice inconsistencies by thinking about the code on a basic level. The AI-tutor encourages this type of debugging for the philanthropist, because it aligns with their intrinsic motivation to help others. It should be noted that the effectiveness of this approach does not imply that students with philanthropic tendencies experience the same motivational satisfaction when interacting with an AI as they do when helping other humans. The AI-tutor will notice the error the student has and then suggest the student to explain that section of faulty code to either another student nearby or to the AI-tutor. This way, the student might notice his error.

A disruptor however might want to break a system. Therefore, he will receive a tip with additional focus on edge cases he might have overlooked or enjoys handling.

A socializer, a free spirit and a player will not receive any special treatment in this situation. The AI-tutor will not get any player type specific instructions and answers with a tip according to the question. Reasoning for this decision is that there is not really a social aspect to add for the socializer. A free spirit might explore on his own. Additionally, there is no suggestion with extrinsic rewards possible to catch the player.

Once the student specific prompt part is engineered, the final prompt will be created. However, the student might have a chat history with the AI-tutor already. If this is the case, the last few question-response pairs will be added to the prompt to give the AI-tutor more context regarding the current state of the conversation. Finally, the prompt will be engineered according to the retrieved sources, the student competence level, player type, the chat history (if available) and the student question. The final prompt will then be sent to the LLM to generate an answer which will be returned to the student.

6.3.3 Follow-up question from student

Prior to this thesis, the AI-tutor lacked support for handling follow-up questions. As the ability to respond to such questions is essential for a positive user experience 6.1, this scenario extends the existing functionality for retrieving relevant course materials by introducing skill level dependent adaptations in the tutor's responses.

An answer alone might not be enough when interacting with the AI-tutor. The student could get a new thought process where he finds another question or he might also have problems understanding the explanation of the AI-tutor correctly. If this is the case, the question might not be recognizable as a lecture question 6.3.1 or a question about code feedback 6.3.2. Therefore, the question might get classified as follow-up question. However, because all other reactive questionhandling is already using chat history, this workflow gets only triggered if it is unknown what the student question is about.

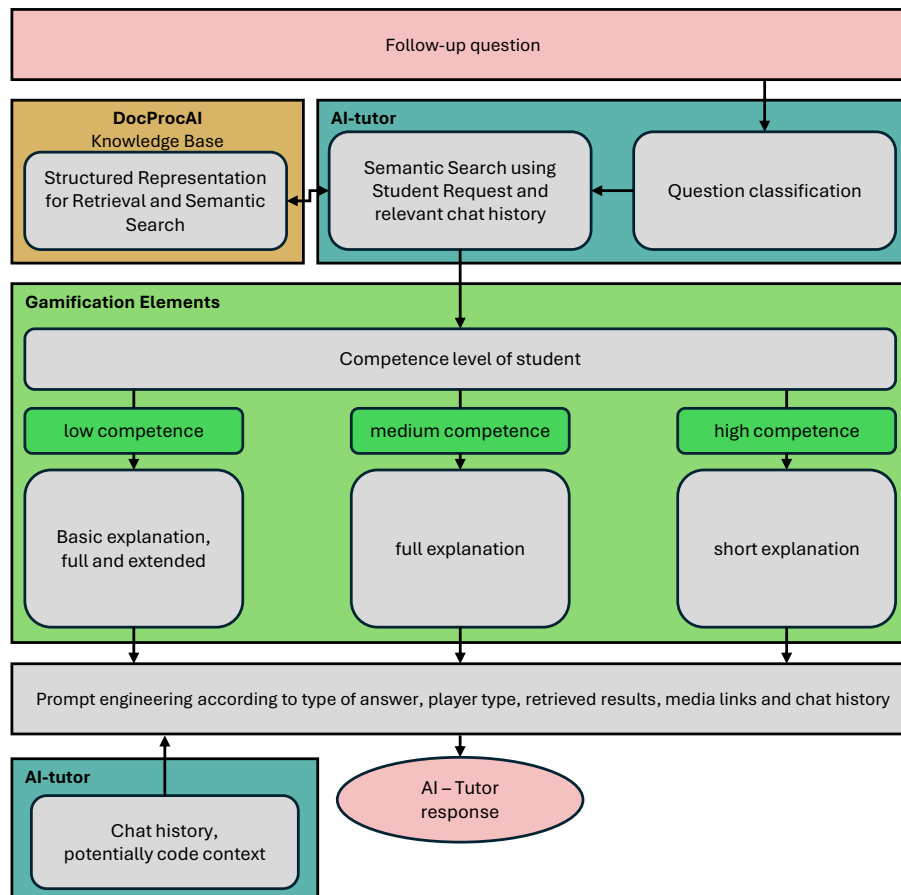


Figure 6.3: Follow-up interaction flow

An example could be: *"Can you explain that to me in different words, please?"*. This query is unable to be categorised correctly only by the question alone but has to be handled according to the context of the chat history the AI-tutor has with the student.

Therefore, a follow-up question always requires a chat history. Without a chat history a follow-up question is pointless. Therefore, a follow-up question should only trigger if and only if there is a chat history. The workflow for a follow-up question is a combination of a lecture question 6.3.1 and described in figure 6.3.

The AI-tutor will retrieve relevant materials according to the student question and the relevant chat history. According to the competence level of the student, the AI-tutor will change its behavior exactly like in 6.3.1. Importantly, the chat history will always be added to the prompt to give the AI-tutor more context about the current state of the context. Finally, the prompt will be engineered according to the retrieved sources, the student competence level, player type, the chat history and the student question.

6.3.4 Assessment hints

Previously, the AI-tutor was capable of providing hints for assessments. However, these hints were not adapted to the student's assigned player type. The current design addresses this limitation by incorporating player type specific behavior into the hint generation process.

While an assignment hint is still a reactive answer to a student request, it is inherently different to the other types of reactive feedback discussed until now, because in an assessment hint, the student does not type a message himself. The student is able to request a hint for an assessment, without writing a request himself but by pressing a button. This way, the AI-tutor will provide a hint on the content of the assessment itself.

Figure 6.3.4 shows the workflow of an assessment hint. Notably, the AI-tutor will not classify the question, since the type of request is already known. The AI-tutor starts by gathering all relevant content of the assessment to start a semantic search.

The hint will be changed according to the player type of the student. The skill level of the student will not be taken into account. The reason for this is that the primary method of calculating a given skill of a given user in MEITREX is how he performs in the assessments related to a given skill. Therefore, the specific skill which would be used for a skill level based hint, is most likely not calculated at that point and most likely undefined. This would not give any advantage.

An achiever is motivated through mastery as shown in 2.6. He would thrive with mastering the challenge himself. Therefore, he gets a smaller tip.

A socializer is motivated through social interactions. Currently, there exists only one way to interact with fellow students inside of MEITREX, which is the forum. Therefore, the AI-tutor searches the forum for questions to the content of the requested hint. If there exists one, the AI-tutor will link to the question in the forum. The student will then gather knowledge from the forum or the student has the opportunity to answer a question about the content. The AI-tutor will still provide a hint and not just link the forum, which will not focus on anything specific.

A philanthropist gets handled similar to a socializer. The philanthropist has the option to answer a question on the forum to the content, so he gains motivation through helping others.

The free spirit gets a tip according to the baseline. The tip does not get changed according to the player type.

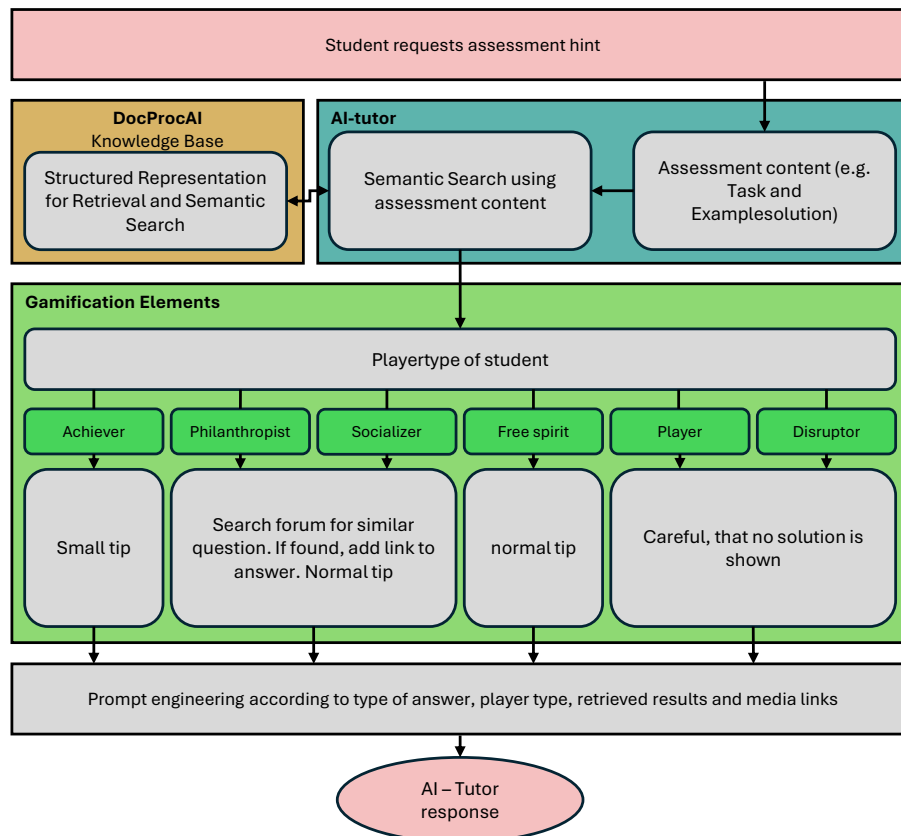


Figure 6.4: Assessment hints interaction scenario

A disruptor and a player are not intrinsic motivated. A player seeks extrinsic rewards for his actions. In the current iteration of MEITREX, the student receives rewards for completing quizzes correctly. Therefore, the AI-tutor will provide a very defensive hint to not spoil the assessment for the student. A disruptor might want to break the system and therefore might want to get the solution for the assessment as well. Therefore, the AI-tutor will also provide a very defensive hint to not spoil the assessment for the student.

6.4 Proactive Feedback

As we showed as answer to RQ4.3 in section 5.4, students are open to proactive feedback from an AI-tutor. Therefore, the AI-tutor will also provide proactive feedback in certain situations. However, it is important to not trigger too often 6.1.

Currently, four different proactive feedback scenarios are designed. First, the student might get proactive code feedback if he fails a testcase multiple times 6.4.1. The student might be stuck and frustrated and therefore could benefit from a proactive tip to get unstuck. Second, the student gets proactive feedback after finishing a test 6.4.2. This encourages the student to reflect on his performance. Third, the student might get proactive feedback after completing a part of the lecture 6.4.3. This will allow the student to reflect on what he learned so far or to motivate him to continue

learning. Fourth, the student might get proactive feedback after being actively idle for a long time 6.4.4. The student might not know on what to do next and therefore could benefit from the AI-tutor suggesting him something to do.

While proactive feedback is a new functionality of the AI-tutor, it will still use the already existing *DocProcAI* knowledge base.

6.4.1 Proactive: Code feedback, no student question, testcases fail for n-th time

Sometimes, someone might get stuck while writing code. To combat this situation, the AI-tutor will proactively provide code feedback. However, giving hints could be perceived as disruptive (section 5.4). Therefore, the AI-tutor should not provide proactive code feedback too often and only if there is a high chance the student is stuck. To achieve this, the AI-tutor will save how often a student failed a testcase for a given exercise. Once the student failed a testcase enough times, the AI-tutor will trigger and provide proactive code feedback. Figure 6.5 shows the workflow of the AI-tutor for this situation.

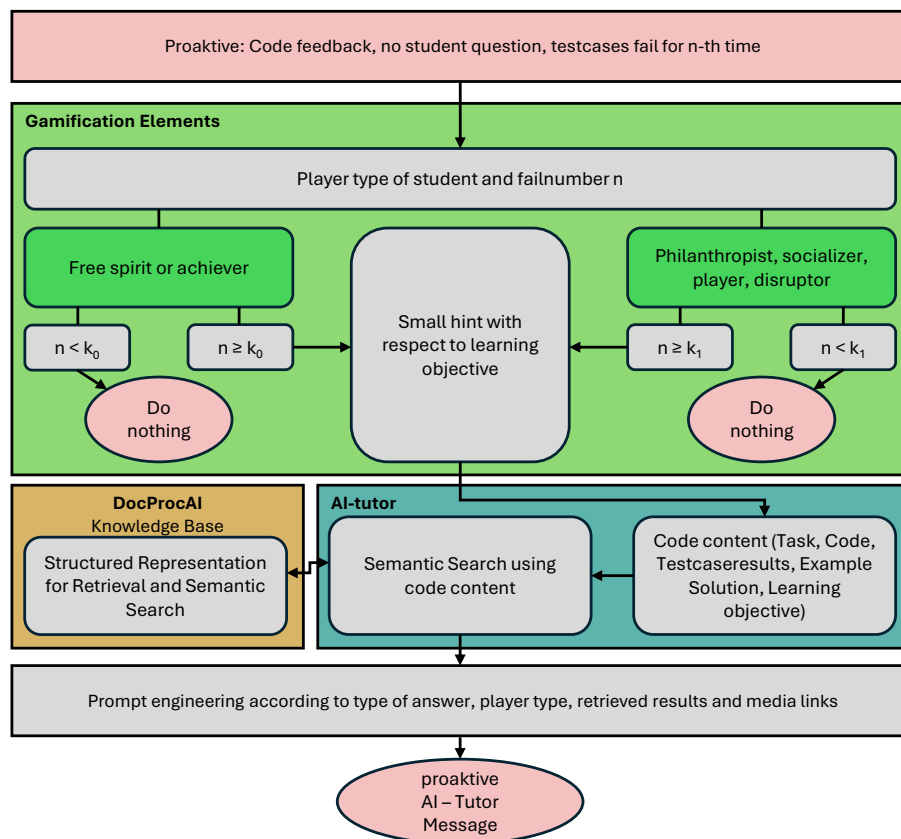


Figure 6.5: Proactive code feedback after repeated test failures

The player type of the student will not influence the behavior of the AI-tutor in this situation, but how fast the AI-tutor triggers. An achiever is mostly concerned about mastering skills and therefore might get more annoyed if the AI-tutor triggers too early. Similarly, a free spirit might want to explore the system on his own and therefore might also get annoyed if the AI-tutor triggers too early.

Therefore, these two player types will get the AI-tutor triggering after more failed testcases than the other player types. Once the AI-tutor decides to provide proactive code feedback, the AI-tutor will gather information about the code the student is currently working on. This includes the code itself, the exercise description and the testcases. Additionally, if an example solution for the exercise exists, it will also be gathered as well as the learning objective of the exercise. Afterwards, a semantic search on course materials will be started to retrieve relevant sources. Afterwards, the prompt gets engineered and the student receives the result proactively.

6.4.2 Proactive: Test done, no student question

In MEITREX, students are able to complete tests to check their knowledge. The AI-tutor will proactively interact with the student once he finished a test. Figure 6.6 shows the workflow of the AI-tutor for this situation.

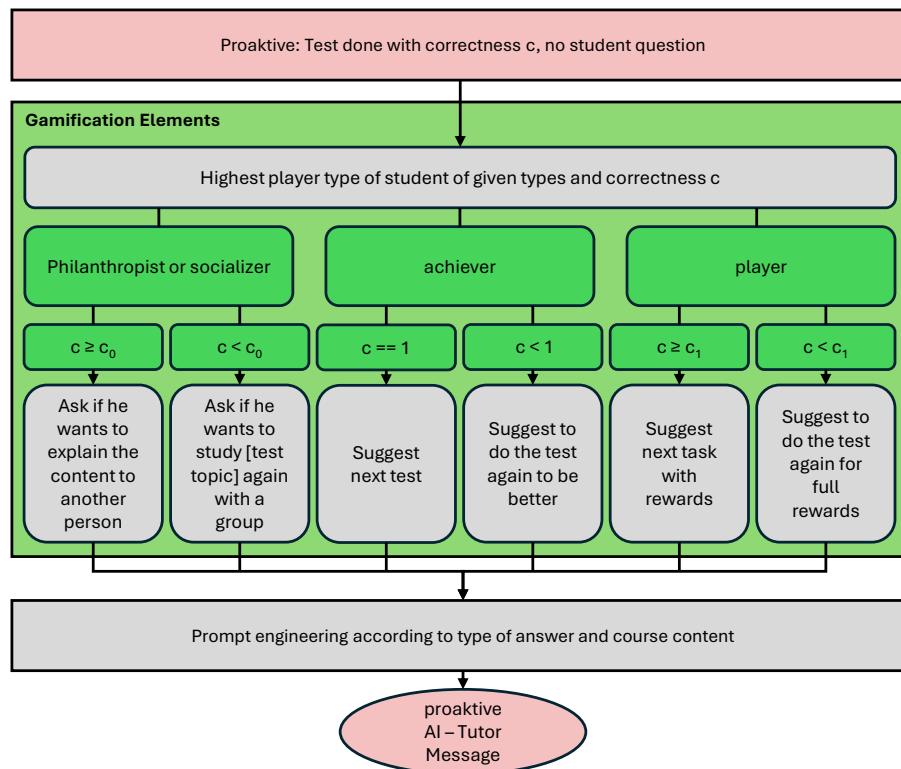


Figure 6.6: Proactive feedback after test completion

First, the AI-tutor retrieves information about the test the student just completed. This includes the correctness of the test and which questions the student answered correctly or incorrectly. The proactive message will be changed according to the player type of the student. An achiever is mostly concerned about mastering skills. He is intrinsically motivated to master skills. If the student did not perform well on the test, the AI-tutor knows that the student did not master the skills behind the test. Therefore, the AI-tutor will suggest the student to review his mistakes, to improve his skills

about the topic of the test and then to retake the test. However, if the student performed well on the test, the AI-tutor will congratulate the student and suggest him the next test he could take to further master his skills.

