

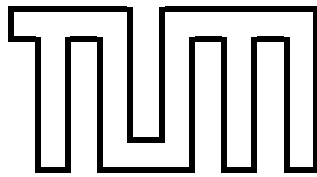
**Technical University of Munich**

School of Computation, Information and Technology  
– Informatics –

Master's Thesis in Informatics

**Leveraging LLMs for Automated  
Feedback Generation on Exercises**

Felix Timotheus Johannes Dietrich



**Technical University of Munich**

School of Computation, Information and Technology  
– Informatics –

Master's Thesis in Informatics

**Leveraging LLMs for Automated  
Feedback Generation on Exercises**

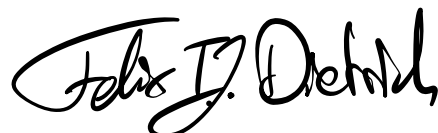
Nutzung von LLMs für die automatische  
Generierung von Feedback zu Übungen

Author: Felix Timotheus Johannes Dietrich  
Supervisor: Prof. Dr. Stephan Krusche  
Advisor: Maximilian Sölch, M.Sc.  
Submission Date: 15. September 2023



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15. September 2023

A handwritten signature in black ink, reading "Felix J. Dietrich". The signature is written in a cursive style with a large, sweeping initial "F".

Felix Timotheus Johannes Dietrich

## Abstract

Leveraging the capabilities of Large Language Models (LLMs) promises a practical solution to the real-world challenge of giving personalized feedback in text and programming exercises within large educational settings. This research utilizes LLMs within Athena, a key part of the Artemis Learning Management System (LMS). The objective is to improve the quality and adaptability of automated feedback while directly supporting tutors in their assessment tasks.

The core aim of this thesis is to enhance Artemis’s existing semi-automated assessment system for text exercises, currently facilitated by CoFee [Ber22], by introducing LLMs into the process. Concurrently, this work introduces LLM-based automated feedback for programming exercises. Both initiatives are supported within a newly established research and development environment, designed to streamline the future creation of automated assessment approaches for Artemis within Athena.

The methodology employs a two-stage approach: first, LLMs are utilized to generate automated feedback for text and programming exercises, followed by a comprehensive evaluation focusing on accuracy, cost, and generation times. Initial results affirm that this integrated approach is promising, especially for text exercises, aligning well with educational objectives and offering substantial potential for future advances.

## Zusammenfassung

Die Nutzung von Large Language Models (LLMs) bietet eine praktische Lösung für die reale Herausforderung, personalisiertes Feedback in Text- und Programmierübungen in großen Bildungsumgebungen zu geben. Diese Forschung integriert LLMs in Athena, einem Schlüsselement des Artemis Learning Management Systems (LMS). Das Ziel ist, die Qualität und Anpassungsfähigkeit des automatisierten Feedbacks zu verbessern und gleichzeitig Tutoren bei ihren Bewertungsaufgaben direkt zu unterstützen.

Der zentrale Fokus dieser Arbeit liegt darauf, das bestehende semi-automatisierte Bewertungssystem von Artemis für Textaufgaben, derzeit durch CoFee [Ber22] erleichtert, durch die Einführung von LLMs zu verbessern. Parallel dazu führt diese Arbeit LLM-basiertes automatisiertes Feedback für Programmieraufgaben ein. Beide Initiativen werden in einer neu geschaffenen Forschungs- und Entwicklungsumgebung unterstützt, die darauf abzielt, die künftige Erstellung automatisierter Bewertungsansätze für Artemis in Athena zu vereinfachen.

Die Methodik folgt einem zweistufigen Ansatz: Erstens werden LLMs eingesetzt, um automatisiertes Feedback für Text- und Programmieraufgaben zu generieren; zweitens erfolgt eine umfassende Evaluierung, die sich auf Genauigkeit, Kosten und Generierungszeiten konzentriert. Erste Ergebnisse bestätigen, dass dieser integrierte Ansatz vielversprechend ist, insbesondere für Textübungen. Es passt gut zu den pädagogischen Zielen und bietet erhebliches Potenzial für zukünftige Fortschritte.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem . . . . .	2
1.2	Motivation . . . . .	3
1.3	Objectives . . . . .	4
1.4	Outline . . . . .	5
<b>2</b>	<b>Large Language Models</b>	<b>6</b>
2.1	What are Large Language Models? . . . . .	6
2.2	Evolution of Current LLMs . . . . .	6
2.3	Real-World Application of LLMs . . . . .	7
2.3.1	(Chat-)Completion Interface . . . . .	8
2.3.2	Fine-Tuning LLMs . . . . .	8
2.3.3	Basic Workflow: Format, Predict, Parse . . . . .	9
<b>3</b>	<b>Related Work</b>	<b>10</b>
<b>4</b>	<b>Requirements Analysis</b>	<b>12</b>
4.1	Overview . . . . .	12
4.2	Current System . . . . .	13
4.2.1	Assessment of Text Exercises . . . . .	13
4.2.2	Assessment of Programming Exercises . . . . .	15
4.3	Proposed System . . . . .	16
4.3.1	Functional Requirements . . . . .	16
4.3.2	Non-Functional Requirements . . . . .	19
4.4	System Models . . . . .	21
4.4.1	Scenarios . . . . .	21
4.4.2	Use Case Model . . . . .	23
4.4.3	Analysis Object Model . . . . .	26
4.4.4	Dynamic Model . . . . .	28
4.4.5	User Interface . . . . .	30

<b>5</b>	<b>System Design</b>	<b>32</b>
5.1	Overview . . . . .	32
5.2	Design Goals . . . . .	33
5.3	Subsystem Decomposition . . . . .	34
5.3.1	Athena Assessment Service . . . . .	35
5.3.2	Text/Programming LLM Module . . . . .	36
5.3.3	Research and Development Playground . . . . .	37
5.3.4	Artemis . . . . .	38
5.4	Hardware Software Mapping . . . . .	39
<b>6</b>	<b>Object Design</b>	<b>41</b>
6.1	Athena Assessment Service . . . . .	41
6.2	Athena: Text LLM Module . . . . .	42
6.2.1	Generating Feedback Suggestions . . . . .	42
6.2.2	Generating Evaluations . . . . .	44
6.3	Athena: Programming LLM Module . . . . .	46
6.4	Research and Development Environment . . . . .	51
6.4.1	Module Configuration . . . . .	51
6.4.2	Module Requests . . . . .	51
6.4.3	Evaluation Management . . . . .	51
<b>7</b>	<b>Evaluation</b>	<b>58</b>
7.1	Design . . . . .	58
7.1.1	Data Selection . . . . .	59
7.1.2	Experiments . . . . .	61
7.2	Objectives . . . . .	63
7.2.1	Text Exercises . . . . .	64
7.2.2	Programming Exercises . . . . .	64
7.3	Results . . . . .	64
7.3.1	Text Exercises . . . . .	65
7.3.2	Programming Exercises . . . . .	66
7.4	Findings . . . . .	67
7.4.1	Text Exercises . . . . .	67
7.4.2	Programming Exercises . . . . .	68
7.5	Discussion . . . . .	68
7.5.1	Text Exercises . . . . .	69
7.5.2	Programming Exercises . . . . .	69
7.5.3	Implications for Artemis . . . . .	70
7.6	Limitations . . . . .	70



<b>8</b>	<b>Summary</b>	<b>72</b>
8.1	Status . . . . .	72
8.1.1	Realized Goals . . . . .	72
8.1.2	Open Goals . . . . .	74
8.2	Conclusion . . . . .	75
8.3	Future Work . . . . .	76
8.3.1	Fine-Tuning Large Language Models . . . . .	76
8.3.2	Agentic Approach for Programming Exercises . . . . .	77
8.3.3	Modeling Exercises . . . . .	77
<b>A</b>	<b>Data Exploration</b>	<b>79</b>
A.1	Data Acquisition and Selection . . . . .	79
A.2	Exploratory Data Analysis (EDA) . . . . .	79
A.2.1	Overview . . . . .	81
A.2.2	Text Exercises . . . . .	82
A.2.3	Programming Exercises . . . . .	87



# Chapter 1

## Introduction

Can technology revolutionize the way we educate by doing what humans can't scale? This question has gained critical importance as student enrollments increase, especially in technical disciplines. The Technical University of Munich (TUM) is a prime example, recording an increase of 8 000 students in the last five years, reaching a total of 48 296 students for the academic year 2021 [Mü22]. This surge includes 2 468 first-year students in computer science alone, emphasizing the scalability challenge in educational settings.