A socializer or a philanthropist is intrinsically motivated through social interactions or helping others. The AI-tutor will check the correctness of the student test. If the student performed better than the threshold, the AI-tutor will suggest the student to explain an open question on the forum about the content of the test he just completed. If the student performed worse than the threshold, the AI-tutor will suggest the student to study the contents of the test again with a group, to improve his knowledge. A player is extrinsically motivated by getting rewards. Therefore, the AI-tutor will suggest the student to retake the test if he did not earn full rewards yet. However, if the student earned full rewards, the AI-tutor will suggest the next task where the student can earn rewards.

Afterwards, the prompt gets engineered and the student receives the result proactively.

6.4.3 Proactive: Lecture milestone done

In MEITREX, the course is divided into different lecture milestones. A lecture milestone contains lecture slides, videos and exercises. Once a student completes a lecture milestone, the student might not know on what to do next. Therefore, the AI-tutor will proactively interact with the student. Figure 6.7 shows the workflow of the AI-tutor for this situation.

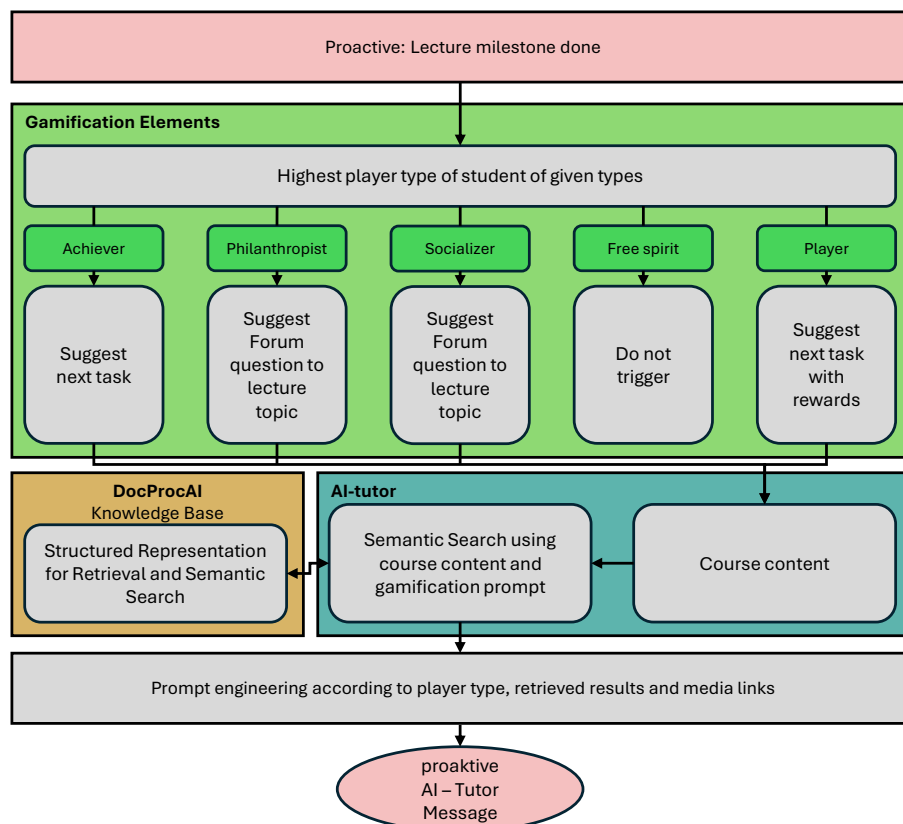


Figure 6.7: Proactive interaction after lecture milestone completion

The proactive message will be changed according to the player type of the student. An achiever is mostly concerned about mastering skills. However, it might be possible, that the student completed the lecture milestone without mastering the skills behind it. Therefore, the AI-tutor suggests the student to do an activity which helps him to master the skills of this lecture milestone. For example, this could be an assessment where the student did not perform well on. A socializer or a philanthropist will be suggested to answer questions on the forum fitting the content of the lecture milestone he has just completed to help fellow students. A player is extrinsically motivated by getting rewards. Therefore, the AI-tutor suggests the student to do an activity where he did not earn full rewards yet.

Free spirits however prefer to explore in their own way. That is the reason for the AI-tutor to not suggest them something else to do. Disruptors are not handled specially. If the main player type is a disruptor, the student will be handled according to the second highest player type.

6.4.4 Proactive: Long idle (active clicking and not starting anything)

It might happen that a student does not know what to do next. The student might then click around randomly, but will not start any activity. To help the student in this situation, the AI-tutor will proactively suggest the student something to do next.

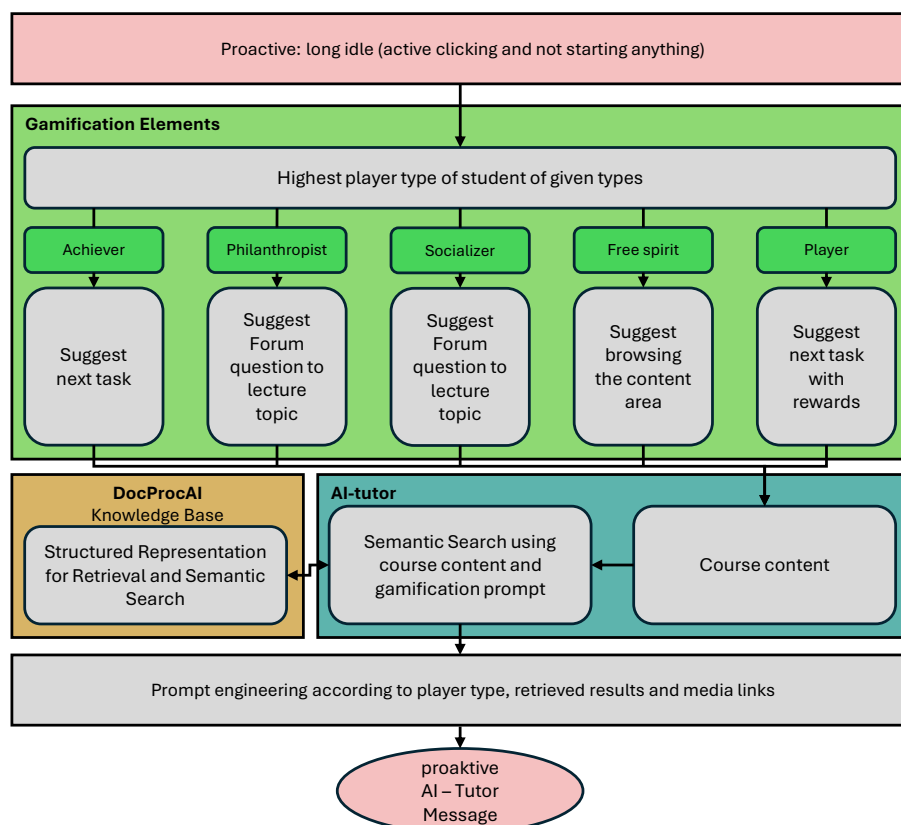


Figure 6.8: Proactive interaction during long idle periods

Figure 6.8 shows the workflow of the AI-tutor for this situation. The suggestion will be changed according to the player type of the student. This way, the AI-tutor can suggest something which motivates the student.

An achiever will be suggested to do an activity which helps him to master skills. This could be an assessment or a challenging exercise. A socializer will be suggested to interact with fellow students. Currently, the only way to interact with fellow students inside of MEITREX is the forum. Therefore, the AI-tutor suggests the student to check the forum for questions to answer or to ask a question himself. A philanthropist will be suggested to help others. The AI-tutor suggests the student to check the forum for questions to answer. A free spirit will be suggested to check the content area for something to do. The student can then decide on its own on what to do next. A player is extrinsically motivated by getting rewards. Therefore, the AI-tutor suggests the student to do an activity where he can earn rewards. In MEITREX, this could be achieved by completing a quizz, a code exercise or finishing lecture materials. Similarly to 6.4.3, disruptors are not handled specially. If the main player type is a disruptor, the student will be handled according to the second highest player type.

6.5 Examples

Some example prompts and corresponding LLM responses are shown in appendix D. To understand how the prompts are engineered, you are invited to read the implementation chapter 7.

7 Implementation of the AI-tutor additions

MEITREX uses microservice architecture. This means that different services exist all doing things modularly. In case one service fails critically, the other services are still running. There exist the following 15 microservices: `assignment_service`, `content_service`, `course_service`, `docprocai_service`, `flashcard_service`, `frontend`, `gamification_service`, `GraphQL_gateway`, `media_service`, `notification_service`, `quiz_service`, `reward_service`, `skilllevel_service`, `tutor_service` and `user_service`. With the exception of the DocProcAI and the frontend each microservice runs in Java while the DocProcAI runs in Python and the frontend runs in TypeScript and HTML.

There was already a prototype for an AI-tutor inside of the tutor service. Our implementation is built on top of the existing implementation and can be found on github <https://github.com/MEITREX> in the corresponding repository for each microservice. Most changes were made in the tutor service https://github.com/MEITREX/tutor_service, but also some changes were necessary in other microservices to support the new functionality of the AI-tutor. This chapter covers the implementation of the designed AI-tutor and changes made to the relevant microservices to support the new functionality.

7.1 Gamification usage and prompt building

To implement the individualized feedback as designed in chapter 6, the gamification elements HEXAD Player Type and SkillLevel of the student had to be integrated into the prompt building process.

7.1.1 Question classification

Before the AI-tutor can generate an appropriate response, it must first determine the type of question being asked. This classification step ensures that each question is routed to the correct handling logic.

The classification is performed by sending the student's question to the large language model along with a specialized prompt template shown in Listing 7.1. This template instructs the model to categorize the question into one of five categories: `SYSTEM`, `LECTURE`, `CODE_FEEDBACK`, `OTHER`, or `UNRECOGNIZABLE`. Based on the returned category, the tutor service then invokes the appropriate service method to handle the specific type of question.

The previous iteration of the AI-tutor already included this question classification step. However, the prompt template was slightly modified to also categorize code feedback and follow-up questions by giving the model the option to choose the `CODE_FEEDBACK` or `OTHER` category respectively.

7 Implementation of the AI-tutor additions

Your job is to categorize a given question into one of five categories: SYSTEM, LECTURE, CODE_FEEDBACK, OTHER, or UNRECOGNIZABLE.

To do this you must also correct typos and grammar in the question where possible. If the corrected version makes the meaning clear, use it to choose the appropriate category.

Categories:

1. SYSTEM: Choose this if the question is about the website or app the user is currently using. For example: "Where do I change my avatar?" These questions relate specifically to the system or platform itself.
2. LECTURE: Choose this if the question is about understanding or explaining academic content related to a university course. These questions should NOT relate to the website or app. These questions are not limited to only a "question" itself, but can also be messages that convey the meaning of trying to understand a topic on a deeper level. For example: "What is unsupervised learning" or "Can you give examples for the Chain rule in differentiation"
3. CODE_FEEDBACK: Choose this if the student is asking for feedback, help, or explanations about code they have written for an assignment. This includes questions about errors, improvements, debugging, or understanding their own code implementation. For example: "Can you help me with my code?" or "Why doesn't my function work?" or "How can I improve my implementation?"
4. OTHER: Choose this if the question if it is neither a SYSTEM, nor a LECTURE nor a CODE_FEEDBACK question and if the question might be a follow up question. For example: "Can you explain it again?" or "Can you explain it differently?" or "Can you provide an example to clarify your previous answer?"
5. UNRECOGNIZABLE: Choose this if the question is too garbled, incomplete, or contains too many typos or grammar issues to be understood reliably.

Only use the UNRECOGNIZABLE category when the meaning is truly unclear even after trying to correct it.

Your response must be a JSON object ONLY (no extra text) with the exact following structure and keys:

```
{
  "question": "the original question here",
  "category": "SYSTEM" | "LECTURE" | "CODE_FEEDBACK" | "OTHER" |
             "UNRECOGNIZABLE"
}
```

QUESTION:
{{question}}

Listing 7.1: Prompt template for question classification

7.1.2 Skill Level Integration

The skill level of a student is used to adjust the complexity and depth of the feedback provided by the AI-tutor. Algorithm 7.2 shows how the skill level is translated into guidance text for the achiever player type that is integrated into the prompt sent to the large language model. The method distinguishes between three competence levels: low, medium, and high, each receiving tailored instruction guidance.

```

private String getSkillLevelGuidanceAchiever(double skillLevel) {
    if (skillLevel <= skillLevelLowThreshold) {
        return "Focus on providing clear, simple hints that guide " +
            "the student toward the next step without " +
            "overwhelming them.";
    } else if (skillLevel < skillLevelHighThreshold) {
        return "Offer balanced hints that assist the student's " +
            "reasoning while still allowing them to work out " +
            "the solution independently. Assume general " +
            "familiarity with the topic.";
    } else {
        return "Assume advanced understanding.";
    }
}

```

Listing 7.2: Skill level guidance generation for achiever player type

7.1.3 Player Type Adaptation

The HEXAD player type is used to customize the tone and style of hints provided by the AI-tutor. Algorithm 7.3 demonstrates how different player types receive different hint strategies. For instance, achievers receive hints that support mastery goals, while philanthropists and socializers are encouraged to collaborate and share knowledge.

```

return switch (playerType) {
    case ACHIEVER ->
        "Give the user a small hint that helps them solve the " +
        "problem without revealing the full solution. Keep the " +
        "hint short and motivating. This is for an Achiever who " +
        "values mastery and goals.";
    case PHILANTHROPIST, SOCIALISER ->
        "Encourage the user to collaborate and share knowledge. " +
        "If relevant, suggest looking for similar questions in " +
        "the forum or encouraging them to help others. This is " +
        "for a Philanthropist or Socialiser who values social " +
        "interaction.";
    case FREE_SPIRIT ->
        "Give the user a normal hint that helps them understand " +
        "the problem and explore solutions independently. " +
        "Encourage exploration and creativity. This is for a " +
        "Free Spirit who values autonomy.";
    case PLAYER, DISRUPTOR ->
        "Be especially careful not to reveal any solution " +
        "directly. Frame the hint as a challenge. This is for " +
        "a Player or Disruptor who values competition.";
};

```

Listing 7.3: Player type-specific hint customization

7.1.4 Prompt Construction

The final prompt sent to the large language model is constructed by combining multiple context elements. Listing 7.4 shows the construction of template arguments that are used to fill a prompt template on the example of a follow-up question. These arguments include the student's question, relevant lecture materials, skill level guidance, conversation history and code context. The previous iteration of the AI-tutor did not include the skill level guidance, conversation history and code context arguments, which were added as part of this thesis to enhance the AI-tutor's capabilities.

```
private LectureQuestionResponse handleFollowUpQuestion(String question, UUID courseId,
LoggedInUser currentUser) {
    ...

    List<TemplateArgs> promptArgs = List.of(
    TemplateArgs.builder()
        .argumentName("question")
        .argumentValue(question)
        .build(),
    TemplateArgs.builder()
        .argumentName("content")
        .argumentValue(contentString)
        .build(),
    TemplateArgs.builder()
        .argumentName("skill")
        .argumentValue(skillLevelPromptContent)
        .build(),
    TemplateArgs.builder()
        .argumentName("conversationHistory")
        .argumentValue(conversationHistory)
        .build(),
    TemplateArgs.builder()
        .argumentName("codeContext")
        .argumentValue(codeContext)
        .build()
    );

    TutorAnswer response = ollamaService.startQuery(
        TutorAnswer.class, prompt, promptArgs, new TutorAnswer(ERROR_MESSAGE));

    conversationHistoryService.addConversationExchange(
        currentUser.getId(), courseId, question, response.getAnswer());

    ...

    return new LectureQuestionResponse(response.getAnswer(), sources);
}
```

Listing 7.4: Template arguments for prompt construction

The final prompt is generated by filling a predefined prompt template with these arguments. An example prompt template for a follow-up question is shown in Listing 7.5. All other prompt templates for different question types follow a similar structure, but differ in specific instructions and context provided to the LLM. They can be found in the appendix C.

While some prompt templates already existed in the earlier implementation of the AI-tutor, all templates of this thesis were based on top of them. The prompt templates shown in this thesis were extended to include the new *skill*, *conversationHistory* and *codeContext* arguments depending on the type of prompt template to enhance the AI-tutor's capabilities, while also changing some instructions.

Once the prompt is constructed, it is sent to the large language model for processing and response generation. The response is then saved in the conversation history for future context before the final answer is returned to the frontend.

In the earlier implementation of the AI-tutor, the AI-tutors workflow for prompt construction was similar, but without the additional context elements introduced in this thesis. Previously, only the prompt templates for question classification, lecture questions and hint generation existed and were changed during this thesis to include the new context elements.

7.2 Implementation of the AI-tutor logic

7.2.1 Already existing Tutor implementation

The workflow of the existing AI-tutor implementation follows a standard request-response pattern. The process is initiated when a user submits a request to the AI-tutor via the user interface or by requesting a hint during an assignment. Consequently, the frontend relays this request to the tutor service. The tutor service processes the incoming data, executes the necessary business logic, and returns a response to the frontend, where the response is displayed to the user.

7.2.2 Saving Gamification Elements and Chat History

Data persistence for gamification elements, conversation logs, and other related entities is managed by extending the *JpaRepository* interface. This approach ensures consistency with the architectural standards, as this persistence pattern is already used across other microservices within MEITREX.

7.2.3 Reactive AI-tutor

The core logic driving the reactive part of the AI-tutor is designed to be straightforward and modular. The control flow branches into distinct cases based on the categorization of the student's question. To ensure maintainability and code readability, each usecase is handled by a dedicated method or service. The responsibilities are distributed among several specialized services:

- The *TutorController* acts as the entry point for incoming GraphQL requests from the frontend and manages the outgoing responses. Additionally, it listens for events propagated by other microservices and calls the corresponding methods to handle these events

7 Implementation of the AI-tutor additions

```
You are an AI tutor helping a student understand lecture material.

Below is a question the student has asked, along with excerpts from
the lecture that are relevant to the question. This might be a followup
question to an earlier exchange. If this is not the case, answer that
you can not help the student with that currently.

If is it a question about the lecture, please use only the information
in the provided lecture content to answer the question as accurately as
possible. If the information is insufficient to answer, say so and avoid
making assumptions.

If it is a question about code, then analyze the student's code and
respond to their question. Be constructive, encouraging, and educational.
If you identify errors or issues in the code, point them out clearly.
If the code looks correct, acknowledge that and provide suggestions for
improvement or best practices. Keep your response focused, clear, and
actionable. It is very important that you do not send code snippets
yourself. Only tell what is wrong or can be improved but never send the
corrected code.

Always answer in the same language as the student's question.
Do not translate or localize technical terms or vocabulary from the
lecture content -- keep them exactly as they appear.

Your entire response must be a valid JSON object of the following form:

{"answer": "your answer here"}

You know how well the student usually performs in this course.
{{skill}}

If it helps clarity, you may refer to specific parts of the lecture
content using their assigned numbers (e.g., "[1]", "[2]", etc.).
---

Question:
{{question}}

---

Relevant Lecture Content:
{{content}}

---

You might also have communication history with the student.
Do not use the information provided by the student as facts, he can
make mistakes. Base your answer on the lecture content.
Conversation History:
{{conversationHistory}}

---

STUDENT'S CODE SUBMISSION:
{{codeContext}}
```

Listing 7.5: Example prompt template for follow-up questions

- The TutorService handles reactive feedback logic
- The StudentCodeSubmissionService is responsible for persisting code submissions and retrieving them when code-specific feedback is required
- The UserPlayerTypeService and UserSkillLevelService manage the storage and retrieval of user-specific context, specifically player types and skill levels, respectively
- The ConversationHistoryService ensures that the already existing communication context is preserved by saving and loading the chat history. In the current implementation, we decided to store the last 3 message exchanges between the student and the AI-tutor to provide sufficient context for generating relevant responses without overwhelming the system with excessive data. Additionally, we filter out any messages that are older than 30 minutes to maintain the relevance of the conversation history

7.3 Proactive Feedback

Unlike the reactive feedback which respond to direct user queries, proactive features are managed centrally by the ProactiveFeedbackService. This class handles all proactive feedback logic, including the generation and storage of feedback messages based on specific triggers.

7.3.1 Implementing into frontend

To integrate proactive feedback into the frontend without disrupting the user experience, a polling mechanism is employed. The client polls the backend every 15 seconds to check for the existence of new feedback. We decided to use polling since it is already implemented in other parts of MEITREX and it is a simple way to achieve the desired functionality without adding too much complexity to the system. The timer was set to 15 seconds to balance responsiveness with system load, ensuring that students receive timely feedback without overwhelming the backend with frequent requests.

During this process, a request is sent to the tutor service, which verifies if any new proactive feedback has been generated and stored for the student. To ensure relevance, proactive feedback is only delivered if it is not older than 30 minutes and has not been delivered before. This prevents the display of outdated feedback in scenarios where a user might log off immediately after triggering a feedback event and return at a later time.

7.4 Communication between different microservices

The AI-tutor as designed in chapter 6 uses different student specific personal information such as HEXAD PlayerType or SkillLevel of the student. However, this information was not available in the tutor service but saved in other microservices. Therefore, communication between different microservices had to be established to retrieve this information from other microservices.

There are two main ways of communicating between microservices currently used in the MEITREX architecture: GraphQL queries and events. I will go briefly over each to explain the core advantages and disadvantages.

7.4.1 GraphQL

Each microservice offers some possible queries it might receive and how it answers it. The main advantage is that communication is possible through standard HTTP Post requests containing JSON payloads. This means that different services can communicate with each other even if they use different programming languages. In the current implementation of MEITREX, most services use the programming language Java, however the frontend does not use Java. This is a good way to communicate between the frontend and other services.

7.4.2 Events

Another way to communicate between services is with events. Every microservice in MEITREX uses the same library, the MEITREX-common library. This library uses a TopicPublisher to share events between different microservices. A service can subscribe to different event types defined in this library as value of an enum. Another service can now publish an event with this TopicPublisher, so every service subscribing to this event type gets triggered with the event sent to it. This way of communication is very efficient for sending data to multiple services at once.

7.4.3 Implementation inside of MEITREX

In the beginning, we tried to implement communication between services with GraphQL. However, because of some dependency issues inside of MEITREX, it was not possible to use GraphQL in the tutor service for external communication with the `gamification_service` or other microservices without rewriting big parts of the system. Therefore, we decided to use events for communication between the `tutor_service` and other services which aligns with new goals of moving communication in MEITREX to be almost exclusively event-driven.

Figure 7.1 shows the relevant communication architecture for changes done to the AI-tutor. The heart of the AI-tutor is the tutor service which handles the AI-tutor logic. The AI-tutor needs information about the player type or skill level of a student to give individual feedback. In the tutor service, this information is saved in its own database. Database duplication is common practice in microservice architecture to ensure loose coupling between services. Whenever the gamification service sets or updates the player type of a student, it publishes an event with this information which the tutor service subscribes to. Similarly, whenever the skill level of a student is updated, the skill level service publishes an event with this information which the tutor service subscribes to. The assignment service publishes an event whenever a student submits code for an assignment, which the tutor service also subscribes to in order to know about the latest code submission of a student.

In the assignment service, whenever a student finishes an assignment like a quiz or a coding exercise, the assignment service publishes an event with information about the student and the completed assignment. The tutor service uses this information to trigger proactive feedback. Additionally, the skilllevel service listens to this event to update the skill level of the student based on the completed assignment, which was already implemented before this thesis.

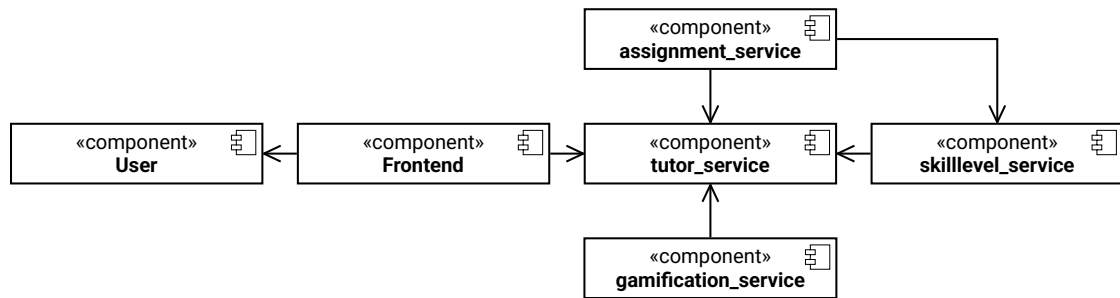


Figure 7.1: Component diagram showing the communication architecture between relevant microservices. The tutor service subscribes to events from the gamification service, skill level service, and assignment service to receive student-specific data.

Communication with the frontend remains almost unchanged. The frontend sends request via GraphQL to the tutor service and receives responses back via GraphQL. The addition of proactive feedback required some changes in the frontend to poll for new proactive feedback every few seconds.

7.5 Thresholds and constraints for evaluation

To be able to evaluate the AI-tutor, the previously undefined thresholds for the various scenarios from chapter 6 had to be defined concretely. The different skill related thresholds were defined as low competence for 0 to 0.3, medium competence for 0.3 to 0.7 and high competence for 0.7 to 1.0. Additionally, to evaluate the whole extend of the AI-tutor without taking too much time, proactive code feedback triggered every time to ensure the student not skipping the evaluation for code feedback.

All thresholds are saved in the `application.properties` file of the tutor service to be easily adjustable in the future if needed.

8 Evaluation

To evaluate the effectiveness and user acceptance of the AI-tutor integrated into the MEITREX learning platform, a user study was conducted with 17 participants and 33 content questions. The study aimed to assess how well the AI-tutor supports learners in coding exercises, focusing on aspects such as clarity of feedback, relevance of hints, and overall user satisfaction.

This chapter covers the design, results and discussion of the evaluation.

8.1 Study Design

The evaluation was designed as a structured session divided into four distinct content parts. Each part was targeted at assessing a specific functional area of the AI-tutor, while the first part collected demographic information about the participants. This modular structure allowed for a focused analysis of each feature of the AI-tutor, ensuring that feedback was relevant and specific to the functionalities being tested. Full questions are listed in Appendix B while raw data are made available as part of our research dataset [DM26a].

8.1.1 Demo course in MEITREX and general procedure

The evaluation took place in a specifically created demo course within the MEITREX platform, which included a coding exercise, two quizzes and lecture materials. While the coding exercise was unrelated to the course content, the quizzes were based on the lecture materials provided in the course. These lecture materials consisted of 5 lectures of the course (*Advanced Software Engineering* by Professor Becker taught at the University of Stuttgart and covered different testing mechanisms in software engineering. The coding exercise required students to implement basic functions in Java, such as calculating the factorial of a number. The focus of the coding exercise was not on the complexity of the task, but rather on evaluating the AI-tutor's ability to provide meaningful feedback and assistance during the coding process without overwhelming a potentially inexperienced programmer with overly complex requirements.

Before starting the evaluation, the evaluation was piloted with two first-semester students to ensure clarity and appropriateness of the tasks and questions after refining the questions with the supervisor. The actual evaluation was then conducted asynchronously, allowing participants to complete the tasks at their own pace at a time of their choosing. Participants were each given one personalised account to access the MEITREX platform and the demo course.

Every participant was able to skip any part of the evaluation they did not wish to complete. Corresponding questions were only shown if the participant indicated that they had completed the respective section. The evaluation consisted of a total of 42 questions, with 34 content questions, including 29 questions using a 5 point Likert scale and a *no answer* option and 4 open ended questions.

8.1.2 Demographic Questions

The first part of the study collected demographic information from the participants, including their course of study, semester, age, gender identity, frequency of AI usage, and MEITREX username. The MEITREX username was important to link survey responses with player type data. This information was crucial for contextualizing the feedback received and understanding the diversity of the participant pool.

8.1.3 Code Feedback Section

This section evaluated the AI-tutor's ability to provide clear and helpful coding feedback. The student was asked to complete a coding exercise inside of the Demo MEITREX course and then ask the AI-tutor for feedback on their code or to ask for help when they had difficulties. The coding exercise was designed to be straightforward, allowing participants to focus on the interaction with the AI-tutor rather than the complexity of the coding task itself. Students had to implement four basic arithmetic functions with the hardest being the calculation of the factorial of a number and were free to ask the AI-tutor for feedback or assistance at any point during the exercise.

This section aims to assess how effectively the AI-tutor can identify and communicate issues in the code to improve the students code. Additionally, it evaluates the usefulness of the feedback provided and whether the AI-tutor gave too much information.

8.1.4 Hint Section

This section assessed the AI-tutors capability to change the complexity of its explanations based on the user's skill level. The student was instructed to first complete the first quiz of the demo course, which covered basic concepts from the provided lecture materials. After completing the first quiz, the AI-tutor should have saved some information about the student's skill level. Therefore, when the student asked for hints during the second quiz, the AI-tutor should have adapted its explanations to fit the student's personal skill level.

8.1.5 Proactive Feedback Section

One of the key features of the AI-tutor is its ability to provide proactive feedback, which aims to assist students without waiting for them to ask for help. At this part of the evaluation, the student should have already received proactive feedback during the coding exercise and quizzes. However, to ensure that the student experienced this feature, they were instructed to take on a quiz to ensure that they would receive proactive feedback from the AI-tutor.

This part of the evaluation focused on assessing the relevance, timing, and overall impact of the proactive feedback provided by the AI-tutor. Additionally, the proactive feedback was evaluated how it influenced the motivation and learning experience of the students. However, the students were also able to answer if the AI-tutor's proactive feedback interrupted their workflow negatively.

8.1.6 General AI-tutor Evaluation

Once the student finished all previous sections, they were asked to complete a comprehensive evaluation of the AI-tutor as a whole. The primary objective was to assess the tool's usability and the quality of its interactions using a standard Likert scale.

Given the importance of accuracy in educational tools, trust and source transparency were central components of this evaluation section. Students were asked to evaluate whether the inclusion of references to original course materials increased their confidence in the AI's answers. Simultaneously, the survey measured whether the volume of these sources was helpful or if it contributed to cognitive overload.

Furthermore, the evaluation explored the pedagogical impact of the tool. It aimed to measure whether the AI-tutor succeeded in motivating students to engage more deeply with the course content and if it encouraged them to ask further questions. Finally, to gauge the potential for future adoption, students indicated whether they would use a similar AI-tutor in future Software Engineering courses or recommend it to their peers.

To complement these structured responses, the evaluation concluded with a section ended with open-ended questions. While the Likert scale provided quantitative data, this section allowed students to express feedback unlinked to predefined answer options, to not constrain their responses. Participants were able to highlight specific features they found helpful, detail bothersome elements and suggest on where improvements could be made. This approach ensures that the evaluation captures both the general sentiment and a more specific user experience with the AI-tutor.

8.1.7 Students reached

The evaluation was open for approximately four weeks, during which the participants could complete the evaluation. We reached out to students studying software engineering related courses at the University of Stuttgart. Similarly to the poll described in Chapter 4, I directly asked students in V38, the computer science building at the University of Stuttgart, to participate in the evaluation. Students had no incentive to participate in the evaluation of this thesis and did so voluntarily.

The evaluation was completed by 17 students. The average completion time was approximately 15 hours and 45 minutes. It was heavily skewed by a few participants who left the evaluation open for multiple days. The median completion time was approximately 45 minutes, which is only 5 minutes less than the piloted part, indicating that most participants completed the evaluation in a reasonable timeframe.

8.2 Results

This chapter presents the results of the user study. The data was collected from 17 participants who completed the evaluation. The results are categorized into demographic information, quantitative feedback on specific AI-tutor features (Code Feedback, Hints, Proactive Feedback), general user acceptance, and qualitative feedback.

We assume for the Likert scale questions that the distance between each point is equal and therefore calculate means and standard deviations. Therefore, the Likert scale ranges from 1 (Strongly Disagree) to 5 (Strongly Agree). The following abbreviations are used throughout this chapter: median (M), standard deviation (SD_v) and confidence interval, here with a confidence level of 95% (CI) calculated with *t*-distribution.

8.2.1 Students demographic

This section describes the demographics of the participants who took part in the evaluation of the AI-tutor integrated into the MEITREX learning platform.