Feedback is essential in educational settings for helping students get better at their subjects, especially in technical areas such as engineering or computer science, where deep understanding is crucial [NMD06]. Teachers have the important responsibility of providing targeted feedback to each student [WHZ<sup>+</sup>15]. In a small class, it is easier for a teacher to know which students need more help and give them specific advice. However, when a class has many students or students that are in different places, it becomes much more challenging for the teacher to give each student the personal feedback they need for effective learning [Kru21].

Concurrently, education technology is undergoing a transformative shift, driven by the widespread use of digital tools, which both test old ways of teaching and open new paths for student-focused learning [CH10]. Learning Management Systems (LMSs) emerge as a pivotal solution in this landscape, addressing the challenge of scalability without compromising the quality of education [TCL21]. Their adaptive frameworks and automated features support instructors in effectively engaging large numbers of students.

At the Technical University of Munich, staff and students have created the LMS *Artemis*<sup>1</sup>, a platform for interactive learning that offers individualized feedback [KS18]. Artemis aims to enhance student engagement in large

---

<sup>1</sup><https://github.com/l1intum/Artemis>

courses through technology-supported, personalized, and interactive teaching and learning, focusing on reducing grading effort and improving feedback quality [Kru21].

Although Artemis employs semi-automatic approaches for text and modeling exercises [BB, Kru22], the platform’s capability for programming exercises is mainly limited to automatic testing, static code analysis, and manual feedback. At the same time, natural language processing (NLP) technologies, notably Large Language Models (LLMs) such as GPT-3 [BMR<sup>+</sup>20], GPT-3.5-Turbo [Tea22], and GPT-4 [Ope23], offer a promising path for scalable semi-automating feedback in an educational setting [KSK<sup>+</sup>23].

Returning to the initial question — can technology scale education in ways humans can’t? This thesis explores this through the lens of Artemis, aiming to automate feedback with LLMs. The goal is to affirm that technology when adequately integrated into systems such as Artemis, can indeed make high-quality, scalable education a reality.

## 1.1 Problem

Feedback is essential for students to understand how well they are doing and what they need to improve [HT07]. However, in the context of large classes, such as those on the Technical University of Munich’s Artemis platform, delivering individualized feedback presents significant challenges for instructors [Kru21].

For instance, from the summer of 2019 to the winter of 2022/23, Artemis recorded an average semester participation of 2 018 students for text exercises and 3 288 students for programming exercises, respectively, as indicated in Figure A.2. With each tutor responsible for assessing an average of 34 students for text exercises and 31 for programming exercises, excluding courses that solely rely on automation, the workload for tutors becomes substantial, as shown in Figure A.3.

In programming exercises, automated tests and static code analysis are heavily used on Artemis to provide feedback, as depicted in Figure A.10. While automated tests ensure functional correctness and static code analysis can partially assess code readability and structure, these tools have their limits. For example, they can’t determine if a student has used the prescribed algorithm or manage issues related to non-executable code. Thus, additional manual feedback from tutors is essential to address these specific aspects.

For text exercises, Artemis currently uses a semi-automatic approach us-

ing *CoFee*<sup>2</sup> [Ber22] within the framework called *Athena*<sup>3</sup>. It is helpful but has limits. It mainly recycles old feedback for similar text answers. If a student's answer is unique, the tool isn't much help. On top of this, its usage has been declining recently, as seen in Figure A.6.

Manual feedback, although valuable, imposes a significant burden on tutors [Ala05]. The process is time-consuming and can compromise the consistency and objectivity of feedback. At scale, this method becomes less practical and poses risks of tutor burnout, affecting overall teaching quality. This can take away valuable time from other important tasks, such as lesson preparation or grading other assignments.

Students also face repercussions from this system. Inadequate or delayed feedback can impair their understanding and skill development. The absence of personalized feedback can lead to frustration and disengagement, potentially causing students to discontinue or even abandon courses.

In summary, the current feedback mechanisms for large courses on the Artemis platform have limitations that affect both tutors and students. Improved, scalable solutions for both text and programming exercises are necessary to offer more individualized and efficient feedback, thereby enhancing the educational experience for all stakeholders.

## 1.2 Motivation

The challenges associated with providing high-quality, personalized feedback in large-scale educational settings underscore the need for innovative solutions. The importance of individualized feedback in enhancing student learning outcomes and overall educational quality cannot be overstated [KR16]. Therefore, addressing the limitations of the current feedback mechanisms in the Artemis platform is not only necessary but also timely.