In total, 17 complete responses were collected from participants who completed the evaluation. All participants were students enrolled at the University of Stuttgart in software engineering related study programs. The distribution of participants across different courses of study is shown in Figure 8.2. The participants were predominantly male (14 males, 2 females, 1 preferred not to say / other). The majority of participants were in their bachelors, with 13 students, while 4 were pursuing their master degree, as detailed in Table 8.3.

The majority of participants indicated that they use AI-based tools regularly, with only one participant reporting rare usage and none reporting never using such tools. This suggests a generally high level of familiarity with AI systems among the surveyed students. Exact usage frequencies are shown in Figure 8.1. Average age of the participants was 21.5 years (SD_v=2.6), with ages ranging from 18 to 26 years.

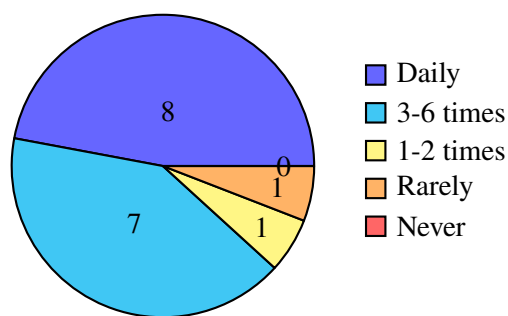


Figure 8.1: Frequency of AI tool usage among the evaluated students (n=17)

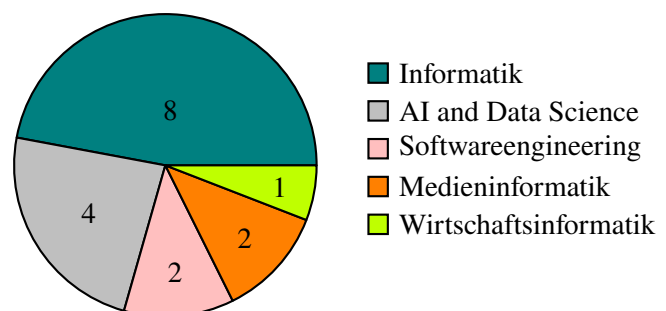


Figure 8.2: Distribution of participants by course of study (n=17)

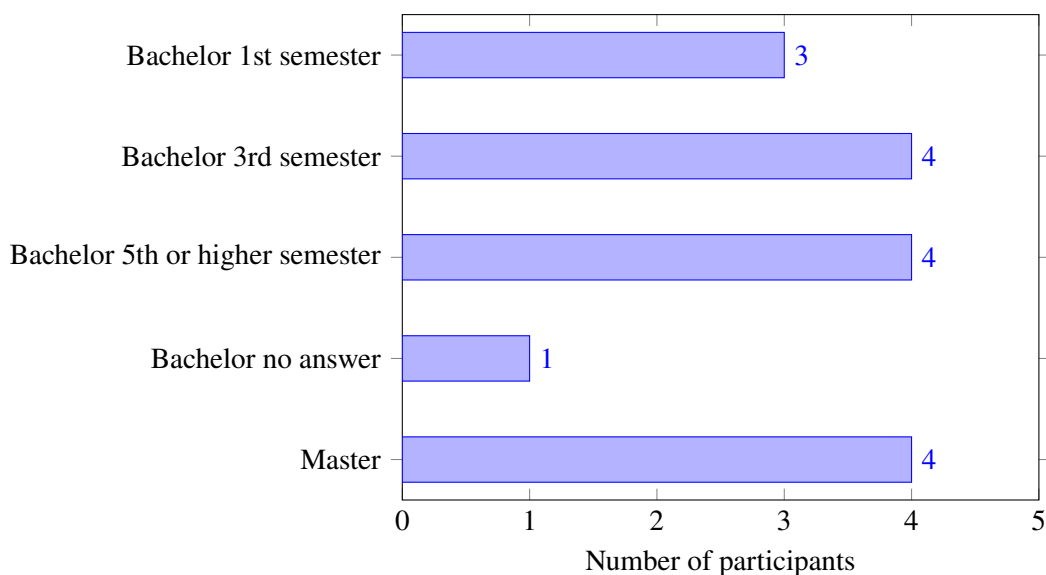


Figure 8.3: Distribution of participants by semester (n=17)

8.2.2 Code Feedback

Participants evaluated the clarity and helpfulness of the code feedback. However, only 12 of the 17 participants completed this section, as some chose to skip the coding exercise. Reasons for skipping are not explicitly collected, but three participants mentioned in a personal conversation afterwards, that they did not feel comfortable with providing the requested root-access to their GitHub account, which is necessary to complete coding assignments in MEITREX.

The feedback is generally perceived as clear and understandable ($M = 3.75$, $SD_v = 1.06$, $CI [3.08, 4.42]$), with the majority of participants agreeing or strongly agreeing with this statement. These results are visualized in Figure 8.4. Students did not perceive the feedback as overwhelming, with most disagreeing with the statement that the AI-tutor provided too much information ($M = 1.83$, $SD_v = 0.58$, $CI [1.47, 2.20]$), as shown in Figure 8.6.

However, when evaluating the helpfulness of the AI-tutor's suggestions in improving code quality, responses were mixed. As shown in Table 8.1, equal numbers of participants expressed agreement and neutrality, indicating that while some found the suggestions beneficial, others were less convinced of their utility. Students were less convinced that the AI-tutor helped them identify errors in their code, with a mean score of 2.50 ($SD_v = 1.27$, $CI [1.69, 3.31]$).

Overall, students indicated mixed willingness to use the AI-tutor again for coding exercises, with an even split between those agreeing and disagreeing ($M = 2.83$, $SD_v = 1.11$, $CI [2.13, 3.54]$), as shown in Figure 8.5.

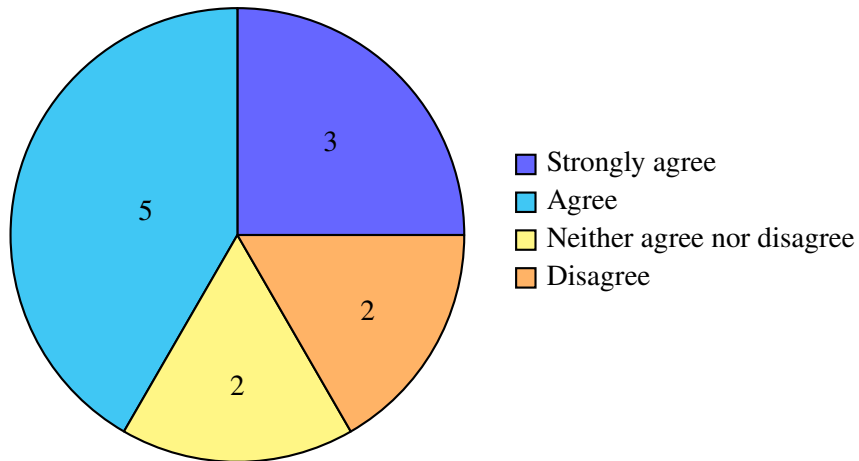


Figure 8.4: The AI-tutor’s coding feedback was clear and understandable (n=12)

Table 8.1: Evaluation of AI-tutor’s helpfulness in code improvement (n=12). Scale: SA=Strongly Agree (5), A=Agree (4), N=Neither (3), D=Disagree (2), SD=Strongly Disagree (1), NA=No Answer

Statement	SA	A	N	D	SD	NA	M	SDv	CI
Suggestions helped improve code	1	4	4	2	1	–	3.17	1.11	[2.46, 3.87]
Helped identify errors	1	1	2	4	2	2	2.50	1.27	[1.59, 3.41]

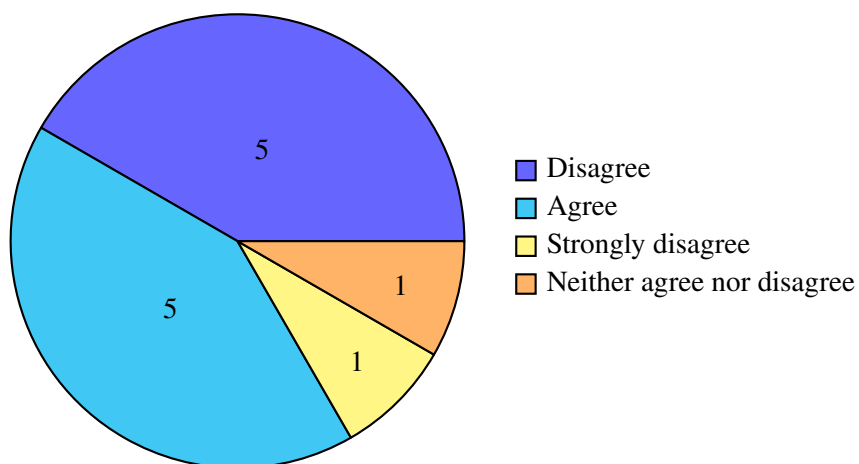


Figure 8.5: I will use the AI-tutor again if I have problems with coding exercises (n=12)

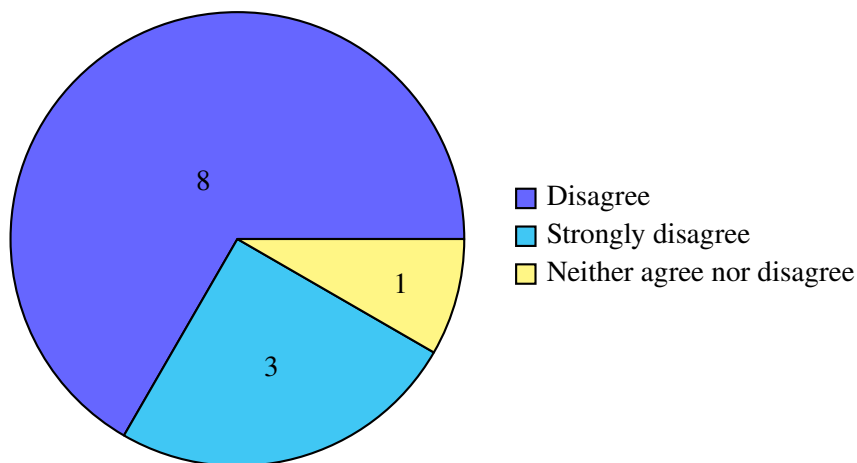


Figure 8.6: The AI-tutor gave me too much information (n=12)

8.2.3 Hints and Explanations

All 17 participants evaluated the AI-tutor's ability to tailor hints and explanations to their skill level. Students were generally neutral to slightly positive about the AI-tutor's adaptability to their skill level ($M = 3.12$, $SDv = 0.70$, $CI [2.76, 3.48]$), while most participants (9) neither agreed nor disagreed with the statement that the hints were fitting to their personal skill, as shown in Figure 8.7.

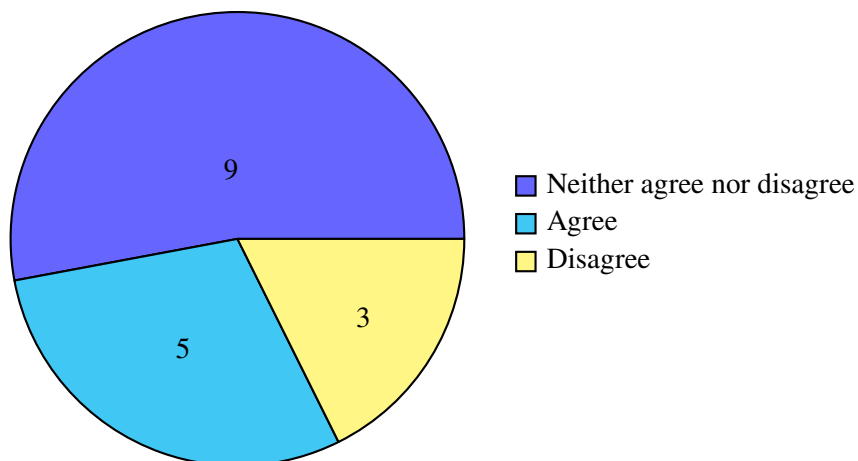


Figure 8.7: The hints provided by the AI-tutor were fitting to my personal skill (n=17)

Participants evaluated whether the AI-tutor avoided explanations that were too simple or too advanced for their level. Results are shown in Table 8.2. While results are perceived as not too simple ($M = 3.38$, $SDv = 1.02$, $CI [2.82, 3.92]$) or too advanced ($M = 3.24$, $SDv = 1.15$, $CI [2.65, 3.83]$), there is a noticeable spread in responses, indicating varied experiences among participants regarding the complexity of explanations provided by the AI-tutor.

Table 8.2: Cross-tabulation of explanation complexity evaluation (n=17)

Avoided too simple (rows) / too advanced (columns)	Strongly agree	Agree	Neither agree nor disagree	Disagree	Strongly disagree	No answer	Total
Strongly agree				1			1
Agree	2	5	1	1			9
Neither agree nor disagree			1				1
Disagree		1	1	2	1		5
Strongly disagree							0
No answer			1				1
Total	2	6	4	4	1	0	17

8.2.4 Proactive Feedback

In total, 15 out of the 17 participants noted that they received proactive feedback from the AI-tutor during the evaluation. However, both missing participants should have received proactive feedback, because both finished both other sections which should have triggered proactive feedback during the evaluation. It is possible that these participants did not notice the proactive feedback or did not want to answer this part of the evaluation.

The evaluation of proactive feedback features is summarized in Table 8.3 and visualized in Figure 8.8. The evaluating students rated the proactive feedback as intentional and meaningful ($M = 3.80$, $SD_v = 0.94$, $CI [3.28, 4.32]$) and not interrupting their workflow ($M = 3.87$, $SD_v = 1.13$, $CI [3.24, 4.49]$). Additionally, students found the proactive feedback motivating ($M = 4.00$, $SD_v = 1.00$, $CI [3.45, 4.55]$).

However, the relevance of the proactive feedback was rated rather low ($M = 2.53$, $SD_v = 1.06$, $CI [1.95, 3.12]$), with most participants disagreeing that the feedback appeared when they needed support ($M = 2.87$, $SD_v = 0.92$, $CI [2.36, 3.37]$). Even though the proactive feedback was motivating, students were neutral to negative about whether it improved their learning experience ($M = 2.93$, $SD_v = 0.96$, $CI [2.40, 3.47]$).

Students were split on whether they would like to receive proactive feedback again ($M = 3.33$, $SD_v = 1.35$, $CI [2.59, 4.08]$). This is especially visible in Figure 8.8, where it is notable that no students neither agreed nor disagreed with this statement, indicating a polarized opinion on the feature. Finally, students generally did not find the proactive feedback more helpful than reactive feedback ($M = 2.57$, $SD_v = 0.76$, $CI [2.14, 3.01]$).

8.2.5 General User Experience

The overall evaluation of the AI-tutor was positive. Results can be found in Table 8.4, while the diverging stacked bar chart in Figure 8.9 provides a visual representation of the distribution of responses. Participants found the tool very easy to use ($M = 4.24$, $SD_v = 1.03$, $CI [3.71, 4.77]$), which is backed up by the low disagreement with the reverse-scored question about ease of use ($M = 2.12$, $SD_v = 1.27$, $CI [1.47, 2.77]$). The trust in the answers of the AI-tutor is generally high ($M = 4.00$, $SD_v = 1.10$, $CI [3.43, 4.57]$), with only 3 participants disagreeing or strongly disagreeing with this statement. Additionally, the references to original course materials were perceived as

Table 8.3: Evaluation of proactive feedback features (n=15). Scale: SA=Strongly Agree (5), A=Agree (4), N=Neither (3), D=Disagree (2), SD=Strongly Disagree (1), NA=No Answer

Statement	SA	A	N	D	SD	NA	M	SDv	CI
Felt intentional and had meaning	3	8	2	2	0	–	3.80	0.94	[3.28, 4.32]
Did not interrupt workflow	4	8	1	1	1	–	3.87	1.13	[3.24, 4.49]
Appeared when needed support	0	5	3	7	0	–	2.87	0.92	[2.36, 3.37]
Was relevant	1	1	5	6	2	–	2.53	1.06	[1.95, 3.12]
Was motivating	5	7	1	2	0	–	4.00	1.00	[3.45, 4.55]
Improved learning experience	0	5	5	4	1	–	2.93	0.96	[2.40, 3.47]
Would like to receive again	3	6	0	5	1	–	3.33	1.35	[2.59, 4.08]
More helpful than reactive	0	1	7	5	1	1	2.57	0.76	[2.14, 3.01]

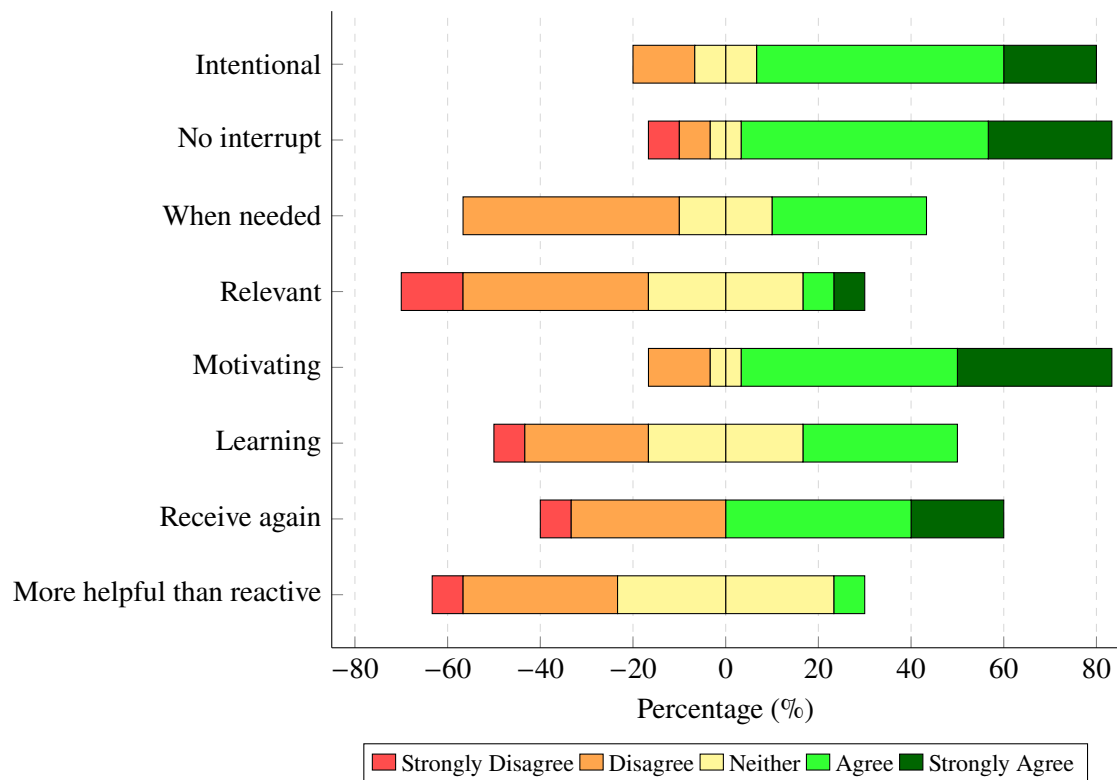


Figure 8.8: Diverging stacked bar chart of proactive feedback ratings (n=15). Negative responses on the left, neutral in center, positive on the right.

not overwhelming ($M = 3.94$, $SDv = 1.03$, $CI [3.41, 4.47]$) and increased trust ($M = 3.75$, $SDv = 1.00$, $CI [3.22, 4.28]$, $n=16$). Students reported that the AI-tutor helped them understand course content ($M = 3.88$, $SDv = 0.70$, $CI [3.52, 4.24]$) and provided a fitting amount of information ($M = 3.75$, $SDv = 1.06$, $CI [3.19, 4.31]$, $n=16$). Most participants agreed that the AI-tutor fit their personality ($M = 3.57$, $SDv = 0.65$, $CI [3.20, 3.94]$, $n=14$), with no participant disagreeing with this statement but 3 participants choosing not to answer this question. However, students were neutral about whether they would use the AI-tutor again in Software Engineering courses ($M = 3.18$, $SDv = 1.29$, $CI [2.52, 3.84]$) or recommend it to others ($M = 3.35$, $SDv = 0.86$, $CI [2.91, 3.79]$).

Table 8.4: General user experience evaluation ($n=17$). Scale: SA=Strongly Agree (5), A=Agree (4), N=Neither (3), D=Disagree (2), SD=Strongly Disagree (1), NA=No Answer

Statement	SA	A	N	D	SD	NA	M	SDv	CI
Was easy to use	9	5	1	2	0	–	4.24	1.03	[3.70, 4.77]
Answers were relevant	3	6	2	4	2	–	3.24	1.35	[2.54, 3.93]
Helped understand content	3	9	5	0	0	–	3.88	0.70	[3.52, 4.24]
Trusted the answers	7	4	3	2	1	–	3.82	1.29	[3.16, 4.49]
Not overwhelmed by sources	6	7	1	3	0	–	3.94	1.09	[3.38, 4.50]
References increased trust	4	6	4	2	0	1	3.75	1.00	[3.22, 4.28]
Motivated to ask more	1	7	5	4	0	–	3.29	0.92	[2.82, 3.77]
Increased engagement	1	10	2	4	0	–	3.47	0.94	[2.99, 3.96]
Was not easy to use*	1	2	2	5	7	–	2.12	1.27	[1.47, 2.77]
Information amount fitting	5	4	5	2	0	1	3.75	1.06	[3.18, 4.32]
Fitted my personality	1	6	7	0	0	3	3.57	0.65	[3.20, 3.94]
Would use again in SE	3	5	2	6	1	–	3.18	1.29	[2.52, 3.84]
Would recommend to others	0	10	3	4	0	–	3.35	0.86	[2.91, 3.80]

*Reverse-scored question: Lower score indicates easier to use

8.2.6 Latency and other issues

Even though there was no specific question about the response time of the AI-tutor, multiple students mentioned it in the open-ended questions. These students complained about long response times from the AI-tutor. In total, 5 students wrote about too long response times from the AI-tutor. One student wrote: *"generating an answer took forever"*.

Additionally, some students reported error messages occurring when the AI-tutor could not handle a request or that the AI-tutor was not aware of the specific page context the student is currently on.

8.3 Discussion

In general, the evaluation results indicate a positive perception of the AI-tutor among the participants. Students found the AI-tutor easy to use and generally trusted its answers, especially when references to original course materials were provided. However, the implementation of the AI-tutor still has room for improvement.

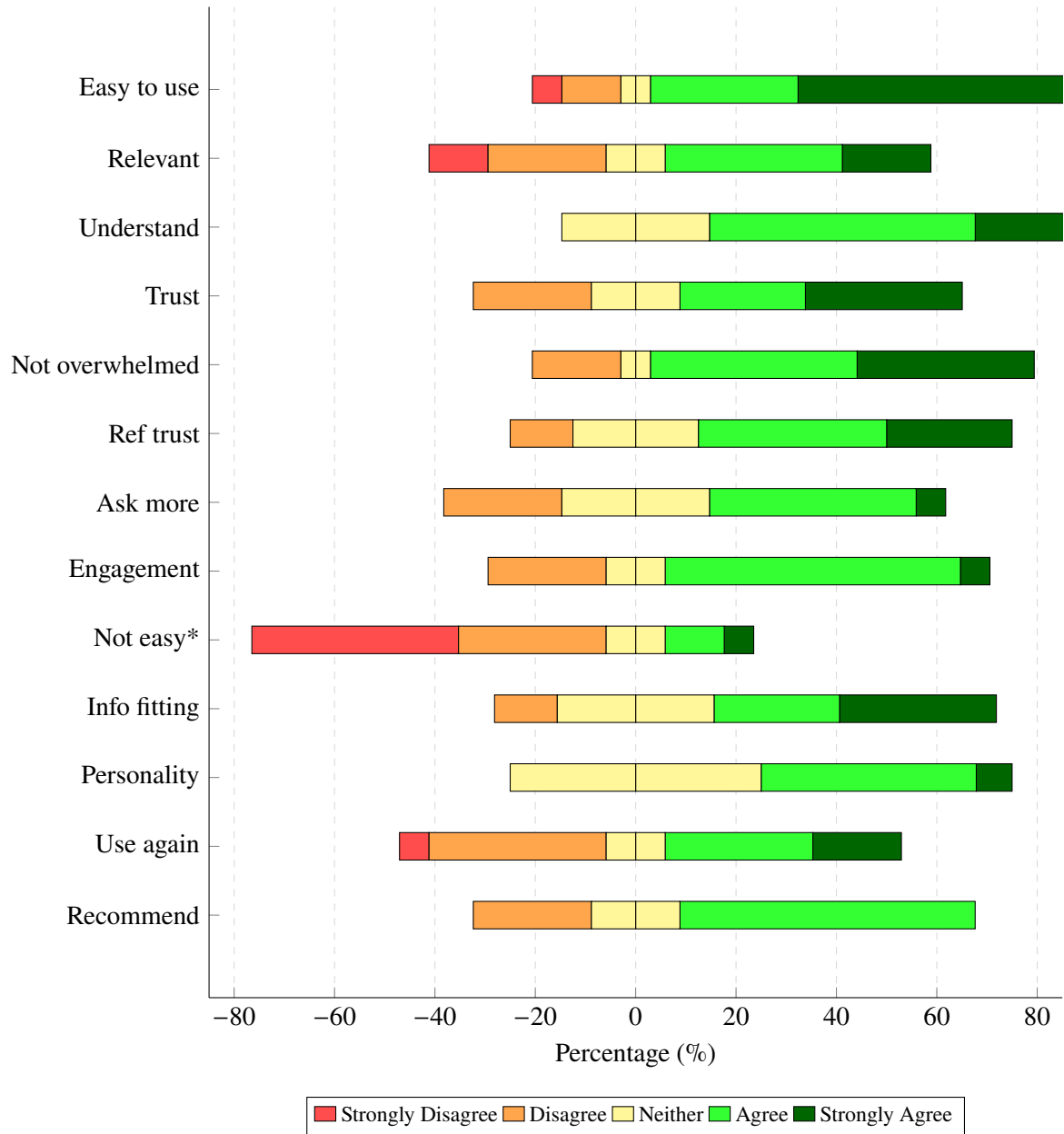


Figure 8.9: Diverging stacked bar chart of general user experience ratings. Negative responses on the left, neutral in center, positive on the right. *Reverse-scored question.

Even though the AI-tutor is received easy to use, students evaluated a neutral score to use it again. One reason for this might be the latency of the AI-tutor. When 5 students mention this issue without being prompted, it indicates that latency is a significant concern that negatively impacts the user experience. Even when the other 12 students did not write about this issue themselves, they might still be affected by it. Especially students using a lot of AI tools which the evaluation consisted of, might perceive this as negative and would not use this AI-tutor for their own studies. Students might receive latency as must-be quality and expect fast results. One indication for this might be the criteria missing in written requirements at the conducted survey in chapter 5. Over each 5 asked questions, it took an average of 37 seconds to answer a question about lecture materials, while around 48 seconds for a question about code feedback. This could be a lot higher than the response time students interacting with AI expect, as prompting for example GPT-5 takes a few seconds. While we did not change anything at the existing Ollama Service in this thesis, it is something which should be improved in the future.

Students seem to trust answers of the AI-tutor. While not adding used sources in this thesis, it is still worth mentioning. However, some students mentioned the AI-tutor not providing relevant sources in the open-ended questions. The ranking of sources provided by the AI-tutor could be improved in the future to provide more relevant sources at the top of the list.

For the code feedback, students found the feedback clear and understandable, but the AI-tutor was not able to help students identify errors in their code effectively. The current developments of AI models might improve this in the future, as code understanding and generation capabilities of AI models are rapidly improving. The model used in MEITREX might not be state of the art anymore, given the fast pace of AI development. One student even mentioned in the open ended questions that *"the LLM behind the AI-tutor does not really feel SOTA anymore"*. An improved model might be able to provide better code feedback in the future.

Additionally, the AI-tutor was not able to adapt its hints and explanations effectively to the students skill levels. While this might be due to the setup of the evaluation, where students only completed one quiz for MEITREX to determine their skill level, the AI-tutor could adapt more fine grained way in the future.

One of the novel features of the AI-tutor is its proactive feedback. While students found the proactive feedback motivating and not interrupting their workflow, some students rated its relevance rather low. This indicates that while the idea of proactive feedback is promising, there is still room for improvement in its implementation. In the evaluation, the proactive feedback was triggered a lot, which could have led to students perceiving it as less relevant and appearing at moments where it was not needed. A more fine grained approach to trigger proactive feedback in the future could improve its relevance. Additionally, the AI-tutor could learn from previous interactions with students to provide less proactive feedback to students who do not want to receive it. Moreover, the proactive feedback could be better tailored to the students current context and needs, which could improve its relevance as well.

To summarize, while the AI-tutor was generally well received by students, there are several areas for improvement. The underlying LLM could be updated to a more state of the art model to combat latency issues and improve overall performance. The code feedback feature could be enhanced to better assist students in identifying and correcting errors in their code. Hints and explanations could be more effectively adapted to the skill levels of individual students. Finally, the proactive feedback approach could be refined to increase its relevance and effectiveness.

8.4 Threats to Validity

In this section, we describe threats to validity. According to “Guidelines for conducting and reporting case study research in software engineering”, four aspects of validity should be considered: *construct, internal, external validity* and reliability

8.4.1 Construct Validity

The construct validity of this evaluation could be threatened by the design of the experimental tasks and the measurements used. While, for example, the code feedback section was designed to be completable in a reasonable timeframe, it might have resulted in a simplified coding exercise that does not fully capture the complexity of real-world coding tasks. Additionally, the evaluation relied heavily on self-reported data from participants, which can be subject to biases.

8.4.2 Internal Validity

Internal validity is threatened when a third factor affects the investigated relationship. A significant threat to the internal validity of this evaluation arises from the latency issues described in Section 8.2.6. Almost 30% of the participants mentioned latency issues in the open-ended questions. This technical limitation could result in a negative bias against the AI-tutor. Students might have been frustrated with the slow response times and rated other aspects of the tool lower as a result. Additionally, the other participants who did not mention latency issues might have also been affected by them, but did not decide to mention them in the open ended questions.

8.4.3 External Validity

This aspect of validity is concerned with the generalizability of the findings. First and foremost, the small sample size of 17 participants limits the generalizability of the results to a broader population. Moreover, the demographics of the sample introduce potential biases. Almost all participants were frequent users of AI, which may not accurately represent current software engineering students. The gender distribution was also skewed, with only 2 female participants and 14 male participants, which limits the generality of the results as well. Finally, the evaluation was conducted exclusively with software engineering students at the University of Stuttgart, which is the target group for the AI-tutor, but might not represent other Universities.

8.4.4 Reliability

A threat to reliability in this study is the relationship between the researcher and the participants. While unrelated to the SQA institute, many participants were friends of the author. This relationship could introduce a social desirability bias, where participants might have answered more favorably than they would have for an independent researcher.

All these factors should be taken into account when interpreting the results of this evaluation.

9 Conclusion

This chapter summarizes the key findings of this thesis in Section 9.1, discusses its benefits in Section 9.2 and limitations in Section 9.3, reflects on lessons learned during the research process in Section 9.4 and outlines potential directions for future work in Section 9.5.

9.1 Summary

Traditional software engineering education often relies on human instructors, who are often limited in their capacity to provide timely and personalized feedback to all students. To address this, this thesis explored the development of an AI-tutor that provides personalized and proactive feedback to students in software engineering education. Based on existing research, we designed and implemented an AI-tutor integrated into the MEITREX learning platform, which focuses on individual personalization based on the hexad player types and student skill level, while also providing proactive feedback to students.

To achieve this, a requirements elicitation phase was conducted through a literature review and a survey with 34 students, gathering key insights about the desired features and functionalities of an AI-tutor in software engineering education. The results of the survey indicated a potential benefit of proactive feedback, which was then added to the design of the AI-tutor. Additionally, the survey indicated that students would actively use an AI-tutor integrated into their learning platform, while also trusting the feedback provided by the AI-tutor. Furthermore, students were accepting of the AI-tutor never providing direct code solutions, which is now a key principle of the designed AI-tutor. In sum, the answers to RQ4.1 – RQ4.5 in 5.2 – 5.6 answer research question RQ1.1.

These insights informed the design of the AI-tutor, which was then implemented into MEITREX. This design and implementation represent a novel iteration of an AI-tutor that provides personalized and proactive feedback in software engineering education.

Finally, the implemented AI-tutor was evaluated through an experimental study with 17 students. The evaluation results indicated that students benefited from the personalized feedback provided by the AI-tutor, which leads to increased motivation and could lead to potentially improved learning outcomes. However, the results regarding the effectiveness of proactive feedback were mixed, suggesting the need for further research in this area. To answer RQ1.2, we found that some students perceived proactive feedback as helpful, while others did not find it beneficial. To fully optimize the proactive feedback approach of the AI-tutor, further improvements are required to increase the relevance of proactive feedback.

Regarding RQ1.3, we demonstrated that utilizing prompt engineering to adapt to competence levels and hexad player types creates an effective AI-tutor. Students reported that the feedback fitted their personality well, leading to higher motivation and engagement.

9.2 Benefits

The design and implementation of the AI-tutor in MEITREX provides benefits to both students and educators in software engineering education. Students benefit from receiving personalized feedback tailored to their individual player types and skill levels, which can enhance their learning experience and motivation, while also receiving feedback in a timely manner, no matter at what time they study. Additionally, students benefit from proactive feedback, especially when they are not actively seeking help, which can help them identify and address issues in their learning process.