The potential benefits of solving this problem are manifold. Firstly, it would significantly improve the learning experience for students. By receiving timely, relevant, and personalized feedback, students can better understand their strengths and areas for improvement, leading to enhanced learning outcomes. This could potentially increase student engagement and success rates in computer science courses at the Technical University of Munich and beyond.

Secondly, it would alleviate the workload of tutors. By automating part of the feedback process, tutors can save time and effort, allowing them to focus on other crucial aspects of teaching, such as lesson preparation, student

---

<sup>2</sup><https://github.com/l1intum/Athena-CoFee>

<sup>3</sup><https://github.com/l1intum/Athena>

engagement, and research. This could lead to improved course quality and increased student satisfaction.

Thirdly, it would contribute to the broader field of education technology. By leveraging recent advances in natural language processing, particularly large language models (LLMs), this thesis aims to demonstrate the potential of these technologies in enhancing the scalability and efficiency of feedback mechanisms in learning management systems. This could pave the way for further research and development in this area, potentially transforming the way feedback is provided in large-scale educational settings.

The work of Bernius *et al.* [BB, BKB22] on semi-automatically grading text exercises within the Artemis platform provides a promising starting point. Their approach has already shown that semi-automatic feedback can reduce grading overhead by up to 85%. This thesis aims to build on this success, exploring the potential of LLMs in automating feedback for both text and programming exercises. By doing so, it seeks to make a significant contribution to the future of scalable, high-quality education.

### 1.3 Objectives

The primary aim of this thesis is to enhance the automated feedback mechanisms within Artemis by leveraging large language models (LLMs). The focus is on addressing the challenges associated with providing individualized and meaningful feedback in the context of text and programming exercises. The objectives are as follows:

**Feedback Generation for Text Exercises.** The first objective is to explore the feasibility of an LLM-driven feedback generation system for text exercises within Artemis. The goal is to develop a system that can generate feedback suggestions, which can assist tutors in the assessment process. The effectiveness of this system will be evaluated on actual exercises, using metrics such as feedback accuracy, token usage, and generation speed.

**Feedback Generation for Programming Exercises.** The second objective is to investigate the potential of an LLM-based feedback system for programming exercises. Given the complexity of coding tasks, this system will aim to generate feedback suggestions that address unique aspects of programming. The system's effectiveness will be evaluated similarly to the previous objective.

**Research and Development Environment.** The final objective is to

establish a research and development environment for Athena. This environment will serve as the foundation for designing feedback processes, comparing alternative LLM-driven approaches or CoFee, and rigorously evaluating their effectiveness within Athena’s framework. The environment will be designed to facilitate quick iterations, thereby serving as an evolving foundation for the continuous improvement of the feedback mechanisms.

In summary, this thesis aims to significantly improve the Artemis platform’s feedback capabilities through the targeted application of LLM technology. By developing specialized feedback system baselines for text and programming exercises and constructing a systematic evaluation environment for research and development, this work seeks to enrich the educational experience for students and reduce the manual workload for tutors.

## 1.4 Outline

The thesis is organized as follows. Chapter 2, Large Language Models, introduces the necessary background for understanding LLMs. In Chapter 3, Related Work, we survey current automated feedback systems. Chapter 4, Requirements Analysis, clarifies the capabilities of our existing system and outlines the goals and requirements for the proposed system. System Design is covered in Chapter 5, which explains our chosen architecture. Chapter 6, Object Design, elaborates on the implementation of the system. Evaluation methods and results are presented in Chapter 7. The thesis concludes with Chapter 8, Summary, which recaps our achievements and suggests directions for future research. Additionally, Data Exploration in Appendix A provides an in-depth analysis of the existing feedback landscape within Artemis to inform data-driven development.

# Chapter 2

## Large Language Models

This chapter lays the groundwork for understanding large language models. We start with a brief overview, Section 2.1, move on to the evolution of LLMs in Section 2.2, and finally discuss their application in real-world settings, in Section 2.3. This framework equips us for the subsequent, more focused examination of leveraging LLMs in Artemis for automated feedback generation.

### 2.1 What are Large Language Models?

Language models are probabilistic models of natural language that assign probabilities to sequences of words, offering a measure of how likely a given sentence is to occur [JM23]. This predictive nature enables them to generate coherent and contextually appropriate text. Large Language Models (LLMs), a subset of language models, distinguish themselves by being trained on vast datasets, comprising billions of tokens. This extensive training enables LLMs to handle a multitude of tasks, from text generation to code writing, without requiring explicit instructions.