Educators benefit from the scalability of the AI-tutor, which can provide consistent feedback to a large number of students without requiring additional human resources. The pedagogical guardrails of no code solutions being provided also ensure that the AI-tutor supports effective learning practices.

9.3 Limitations

Despite the promising results, this thesis has several limitations. First, the evaluation of the AI-tutor was conducted with a relatively small sample size of 17 students, which may limit the generalizability of the findings. Similarly, the survey conducted during the requirements elicitation phase had a limited sample size of 34 students, which may not fully capture the diverse perspectives and needs of all students in software engineering education. Second, the effectiveness of proactive feedback provided by the AI-tutor showed mixed results, indicating that further research is needed to optimize this aspect of the AI-tutor. Third, the AI-tutor relies on the capabilities of the underlying LLM, which may have limitations in understanding complex programming concepts or providing accurate feedback in all scenarios. Lastly, the implementation of the proactive feedback mechanism with polling as in other parts of MEITREX may introduce latency in feedback delivery if scaled to a larger number of students, which could then impact the timeliness of the feedback provided.

9.4 Lessons Learned

Currently, MEITREX is using a mixture of synchronous GraphQL requests and asynchronous events to communicate between microservices. When extending MEITREX with our contributions, this setup resulted in multiple challenges: First, the synchronous requests between services result in higher coupling, potentially leading to larger failures in the system if one of the services fails, which is undesirable in this microservice architecture. However, even bigger challenges resulted from the current code sharing setup used: Some services provide API utilities, which are then consumed by other services as a code dependency. This compile-time coupling between individually developed services in multiple repositories introduced dependency challenges when updating services and significantly hindered our implementation progress. To mitigate these issues, we started migrating to use more event based communication, with the accepted drawback of more data duplication between services. For future projects, we highly recommend several measure: First, to decrease coupling, asynchronous events should be preferred. Additionally, code dependencies between the services should be completely avoided. If necessary, a utility library could be introduced.

It proved to be more challenging than expected to find participants for the evaluation. We think this could be because of the higher time investment compared to the survey. It might be beneficial to offer higher incentives for participation in future studies.

9.5 Future Work

Multiple areas for future work can be identified based on the findings and limitations of this thesis. First and foremost, further research is needed to optimize the proactive feedback mechanism of the AI-tutor. This could involve exploring different trigger mechanisms for proactive feedback, to better identify when students would benefit most from such feedback. Additionally, proactive feedback could be improved to be more relevant to the student's current context and learning needs.

Moreover, future work could focus on enhancing the capabilities of the underlying LLM to better understand complex programming concepts and provide more accurate and relevant feedback, especially in programming tasks.

Finally, conducting larger-scale evaluations with diverse student populations would help to validate the effectiveness of the AI-tutor and its generalizability across different contexts in software engineering education.

Bibliography

- [BBB+93] C. Berger, R. Blauth, D. Boger, et al. “Kano’s methods for understanding customer-defined quality”. In: (1993) (cit. on pp. 20, 22).
- [BCD89] J. S. Brown, A. Collins, P. Duguid. “Situated Cognition and the Culture of Learning”. In: *Educational Researcher* 18.1 (1989), pp. 32–42. DOI: [10.3102/0013189X018001032](https://doi.org/10.3102/0013189X018001032). eprint: <https://doi.org/10.3102/0013189X018001032>. URL: <https://doi.org/10.3102/0013189X018001032> (cit. on p. 3).
- [BFK24] P. Bassner, E. Frankford, S. Krusche. “Iris: An AI-Driven Virtual Tutor for Computer Science Education”. In: *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. ITiCSE 2024. Milan, Italy: Association for Computing Machinery, 2024, pp. 394–400. ISBN: 9798400706004. DOI: [10.1145/3649217.3653543](https://doi.org/10.1145/3649217.3653543). URL: <https://doi.org/10.1145/3649217.3653543> (cit. on pp. 11, 15, 37).
- [BMR+20] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems*. Vol. 33. 2020, pp. 1877–1901 (cit. on pp. 5, 6).
- [Bra21] V. M. Bradley. “Learning Management System (LMS) use with online instruction”. In: *International Journal of Technology in Education (IJTE)* 4.1 (2021), pp. 68–92. DOI: [10.46328/ijte.36](https://doi.org/10.46328/ijte.36). URL: <https://doi.org/10.46328/ijte.36> (cit. on p. 8).
- [CB18] E. A. Crisp, C. J. Bonk. “Defining the Learner Feedback Experience”. In: *TechTrends* 62.6 (2018), pp. 585–593. ISSN: 1559-7075. DOI: [10.1007/s11528-018-0264-y](https://doi.org/10.1007/s11528-018-0264-y). URL: <https://doi.org/10.1007/s11528-018-0264-y> (cit. on pp. 1, 3, 4, 15).
- [CSD19] C. Casalnuovo, K. Sagae, P. Devanbu. “Studying the difference between natural and programming language corpora”. In: *Empirical Software Engineering* 24.4 (2019), pp. 1823–1868. ISSN: 1573-7616. DOI: [10.1007/s10664-018-9669-7](https://doi.org/10.1007/s10664-018-9669-7). URL: <https://doi.org/10.1007/s10664-018-9669-7> (cit. on p. 6).
- [DM26a] M. Dallinger, N. Meißner. *Evaluation AI-tutor in MEITREX with Adaptive Gamification*. Zenodo, Jan. 2026. DOI: [10.5281/zenodo.18215499](https://doi.org/10.5281/zenodo.18215499). URL: <https://doi.org/10.5281/zenodo.18215499> (cit. on pp. 61, 101).
- [DM26b] M. Dallinger, N. Meißner. *Survey AI-tutoring in software engineering education*. Zenodo, Jan. 2026. DOI: [10.5281/zenodo.18215764](https://doi.org/10.5281/zenodo.18215764). URL: <https://doi.org/10.5281/zenodo.18215764> (cit. on pp. 25, 97).
- [Fra16] G. M. Francom. “Principles for task-centered instruction”. In: *Instructional-Design Theories and Models, Volume IV*. Routledge, 2016, pp. 65–92 (cit. on pp. 4, 15).

- [FSB+24] E. Frankford, C. Sauerwein, P. Bassner, S. Krusche, R. Breu. “AI-Tutoring in Software Engineering Education”. In: *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*. ICSE-SEET '24. Lisbon, Portugal: Association for Computing Machinery, 2024, pp. 309–319. ISBN: 9798400704987. DOI: [10.1145/3639474.3640061](https://doi.org/10.1145/3639474.3640061). URL: <https://doi.org/10.1145/3639474.3640061> (cit. on p. 11).
- [GG22] N. Garud-Patkar, N. Gautam. “Transactional distance during COVID-19: Examining student relationships & learning strategies during the emergency shift to online learning”. In: *International Communication Research Journal* 57.1 (2022), pp. 20–37 (cit. on p. 22).
- [GHJv17] A. Gerdes, B. Heeren, J. Jeurig, L. T. van Binsbergen. “Ask-Elle: an Adaptable Programming Tutor for Haskell Giving Automated Feedback”. In: *International Journal of Artificial Intelligence in Education* 27.1 (2017), pp. 65–100. ISSN: 1560-4306. DOI: [10.1007/s40593-015-0080-x](https://doi.org/10.1007/s40593-015-0080-x). URL: <https://doi.org/10.1007/s40593-015-0080-x> (cit. on p. 13).
- [Gir23] L. Giray. “Prompt Engineering with ChatGPT: A Guide for Academic Writers”. In: *Annals of Biomedical Engineering* 51.12 (2023), pp. 2629–2633. ISSN: 1573-9686. DOI: [10.1007/s10439-023-03272-4](https://doi.org/10.1007/s10439-023-03272-4). URL: <https://doi.org/10.1007/s10439-023-03272-4> (cit. on p. 6).
- [Git24] GitHub. *Using GitHub Copilot Chat*. 2024. URL: <https://docs.github.com/en/copilot/using-github-copilot/copilot-chat> (cit. on pp. 6, 7).
- [GS05] G. Gibbs, C. Simpson. “Conditions Under Which Assessment Supports Students’ Learning”. In: *Learning and Teaching in Higher Education* 1 (2005). Learning and Teaching in Higher Education (LATHE) was published by the University of Gloucestershire between 2005 and 2011. LATHE was edited by Phil Gravestock and Michele Hills. Publication has ceased, the final issue was issue No.5, pp. 3–31. ISSN: 1742-240X. URL: <https://eprints.glos.ac.uk/3609/> (cit. on p. 3).
- [GvRv24] S. Groothuijsen, A. van den Beemt, J. C. Remmers, L. W. van Meeuwen. “AI chatbots in programming education: Students’ use in a scientific computing course and consequences for learning”. In: *Computers and Education: Artificial Intelligence* 7 (2024), p. 100290. ISSN: 2666-920X. DOI: <https://doi.org/10.1016/j.caeai.2024.100290>. URL: <https://www.sciencedirect.com/science/article/pii/S2666920X24000936> (cit. on p. 7).
- [Hat15] J. Hattie. “The applicability of visible learning to higher education”. In: *Scholarship of Teaching and Learning in Psychology* 1.1 (2015), pp. 79–91. DOI: [10.1037/stl0000021](https://doi.org/10.1037/stl0000021) (cit. on p. 3).
- [HBG+16] A. Hindle, E. T. Barr, M. Gabel, Z. Su, P. Devanbu. “On the naturalness of software”. In: *Commun. ACM* 59.5 (Apr. 2016), pp. 122–131. ISSN: 0001-0782. DOI: [10.1145/2902362](https://doi.org/10.1145/2902362). URL: <https://doi.org/10.1145/2902362> (cit. on p. 6).
- [HT07] J. Hattie, H. Timperley. “The power of feedback”. In: *Review of educational research* 77.1 (2007), pp. 81–112 (cit. on p. 4).
- [HT99] A. Hunt, D. Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, 1999. ISBN: 978-0135957059 (cit. on p. 42).

- [Imp25] J. Impiö. “Developing a Proactive Programming Assistant Leveraging an LLM for Personalized Real-time Feedback”. In: (2025) (cit. on p. 13).
- [JM07] W. Jaehrig, M. L. Miller. “Feedback Types in Programmed Instruction: A Systematic Review”. In: *The Psychological Record* 57.2 (2007), pp. 219–232. issn: 2163-3452. doi: [10.1007/BF03395573](https://doi.org/10.1007/BF03395573). url: <https://doi.org/10.1007/BF03395573> (cit. on p. 3).
- [KJH16] H. Keuning, J. Jeuring, B. Heeren. “Towards a Systematic Review of Automated Feedback Generation for Programming Exercises”. In: *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE ’16. Arequipa, Peru: Association for Computing Machinery, 2016, pp. 41–46. isbn: 9781450342315. doi: [10.1145/2899415.2899422](https://doi.org/10.1145/2899415.2899422). url: <https://doi.org/10.1145/2899415.2899422> (cit. on pp. 4, 15).
- [Kul77] R. W. Kulhavy. “Feedback in Written Instruction”. In: *Review of Educational Research* 47.2 (1977), pp. 211–232. doi: [10.3102/00346543047002211](https://doi.org/10.3102/00346543047002211). eprint: <https://doi.org/10.3102/00346543047002211>. url: <https://doi.org/10.3102/00346543047002211> (cit. on pp. 3, 4, 15).
- [LAJ05] E. Lahtinen, K. Ala-Mutka, H.-M. Järvinen. “A study of the difficulties of novice programmers”. In: *Acm sigcse bulletin* 37.3 (2005), pp. 14–18 (cit. on p. 4).
- [LDM+23] J. Leinonen, P. Denny, S. MacNeil, S. Sarsa, S. Bernstein, J. Kim, A. Tran, A. Hellas. “Comparing code explanations created by students and large language models”. In: *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. 2023, pp. 124–130 (cit. on pp. 5, 6).
- [LKK25] D. Lohr, H. Keuning, N. Kiesler. “You’re (Not) My Type- Can LLMs Generate Feedback of Specific Types for Introductory Programming Tasks?” In: *Journal of Computer Assisted Learning* 41.1 (2025). e13107 JCAL-24-434.R1, e13107. doi: <https://doi.org/10.1111/jcal.13107>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/jcal.13107>. url: <https://onlinelibrary.wiley.com/doi/abs/10.1111/jcal.13107> (cit. on p. 5).
- [LPP+20] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474. url: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf (cit. on p. 6).
- [LSSD24] M. Liffiton, B. E. Sheese, J. Savelka, P. Denny. “CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes”. In: *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. Koli Calling ’23. Koli, Finland: Association for Computing Machinery, 2024. isbn: 9798400716539. doi: [10.1145/3631802.3631830](https://doi.org/10.1145/3631802.3631830). url: <https://doi.org/10.1145/3631802.3631830> (cit. on pp. 12, 15).
- [LYF+23] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, G. Neubig. “Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing”. In: *ACM Computing Surveys* 55.9 (2023), pp. 1–35 (cit. on p. 6).

- [LZL+24] R. Liu, C. Zenke, C. Liu, A. Holmes, P. Thornton, D. J. Malan. “Teaching CS50 with AI: Leveraging Generative Artificial Intelligence in Computer Science Education”. In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2024. Portland, OR, USA: Association for Computing Machinery, 2024, pp. 750–756. ISBN: 9798400704239. DOI: [10.1145/3626252.3630938](https://doi.org/10.1145/3626252.3630938). URL: <https://doi.org/10.1145/3626252.3630938> (cit. on p. 12).
- [Mar15] A. Marczewski. “User types”. In: *Even Ninja Monkeys Like to Play: Gamification, Game Thinking and Motivational Design*. Vol. 1. 2015, pp. 65–80 (cit. on pp. 1, 7–9, 16).
- [MBSB25] N. Meißner, P. Bredl, S. Speth, S. Becker. “Enhancing Motivation in Software Engineering Education through Gamified Agile Project-based Learning”. In: *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. FSE Companion ’25. Clarion Hotel Trondheim, Trondheim, Norway: Association for Computing Machinery, 2025, pp. 835–846. ISBN: 9798400712760. DOI: [10.1145/3696630.3727241](https://doi.org/10.1145/3696630.3727241). URL: <https://doi.org/10.1145/3696630.3727241> (cit. on p. 8).
- [Mei24] N. Meißner. “MEITREX - Gamified and Adaptive Intelligent Tutoring in Software Engineering Education”. In: *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*. ICSE-Companion ’24. Lisbon, Portugal: Association for Computing Machinery, 2024, pp. 198–200. ISBN: 9798400705021. DOI: [10.1145/3639478.3639804](https://doi.org/10.1145/3639478.3639804). URL: <https://doi.org/10.1145/3639478.3639804> (cit. on pp. 8, 9).
- [MEI25] MEITREX. *MEITREX: Modular Embedded Intelligent Tutoring and Remote Education eXperience*. Version 1.0. Source code and microservices architecture. GitHub, 2025. URL: <https://github.com/MEITREX> (visited on 12/27/2025) (cit. on pp. 9, 40, 42).
- [Mer13] M. D. Merrill. *First Principles of Instruction: Identifying and Designing Effective, Efficient and Engaging Instruction*. San Francisco: Pfeiffer, 2013 (cit. on pp. 3, 15).
- [MGF+20] S. Marwan, G. Gao, S. Fisk, T. W. Price, T. Barnes. “Adaptive Immediate Feedback Can Improve Novice Programming Engagement and Intention to Persist in Computer Science”. In: *Proceedings of the 2020 ACM Conference on International Computing Education Research*. ICER ’20. Virtual Event, New Zealand: Association for Computing Machinery, 2020, pp. 194–203. ISBN: 9781450370929. DOI: [10.1145/3372782.3406264](https://doi.org/10.1145/3372782.3406264). URL: <https://doi.org/10.1145/3372782.3406264> (cit. on pp. 12, 13).
- [MLK25] M. Mueller, C. List, 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*. ECSEE ’25. Association for Computing Machinery, 2025, pp. 1–10. ISBN: 9798400712821. DOI: [10.1145/3723010.3723034](https://doi.org/10.1145/3723010.3723034). URL: <https://doi.org/10.1145/3723010.3723034> (cit. on pp. 13, 14).
- [MTH+23] S. MacNeil, A. Tran, A. Hellas, J. Kim, S. Sarsa, P. Denny, S. Bernstein, J. Leinonen. “Experiences from using code explanations generated by large language models in a web software development e-book”. In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 2023, pp. 931–937 (cit. on p. 5).