### 2.2 Evolution of Current LLMs

In 2017, Vaswani *et al.* at Google introduced the Transformer model in “Attention Is All You Need”, which relies solely on attention mechanisms. This breakthrough served as the foundation for subsequent LLMs by achieving unmatched results in machine translation tasks [VSP<sup>+</sup>23]. OpenAI presented the first generation of the Generative Pre-trained Transformer (GPT), emphasizing pre-training on diverse, unlabeled text to improve performance across a range of tasks [RNSS]. Devlin *et al.* further expanded the field



with BERT, which demonstrated state-of-the-art results in natural language understanding by using deep bidirectional representations [DCLT19].

As the models evolved, OpenAI released GPT-2, highlighting the ability of LLMs to learn tasks without explicit supervision [RWC<sup>+</sup>]. Meanwhile, Google launched T5, a unified framework for text-based language problems [RSR<sup>+</sup>20]. GPT-3 took scalability to new heights with 175 billion parameters, showcasing exceptional few-shot learning abilities [BMR<sup>+</sup>20].

Further specialization occurred with models such as Codex, fine-tuned for code writing [CTJ<sup>+</sup>21], and LaMDA, for dialog applications [TDFH<sup>+</sup>22]. Research also turned toward alignment with human intent, as seen in Instruct-GPT [OWJ<sup>+</sup>22] together with its evolution into GPT-3.5-Turbo [Tea22]. Lastly, the recent introduction of GPT-4 suggests the possibility of multimodal inputs, demonstrating advancements not just in scale but also in scope [Ope23].

In contrast to proprietary models from OpenAI and other providers, BLOOM and the LLaMA series stand out as open-access LLMs. BLOOM, developed through collaborative research, presents a substantial 176-billion parameter model trained on a rich multilingual corpus [WSF<sup>+</sup>23]. However, the LLaMA series, particularly LLaMA 2, emerges as exceptionally noteworthy. Ranging from 7 to 70 billion model parameters, LLaMA 2 has demonstrated superior performance in benchmarks, even outpacing GPT-3 in most metrics. Critically, the open-access nature of LLaMA 2 allows for fine-tuning and execution on independent infrastructure, empowering the broader research community [TMS].

The rapid evolution of LLMs offers a promising landscape for educational technology, especially in automating feedback. Advancements in both closed-source and open-source models pave the way for automation needed in platforms such as Artemis. This ongoing growth aligns well with the objectives of this thesis and directly leads us to the next section, where we discuss how to leverage LLMs in real-world applications.

## 2.3 Real-World Application of LLMs

Despite their complexity, LLMs have been readily integrated into various real-world applications. While computational costs and infrastructure requirements pose challenges, services such as Replicate<sup>1</sup> and HuggingFace<sup>2</sup> have mitigated these issues by offering cloud-based solutions. These services

---

<sup>1</sup><https://replicate.com>

<sup>2</sup><https://huggingface.co>

provide streamlined access to open-source models and handle the intricacies of scaling and execution. Closed-source models from providers such as OpenAI<sup>3</sup> and Anthropic<sup>4</sup> offer APIs to integrate LLMs into applications effortlessly without revealing their model weights.

### 2.3.1 (Chat-)Completion Interface

The completion interface, for models such as GPT-3 [BMR<sup>+</sup>20], has been the most common interaction mechanism with LLMs. Provided with a *prompt*, the input text, the model generates a sequence of *tokens* as output until a stop token is reached, this sequence is called the *completion*. A token is a single unit of text, usually a word or punctuation mark.

A newer more structured approach takes the chat completion interface, as used by GPT-3.5-Turbo [Tea22] or GPT-4 [Ope23]. This interface is tailored for dialogue-oriented applications, in the form of messages from different roles, and allows for more interactive and conversational engagement with the model. Usually, only one *system message* is provided for *guidance*, followed by a history of *user* and *assistant messages*. Upon calling the chat completion interface with this input, the model generates a completion for the next *assistant message*. The roles of the messages might vary depending on the application, but the most common ones are *user* and *assistant*. This allows for a more instructive and contextual dialogue between the user and the system.

### 2.3.2 Fine-Tuning LLMs

*Fine-tuning* is an indispensable tool in the deployment of LLMs for specialized applications. During this process, an already trained general-purpose model such as LLaMA 2, GPT-3, or GPT-3.5-Turbo undergoes additional training on a dataset that closely represents the target application's context. In essence, fine-tuning tailors the model to understand the domain-specific nuances and expectations, thereby increasing its performance in specialized tasks. Open-source models such as LLaMA 2 aim to simplify this process, offering straightforward guidelines for fine-tuning [TMS]. Furthermore, commercial platforms such as OpenAI provide fine-tuning capabilities through their API<sup>5</sup>, making it feasible to adapt models such as GPT-3, GPT-3.5-Turbo, and the soon GPT-4 for specific applications.

---

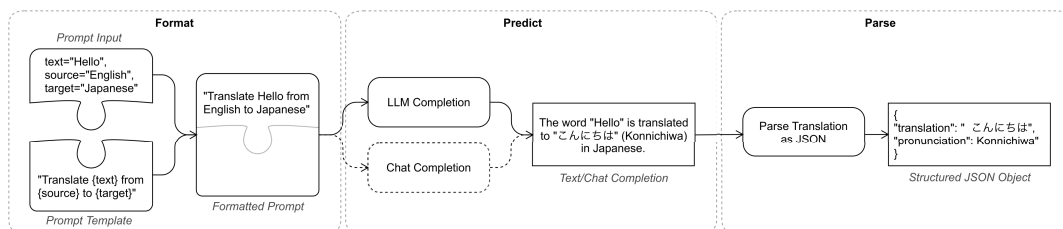
<sup>3</sup><https://openai.com>

<sup>4</sup><https://www.anthropic.com>

<sup>5</sup><https://platform.openai.com/docs/guides/fine-tuning>

### 2.3.3 Basic Workflow: Format, Predict, Parse

The typical workflow for integrating an LLM into an application can be summarized in three steps, as illustrated in Figure 2.1: **Format**, **Predict**, and **Parse**. First, we construct a prompt that serves as the model’s input, often generated from a prompt template with placeholders for prompt input. Second, we execute the model to produce a sequence of tokens as output. Finally, depending on the application, we parse the completion output into a structured format such as JSON for further processing.



**Figure 2.1:** Basic Workflow using Large Language Models.

The ability for LLMs to consistently generate structured output such as JSON is not a given. It largely depends on how powerful the model is and how it was trained. For example, OpenAI’s chat models, GPT-3.5-Turbo [Tea22] and GPT-4 [Ope23], support *function calling*<sup>6</sup>, which allows the LLM to return a structured output directly. This is achieved by fine-tuning the model on JSON data, enabling it to understand the structure of JSON and produce a structured output consistently. This is a significant advantage for applications that require structured output, such as feedback generation within Artemis, as it eliminates the need for additional parsing steps.

<sup>6</sup><https://openai.com/blog/function-calling-and-other-api-updates>

# Chapter 3

## Related Work

This chapter examines the existing research on automated feedback systems in educational contexts, focusing on text and programming exercises. The goal is to position this thesis within the broader academic landscape, identifying its unique contributions.

Basu *et al.* developed *Powergrading*, a machine learning technique combined with human expertise to cluster similar student answers for automated grading of short answers [BJV13]. While this approach streamlines the grading process, it is not directly applicable to programming exercises and does not utilize latest advancements of large language models (LLMs).

Singh *et al.* introduced *Gradescope*, an online platform for grading handwritten assignments and exams using dynamically evolving rubrics [SKGA17]. *CoFee*, developed by Bernius *et al.*, uses machine learning to automatically provide feedback on open-ended text exercises [BKB22]. It decreases the workload for instructors but relies on reusing previous feedback for similar answers. Schwind later extended this concept with *ThemisML* for programming exercises [Sch23].

In Automated Essay Scoring (AES), Dong *et al.* applied recurrent and convolutional neural networks (RNNs and CNNs) for grading [DZY17]. Rodriguez *et al.* utilized BERT and XLNet models to similar ends [RJO19]. These methods, however, focus primarily on scoring full-length essays rather than providing feedback on specific aspects of the problem statement.

For programming exercises, existing systems such as *WebCAT* [EP08] and the *Artemis* LMS [Kru21], subject of this thesis, concentrate on automated testing and static code analysis. Automatic testing and static code analysis are limited in providing feedback on non-executable code or specific aspects of the problem statement, which still require manual assessment. This is where LLMs can play a crucial role to cover these gaps.

Kasneci *et al.* conducted a comprehensive review on the opportunities of

---

LLMs in educational settings [KSK<sup>+</sup>23]. They argue for a human-centered approach, which resonates with this thesis's aim to create a semi-automated system