- [MTS+26] N. Meißner, L. Trautwein, S. Speth, N. N. Koch, S. Becker. “Utilizing Adaptive Gamification in an Intelligent Tutoring System for Software Engineering Education”. In: *Proceedings of the 48th International Conference on Software Engineering: Software Engineering Education and Training*. ICSE-SEET '26. Rio de Janeiro, Brazil: Association for Computing Machinery, 2026. doi: [10.1145/3786580.3786945](https://doi.org/10.1145/3786580.3786945). URL: <https://doi.org/10.1145/3786580.3786945> (cit. on pp. 8, 9).
- [OWJ+22] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, R. Lowe. “Training language models to follow instructions with human feedback”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 27730–27744. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf (cit. on p. 5).
- [RH09] P. Runeson, M. Höst. “Guidelines for conducting and reporting case study research in software engineering”. In: *Empir. Softw. Eng.* 14.2 (2009), pp. 131–164 (cit. on p. 73).
- [RML17] C. M. Reigeluth, R. D. Myers, D. Lee. “The learner-centered paradigm of education”. In: *Instructional-design theories and models: The learner-centered paradigm of education*. Ed. by C. M. Reigeluth, B. J. Beatty, R. D. Myers. Vol. IV. New York, NY: Routledge, 2017, pp. 5–32 (cit. on p. 4).
- [Shu08] V. J. Shute. “Focus on Formative Feedback”. In: *Review of Educational Research* 78.1 (2008), pp. 153–189. doi: [10.3102/0034654307313795](https://doi.org/10.3102/0034654307313795). eprint: <https://doi.org/10.3102/0034654307313795>. URL: <https://doi.org/10.3102/0034654307313795> (cit. on p. 4).
- [Tur07] A. M. Turing. “Computing machinery and intelligence”. In: *Parsing the Turing test: Philosophical and methodological issues in the quest for the thinking computer*. Springer, 2007, pp. 23–65 (cit. on p. 38).
- [VP18] M. Verleger, J. Pembridge. “A Pilot Study Integrating an AI-driven Chatbot in an Introductory Programming Course”. In: *2018 IEEE Frontiers in Education Conference (FIE)*. 2018, pp. 1–4. doi: [10.1109/FIE.2018.8659282](https://doi.org/10.1109/FIE.2018.8659282) (cit. on p. 6).
- [VSP+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 5).
- [Whi23] J. White. “A prompt pattern catalog to enhance prompt engineering with ChatGPT”. In: *arXiv preprint arXiv:2302.11382* (2023) (cit. on p. 6).
- [WRZ+20] W. Wang, Y. Rao, R. Zhi, S. Marwan, G. Gao, T. W. Price. “Step Tutor: Supporting Students through Step-by-Step Example-Based Feedback”. In: *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE '20. Trondheim, Norway: Association for Computing Machinery, 2020, pp. 391–397. ISBN: 9781450368742. doi: [10.1145/3341525.3387411](https://doi.org/10.1145/3341525.3387411). URL: <https://doi.org/10.1145/3341525.3387411> (cit. on p. 5).
- [WW17] W. R. Watson, S. L. Watson. “Principles for Personalized Instruction”. In: *Instructional-Design Theories and Models*. Ed. by C. M. Reigeluth, B. J. Beatty, R. D. Myers. Hillsdale: Routledge, 2017, pp. 93–120 (cit. on p. 4).

Bibliography

All links were last followed on January 11, 2025.

A Survey Appendix

A.1 Survey Questionnaire (translated to English)

The questionnaire used in the survey is shown in the following sections.

A.1.1 Part 1

Question 1: course of study

Possible answers:

- Informatik
- Softwaretechnik / Software Engineering
- Data Science / AIDS
- Medieninformatik
- Maschinelle Sprachverarbeitung
- other:

The last option allowed the student to write down their own study program.

Question 2: Progress in studies

Possible answers:

- up to the 3rd semester, Bachelor's degree
- 4th and following semester, Bachelor's degree
- in the Master's programme
- Finished with the Master

Question 3: Did you study computer science at school before starting university?

Possible answers:

- Yes
- No

Question 4: Largest project programmed to date?

Note: In which you (also partly) participated in programming. Possible answers:

- ≤ 300 lines of code
- $300 < \text{number of lines of code} \leq 2000$ lines of code
- $2000 < \text{number of lines of code} \leq 10000$ lines of code
- > 10000 lines of code

Question 5: How often do you actively use AI? e.g. ChatGPT

Possible answers:

- daily
- about 3 times a week
- about once a week
- very rarely

Question 6: Have you ever had a tutoring job during a semester?

Possible answers:

- Yes
- No

A.1.2 Part 2

Question 7: How much do you generally trust the technical accuracy of your tutor's answers?

Possible answers:

- I completely trust the answer
- I tend to trust the answer
- I'm undecided / I don't know
- I tend not to trust the answer
- I would not trust the answer at all

Question 8: Who are you most likely to ask for help when you have problems solving programming tasks?

Note: Please rank the options according to your preference (1=highest priority, 4=lowest priority).
Possible answers:

- Tutor
- I prefer to look at the script/lecture slides again and try it myself
- Fellow students
- AI chatbot (e.g. ChatGPT)

Question 9: Do you actively ask for help when you encounter problems in programming tasks that you cannot solve on your own?

Possible answers:

- Very often
- Quite often
- Occasionally
- Quite rarely
- Never

Question 10: For what reasons have you asked a tutor for help in the past, or would you do so in the future?

Note: Multiple choice possible. Possible answers:

- I had questions about the content
- I needed help with an assignment
- I wanted feedback on a solution/line of thinking
- I was unsure about the content of an assignment
- I was unsure how to get started
- I had motivation problems
- I have never contacted a tutor
- other:

The last option allowed the student to write down their own reason.

A.1.3 Part 3

Question 11: To what extent would you trust an automatically generated answer if it is supported by references to lecture slides or solutions to exercises?

Possible answers:

- I would completely trust the answer
- I would tend to trust the answer
- I'm undecided / I don't know
- I would tend not to trust the answer
- I would not trust the answer at all

Question 12: Would you actively use an AI-based application to check your source code for errors or areas for improvement?

Possible answers:

- Yes, definitely
- Probably yes
- Undecided / Don't know
- Probably no
- No, definitely not

Question 13: For what reasons would you contact an AI tutor?

Note: Multiple choice possible. Possible answers:

- I have comprehension questions about the content
- I need help with an assignment
- I want feedback on a solution/line of thinking
- I was unsure about the content of an assignment
- I am unsure how to get started
- I have motivation problems
- I would never use an AI tutor

Question 14: Imagine you are in the process of solving a problem and have already fully understood the topic. Now you receive a detailed explanation of this topic. How does that make you feel?

Possible answers:

- I find this helpful and supportive
- I find it unnecessary, but not problematic
- I take note of it, but it doesn't bother me
- I find it annoying or patronizing

Question 15: Imagine you are focused and motivated on a programming task that you find challenging but solvable. You want to work out the solution on your own. Now your tutor or fellow student gives you an unsolicited hint that anticipates a central part of the task. How do you feel about such a situation?

Possible answers:

- Rather helpful – I benefit even without specifically asking for it
- Neutral – I'm undecided
- Rather disturbing – I would have asked for help myself
- Very disturbing – I would have preferred to continue without any suggestion

Question 16: Imagine you are working on a difficult programming task. You have been working on it for a long time and your test cases are still failing. Now you automatically receive a tip for solving your task via a chat interface. This response begins with motivational feedback, e.g. "You are on the right track" or "Tackling a problem several times shows perseverance". How would you feel if you received motivational words in this situation through the chat window?

Possible answers:

- I think that is excellent
- I think that is acceptable
- I can accept that
- I expect that
- I think that is bad

Question 17: Imagine you are working on a difficult programming task. You have been working on it for a long time and your test cases are still failing. Now you DO NOT automatically receive a tip for solving your task via a chat interface. This response begins with motivational feedback, e.g. “You are on the right track” or “Tackling a problem several times shows perseverance”. How would you feel if you received NO motivational words in this situation through the chat window?

Possible answers:

- I think that is excellent
- I think that is acceptable
- I can accept that
- I expect that
- I think that is bad

Question 18: What is your opinion on receiving feedback on a programming task that identifies a stylistic flaw (e.g. unclear naming, poor structure, etc.) and you are only given a hint as to what should be improved, but without being told the exact code solution?

Possible answers:

- Very good – I would like to work out the solution myself
- Rather good – a hint is sufficient for me
- Neutral – it depends on the context
- Rather poor – I would prefer an example illustrating such a style
- Very poor – I want to see the complete solution

A.1.4 Part 4

Question 19: Finally, can you think of anything that would be a key point here?

Note: Free text questions. Possible answers:

- What should I perhaps implement?
- What is important to you in this regard?
- What would you find particularly bad?

A.2 Survey Questionnaire (original in German)

The questionnaire used in the survey is shown in the following sections.

A.2.1 Part 1

Frage 1: Studiengang

Possible answers:

- Informatik
- Softwaretechnik / Software Engineering
- Data Science / AIDS
- Medieninformatik
- Maschinelle Sprachverarbeitung
- Sonstiges:

The last option allowed the student to write down their own study program.

Frage 2: Fortschritt im Studium

Possible answers:

- bis zum 3. Semester, Bachelor
- Ab 4. Semester, Bachelor
- im Master
- Fertig mit dem Master

Frage 3: Hattest du Informatik in der Schule vor Studienbeginn?

Possible answers:

- Ja
- Nein

Frage 4: Größtes bisher programmiertes Projekt?

Note: An dem du (auch in Teilen) mitprogrammiert hast. Possible answers:

- ≤ 300 Zeilen
- $300 \text{ Zeilen} < \# \text{Zeilen} \leq 2000$ Zeilen
- $2000 \text{ Zeilen} < \# \text{Zeilen} \leq 10000$ Zeilen
- > 10000 Zeilen

Frage 5: Wie häufig nutzt du aktiv KI? z.B. ChatGPT

Possible answers:

- täglich
- ca. 3x die Woche
- ca. 1x die Woche
- sehr selten

Frage 6: Hattest du schon in einem Semester einen Tutorenjob?

Possible answers:

- Ja
- Nein

A.2.2 Part 2

Frage 7: Wie sehr vertraust du in der Regel der fachlichen Richtigkeit der Antworten deines Tutors?

Possible answers:

- Ich vertraue der Antwort voll und ganz
- Ich vertraue der Antwort eher
- Ich bin unentschieden / Weiß nicht
- Ich vertraue der Antwort eher nicht
- Ich würde der Antwort überhaupt nicht vertrauen

Frage 8: Wen frägst du am ehesten nach Hilfe, wenn du Probleme beim Lösen von Programmieraufgaben hast?

Note: Bitte nummeriere die Optionen nach deiner Präferenz (1=höchste Priorität, 4=niedrigste Priorität). Possible answers:

- Tutor
- Ich schaue lieber nochmal ins Script / In die Vorlesungsfolien und versuche es selbst nochmal
- Kommilitonen
- KI-Chatbot (z.B. ChatGPT)

Frage 9: Frägst du aktiv nach Hilfe, wenn du bei Programmieraufgaben auf Probleme stößt, die du nicht selbstständig lösen kannst?

Possible answers:

- Sehr häufig
- Eher häufig
- Gelegentlich
- Eher selten
- Nie

Frage 10: Aus welchen Gründen hast du in der Vergangenheit einen Tutor gefragt bzw. würdest dies in Zukunft?

Note: Mehrfachauswahl möglich. Possible answers:

- Ich hatte Verständnisfragen zu Inhalten
- Ich brauchte Hilfe bei einer Aufgabe
- Ich wollte Feedback zu einer Lösung / einem Gedankengang
- Ich war unsicher wegen dem Inhalt einer Abgabe
- Ich war unsicher, wie ich anfangen soll
- Ich hatte motivationale Probleme
- Ich habe nie einen Tutor kontaktiert
- Sonstiges:

The last option allowed the student to write down their own reason.

A.2.3 Part 3

Frage 11: Inwiefern würdest du einer automatisiert generierten Antwort vertrauen, wenn diese durch Verweise auf Vorlesungsfolien oder Lösungen von Übungsaufgaben gestützt wird?

Possible answers:

- Ich würde der Antwort voll und ganz vertrauen
- Ich würde der Antwort eher vertrauen
- Ich bin unentschieden / Weiß nicht
- Ich würde der Antwort eher nicht vertrauen
- Ich würde der Antwort überhaupt nicht vertrauen

Frage 12: Würdest du eine KI-basierte Anwendung aktiv nutzen, um deinen Quellcode auf Fehler oder Verbesserungsmöglichkeiten überprüfen zu lassen?

Possible answers:

- Ja, auf jeden Fall
- Eher ja
- Unentschieden / Weiß nicht
- Eher nein
- Nein, auf keinen Fall

Frage 13: Aus welchen Gründen würdest du einen KI-Tutor kontaktieren?

Note: Mehrfachauswahl möglich. Possible answers:

- Ich habe Verständnisfragen zu Inhalten
- Ich brauche Hilfe bei einer Aufgabe
- Ich will Feedback zu einer Lösung / einem Gedankengang
- Ich bin unsicher wegen dem Inhalt einer Abgabe
- Ich bin unsicher, wie ich anfangen soll
- Ich hab motivationale Probleme
- Ich würde nie einen KI-Tutor verwenden

Frage 14: Stell dir vor, du bist gerade dabei eine Aufgabe zu lösen und hast das Thema schon vollständig verstanden. Nun bekommst du eine ausführliche Erklärung zu diesem Thema. Wie fühlst du dich dabei?

Possible answers:

- Ich empfinde das als hilfreich und unterstützend
- Ich finde es überflüssig, aber nicht problematisch
- Ich nehme es zur Kenntnis, aber es stört mich nicht
- Ich empfinde es als störend oder bevormundend
- Ich würde mir wünschen, solche Erklärungen nicht zu bekommen

Frage 15: Stelle dir vor, du bist konzentriert und motiviert an einer Programmieraufgabe, die du zwar als anspruchsvoll, aber lösbar empfindest. Du möchtest die Lösung eigenständig erarbeiten. Nun gibt dir dein Tutor oder Kommilitone ungefragt einen Hinweis, der einen zentralen Teil der Aufgabe vorwegnimmt. Wie empfindest du eine solche Situation?

Possible answers:

- Sehr hilfreich - ich schätze spontane Unterstützung
- Eher hilfreich - ich profitiere auch ohne konkrete Nachfrage
- Neutral - ich bin unentschieden
- Eher störend - ich hätte mir Unterstützung selbst geholt
- Sehr störend - ich hätte lieber ohne Hinweis weitergearbeitet

Frage 16: Stell dir vor, du bearbeitest eine schwierige Programmieraufgabe. Du bist bereits eine längere Zeit an dieser Aufgabe und deine Testfälle sind weiterhin fehlerhaft. Nun bekommst du durch ein Chatinterface automatisch einen Tipp zum Lösen deiner Aufgabe. Diese Antwort beginnt mit einer motivierenden Rückmeldung, z. B. 'Du bist auf dem richtigen Weg,' oder 'Ein Problem mehrmals anzugehen zeigt von Durchhaltevermögen'. Wie würdest du dich fühlen, wenn du motivierende Worte in dieser Situation durch das Chatfenster bekommst?

Possible answers:

- Ich finde das großartig
- Ich finde das in Ordnung
- Ich kann das akzeptieren
- Ich erwarte das
- Ich finde das schlecht

Frage 17: Stell dir vor, du bearbeitest eine schwierige Programmieraufgabe. Du bist bereits eine längere Zeit an dieser Aufgabe und deine Testfälle sind weiterhin fehlerhaft. Nun bekommst du durch ein Chatinterface automatisch einen Tipp zum Lösen deiner Aufgabe. Diese Antwort beginnt OHNE eine motivierenden Rückmeldung, (wie z. B. 'Du bist auf dem richtigen Weg.' oder 'Ein Problem mehrmals anzugehen zeigt von Durchhaltevermögen'). Wie würdest du dich fühlen, wenn du KEINE motivierende Worte in dieser Situation durch das Chatfenster bekommst?

Possible answers:

- Ich finde das großartig
- Ich finde das in Ordnung
- Ich kann das akzeptieren
- Ich erwarte das
- Ich finde das schlecht

Frage 18: Was ist deine Meinung dazu, wenn du Feedback zu einer Programmieraufgabe bekommst, das einen stilistischen Mangel erkennt (z. B. unklare Benennung, schlechte Struktur, etc.) und du lediglich einen Hinweis bekommst, was verbessert werden sollte, aber ohne, dass du die genaue Code-Lösung gesagt bekommst.

Possible answers:

- Sehr gut - ich möchte selbst die Lösung erarbeiten
- Eher gut - ein Hinweis reicht mir aus
- Neutral - kommt auf den Kontext an
- Eher schlecht - ich hätte gerne ein Beispiel, wie so ein Stil aussieht
- Sehr schlecht - ich will die vollständige Lösung sehen

A.2.4 Part 4

Frage 19: Fällt dir abschließend etwas ein, was ein zentraler Punkt hierfür wäre?

Note: Freitext-Fragen. Possible answers:

- Was sollte ich vielleicht implementieren?
- Was ist dir dabei wichtig?
- Was fändest du besonders schlimm?

A.3 Raw Survey data

Raw data from the eval can be found here [DM26b].

Table A.1: Distribution of students by course of study and study progress

Course of Study	Bachelor ≤ 3	Bachelor > 3	Master	Total (%)
Informatik	3	13	3	19 (55.88%)
Softwaretechnik / Software Engineering	5	0	6	11 (32.35%)
Data Science / AIDS	0	1	0	1 (2.94%)
Medieninformatik	1	0	0	1 (2.94%)
Maschinelle Sprachverarbeitung	1	0	0	1 (2.94%)
Mathematik	0	0	1	1 (2.94%)

Table A.2: Biggest programmed project by students

Progress in studies	< 300 LoC	300 LoC < #LoC \leq 2000 LoC	2000 LoC < #LoC \leq 10000 LoC	> 2000 LoC
bachelor ≤ 3	2	6	1	1
bachelor > 4	1	2	7	4
master	0	1	3	6
total	3	9	11	11

B Evaluation Appendix

B.1 Evaluation Questionnaire

Questions 8–12, 14–16, and 18–38 used a 5-point Likert scale with the following options: Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree, Strongly Agree. Additionally, a *No Answer* option was available for all Likert scale questions. Questions 7, 13, and 17 were Yes/No questions asking if the participant completed the respective section. The following questions of each section were only shown if the participant answered "Yes" to these questions. Questions 39–42 were open-ended questions. Question 6 asked for the MEITREX username to be able to link survey responses with usage data, as well as player type data.

B.1.1 Demographic Questions

1. Course of study
2. In which semester are you?
3. How old are you?
4. Which of the following best represents your gender identity?
5. How often do you use AI actively?
6. Username of MEITREX account

B.1.2 Code Feedback Section

7. Did you finish the instructions and are ready for the evaluation for it? (Yes/No)
8. The AI Tutor's coding feedback was clear and understandable
9. The AI Tutor's suggestions helped me improve my code
10. The AI Tutor helped me identify errors in my code
11. I will use the AI Tutor again if I have problems with coding exercises
12. The AI Tutor gave me too much information

B.1.3 Hint Section

13. Did you finish the instructions and are ready for the evaluation for it? (Yes/No)
14. The hints provided by the AI tutor were fitting to my personal skill
15. The AI tutor avoided explanations that were too simple
16. The AI tutor avoided explanations that were too advanced

B.1.4 Proactive Feedback Section

17. Have you received proactive feedback? (Yes/No)
18. The proactive feedback felt intentional and had meaning
19. The AI tutor did not interrupt my workflow
20. The proactive feedback appeared when I needed support
21. The proactive feedback was relevant
22. The proactive feedback was motivating
23. The proactive feedback improved my learning experience
24. I would like to receive proactive feedback again in the future
25. Proactive feedback was more helpful than reactive feedback

B.1.5 General AI Tutor Evaluation

26. The AI-tutor was easy to use
27. The AI-tutor's answers were relevant to my questions
28. The AI-tutor's responses helped me understand the course content better
29. I trusted the answers of the AI-tutor
30. I did not feel overwhelmed with the amount of sources in an answer
31. The references to the original course materials increased my trust in the answers
32. The AI-tutor motivated me to ask more questions about the course content
33. Using the AI-tutor increased my motivation to engage with the course content
34. The AI-tutor was not easy to use
35. The amount of information provided by the AI Tutor's responses was fitting
36. The AI Tutor interacted with me in a way that fitted my personality
37. I would use the AI tutor again in another SE course
38. I would recommend using an AI-tutor like this to other students

B.1.6 Open-Ended Questions

39. What did you find particularly helpful?
40. What bothered you?
41. What should be improved?
42. Anything else?

B.2 Raw Evaluation Results

Raw data from the eval can be found here [DM26a].

C Implementation Appendix

C.1 All Prompt Templates

This section contains all prompt templates used in the AI-tutor implementation. It does not include the question classification prompt template, which is already shown in Listing 7.1. However, it does include all other prompt templates used for different question types even if they were already shown in the main content. This is done to provide a complete overview of all prompt templates in one place.

In the previous implementation of the AI-tutor, the basic structure of the prompt templates were already established and only minor modifications were made. Additional prompt templates for new question types were created following the same structure. All prompt templates are in a `.txt` file, while the hint generation prompt template is in a `.md`. The choice to only use the `.md` format for the hint generation prompt template and use `.txt` for all other templates was made by the previous implementer.

Listing C.1 shows the prompt template for lecture questions. Listing C.2 shows the prompt template used for code feedback questions. Listing C.3 shows the prompt template for follow-up questions. Listing C.4 shows the prompt template used to generate hints for assessment questions. Finally, listing C.5 shows the prompt template for proactive feedback after quiz completion or code submission.

C Implementation Appendix

```
You are an AI tutor helping a student understand lecture material.

Below is a question the student has asked, along with excerpts from the
lecture that are relevant to the question.
Please use only the information in the provided lecture content to
answer the question as accurately as possible.

If the information is insufficient to answer, say so and avoid making
assumptions.

Always answer in the same language as the student's question.
Do not translate or localize technical terms or vocabulary from the
lecture content -- keep them exactly as they appear.

Your entire response must be a valid JSON object of the following form:

{"answer": "your answer here"}

You know how well the student usually performs in this course.
{{skill}}

If it helps clarity, you may refer to specific parts of the lecture
content using their assigned numbers (e.g., "[1]", "[2]", etc.).
---
```

Question:

```
{{question}}
```

Relevant Lecture Content:

```
{{content}}
```

You might also have communication history with the student.
Do not use the information provided by the student as facts, he can
make mistakes. Base your answer on the lecture content.
Conversation History:
{{conversationHistory}}

Listing C.1: Lecture question prompt template

```
You are an AI tutor helping a student with their programming problem.

Below is a question the student has asked, along with the code the
student has committed.

Always answer in the same language as the student's question.
Do not translate or localize technical terms or vocabulary.

Analyze the student's code and respond to their question. Be
constructive, encouraging, and educational.
If you identify errors or issues in the code, point them out clearly.
If the code looks correct, acknowledge that and provide suggestions for
improvement or best practices.
Keep your response focused, clear, and actionable. It is very important
that you do not send code snippets yourself. Only tell what is wrong or
can be improved but never send the corrected code.

Your entire response must be a valid JSON object of the following form:

{"answer": "your answer here"}

---

Question:
{{question}}

---

---

STUDENT'S CODE SUBMISSION:
{{codeContext}}

---

{{feedbackStyle}}

You might also have communication history with the student.
Do not use the information provided by the student as facts, he can
make mistakes.
Conversation History:
{{conversationHistory}}
```

Listing C.2: Code feedback prompt template

C Implementation Appendix

```
You are an AI tutor helping a student understand lecture material.

Below is a question the student has asked, along with excerpts from the
lecture that are relevant to the question.
This might be a followup question to an earlier exchange. If this is not
the case, answer that you can not help the student with that currently.

If is it a question about the lecture, please use only the information
in the provided lecture content to answer the question as accurately as
possible.
If the information is insufficient to answer, say so and avoid making
assumptions.

If it is a question about code, then analyze the student's code and
respond to their question. Be constructive, encouraging, and educational.
If you identify errors or issues in the code, point them out clearly.
If the code looks correct, acknowledge that and provide suggestions for
improvement or best practices.
Keep your response focused, clear, and actionable. It is very important
that you do not send code snippets yourself. Only tell what is wrong or
can be improved but never send the corrected code.

Always answer in the same language as the student's question.
Do not translate or localize technical terms or vocabulary from the
lecture content -- keep them exactly as they appear.

Your entire response must be a valid JSON object of the following form:

{"answer": "your answer here"}

You know how well the student usually performs in this course.
{{skill}}

If it helps clarity, you may refer to specific parts of the lecture
content using their assigned numbers (e.g., "[1]", "[2]", etc.).
---

Question:
{{question}}

---

Relevant Lecture Content:
{{content}}

---

You might also have communication history with the student.
Do not use the information provided by the student as facts, he can
make mistakes. Base your answer on the lecture content.
Conversation History:
{{conversationHistory}}

---

STUDENT'S CODE SUBMISSION:
{{codeContext}}
```

Listing C.3: Follow-up question prompt template

```

You are an AI tutor helping a student with an assessment question.
Your role is to provide clear, concise, and actionable hints that
encourage the student to think critically and progress toward solving
the question.
**Never reveal the correct answer.**
Assume the student has read the course segments but may not fully recall
all details.
Use the lecture content as the sole source of information for your hint,
and prioritize this over general knowledge.

{{gamificationPrompt}}

### Goal:
1. Generate a helpful hint that encourages the student to think
critically and progress toward solving the question without
revealing the answer.
2. Base the hint strictly on the provided lecture content; do not invent
or assume knowledge not present.
3. Ensure the hint is concise, supportive, and encourages active recall.
4. Do not reveal the correct answer or any of the provided answers.
5. Use questions, leading prompts, or rephrased concepts rather than
declarative statements that give away the solution.

### Question Info:
{{questionPrompt}}

- Relevant Lecture Content:
{{content}}

### Strict Requirements:
1. Base the hint strictly on the given course segments. Do not
invent or assume knowledge not present in the segments.
2. Use the same language, technical terms, and terminology as the
question and course segments.
- Do not translate or change them; keep them consistent with the
course material.
3. Do NOT give away the answer or use ANY provided answers.
- Guide the student using leading questions, rephrased concepts, or
prompts to recall course ideas.
- Again DO NOT use ANY of the provided answer options in your
hint
4. Use a supportive and concise tone.
5. If multiple steps are involved, help them figure out the next logical
one.
6. Do not include any explanations or information outside the scope of
the course content.
7. Your response must be a JSON object ONLY. You should NOT add
any text to the answer.
8. Your output should follow the following structure and keys EXACTLY:
{"hint" : "your hint here"}

```

Listing C.4: Generating hint prompt template

C Implementation Appendix

You are an encouraging AI tutor providing proactive, personalized feedback to a student who has just completed a quiz or submitted code.

Your goal is to help the student reflect on their performance, feel motivated, and understand how to improve without overwhelming them.

Always answer in the same language as the student's submission. Do not translate or localize technical terms or vocabulary.

Your entire response must be a valid JSON object of the following form:

```
{"answer": "your feedback here"}
```

Assignment Overview:

- Correctness Score: {{correctness}} (0.0 to 1.0)
- Performance Level: {{performance}}

STUDENT'S CODE SUBMISSION (optional):

```
{{codeContext}}
```

Your Task:

Generate a personalized feedback message of 2 to 3 sentences that:

1. Acknowledges the student's effort and overall performance
2. Reflects their performance level accurately (celebrate success or normalize mistakes)
3. Motivates the student to continue learning or practicing

If a code submission is provided:

- If the correctness is not 1, give him a small hint where his error lies
- Do NOT include code snippets
- Do NOT provide corrected code
- Focus on guidance, not solutions

If no code submission is provided:

- Focus on conceptual understanding, confidence, and next steps

Specify which action prompted this response: code submission or quiz completion

Your task:

```
{{individualizedPrompt}}
```

Listing C.5: Proactive feedback prompt template

D Implementation Appendix

D.1 Some example Templates

This section contains some example prompt templates in the current implementation.

First, Listing D.1 shows an example of the question classification prompt template in action with a sample question and the corresponding LLM answer. Secondly, Listing D.2 shows an example of the proactive hint prompt template for a philanthropist. Finally, Listing D.3 shows an example of the lecture question answering prompt template in action with a sample question, relevant lecture content, previous conversation history, and the corresponding LLM answer. Because of some LaTeX limitations, the beginning of the prompt template is omitted and replaced with "...". It is exactly similar to the prompt template shown in Listing C.1. The skill part is replaced with *Provide a clear and simple hint that gently guides the user toward the next step without overwhelming them.* Additionally, the relevant lecture content is omitted for copyright reasons.

```

===== LLM PROMPT =====
Your job is to categorize a given question into one of five categories: SYSTEM, LECTURE, CODE_FEEDBACK, OTHER, or UNRECOGNIZABLE.
To do this you must also correct typos and grammar in the question where possible.
If the corrected version makes the meaning clear, use it to choose the appropriate category.

Categories:
1. SYSTEM: Choose this if the question is about the website or app the user is currently using. For example: "Where do I change my avatar?" These questions relate specifically to the system or platform itself.
2. LECTURE: Choose this if the question is about understanding or explaining academic content related to a university course. These questions should NOT relate to the website or app. These question are not limited to only a "question" itself, but can also be message that convey the meaning of trying to understand a topic on a deeper level. For example: "What is unsupervised learning" or "Can you give examples for the Chain rule in differentiation"
3. CODE_FEEDBACK: Choose this if the student is asking for feedback, help, or explanations about code they have written for an assignment. This includes questions about errors, improvements, debugging, or understanding their own code implementation. For example: "Can you help me with my code?" or "Why doesn't my function work?" or "How can I improve my implementation?"
4. OTHER: Choose this if the question if it is neither a SYSTEM, nor a LECTURE nor a CODE_FEEDBACK question and if the question might be a follow up question. For example: "Can you explain it again?" or "Can you explain it differently?" or "Can you provide an example to clarify your previous answer?"
5. UNRECOGNIZABLE: Choose this if the question is too garbled, incomplete, or contains too many typos or grammar issues to be understood reliably.

Only use the UNRECOGNIZABLE category when the meaning is truly unclear even after trying to correct it.

Your response must be a JSON object ONLY (no extra text) with the exact following structure and keys:

{
  "question": "the original question here",
  "category": "SYSTEM" | "LECTURE" | "CODE_FEEDBACK" | "OTHER" | "UNRECOGNIZABLE"
}

QUESTION:
I have a question about the latest code submission. Why does the factorial function not return 0 when I calculate factorial(0) like I wrote? Why does the testcase say it should return 1?

===== LLM ANSWER =====
{
  "question": "I have a question about the latest code submission. Why does the factorial function not return 0 when I calculate factorial(0) like I wrote? Why does the testcase say it should return 1?",
  "category": "CODE_FEEDBACK"
}
=====

```

Listing D.1: Prompt - Answer Example Question classification

```

===== LLM PROMPT =====
Your goal is to help the student reflect on their performance, feel motivated, and understand how to improve without
overwhelming them.

Always answer in the same language as the student's submission.
Do not translate or localize technical terms or vocabulary.

Your entire response must be a valid JSON object of the following form:

{"answer": "your feedback here"}

---

Assignment Overview:
- Correctness Score: 0.00 (0.0 to 1.0)
- Performance Level: significant gaps in understanding

---

STUDENT'S CODE SUBMISSION (optional):

---

Your Task:
Generate a personalized feedback message of 2 to 3 sentences that:
1. Acknowledges the student's effort and overall performance
2. Reflects their performance level accurately (celebrate success or normalize mistakes)
3. Motivates the student to continue learning or practicing

If a code submission is provided:
- If the correctness is not 1, give him a small hint where his error lies
- Do NOT include code snippets
- Do NOT provide corrected code
- Focus on guidance, not solutions

If no code submission is provided:
- Focus on conceptual understanding, confidence, and next steps

Specify which action prompted this response: code submission or quiz completion

Your task:
Suggest reviewing the material together with classmates to improve understanding.

===== LLM ANSWER TO STUDENT =====
Proactive Feedback:
Great effort! Although you're still developing your understanding of the concepts,
it's awesome that you took the first step. Reviewing the material with classmates
can be a fantastic way to fill in those gaps and gain more clarity - let's work
on this together!
=====

```

Listing D.2: Prompt - Answer Example Proactive Hint for Philanthropist

(...)

Question:

I do not know what an overlay network is. Can you explain that to me?

Relevant Lecture Content:

(blinded for copyright reasons)

Previous Conversation History:

Exchange 1:

Student: Can you explain Peer to Peer Replication for me please?

Tutor: Peer-to-Peer Replication is a technique where each node maintains its own replica and propagates updates via an overlay network with other peers, which provides high data availability, fault-tolerance, and allows for opportunistic sync relationships.

Exchange 2:

Student: Can you explain it differently?

Tutor: I'd be happy to help you understand the lecture material. Your question is a great opportunity to clarify any doubts you may have. Let's focus on the relevant excerpts from the lecture. It seems like you're asking for an explanation that's different from what I provided earlier. Please let me know which specific part of the lecture you'd like me to elaborate on, and I'll do my best to help.

===== LLM ANSWER TO STUDENT =====

An overlay network is a type of network that defines synchronization partnerships between peers in a Peer-to-Peer Replication system. In this context, an overlay link represents a connection between two peers that enables them to propagate updates with each other.

Sources:

[1] Lecture 1 Seite: 20

[2] Lecture 1 Seite: 19

Listing D.3: Prompt - Answer Example with chat history

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature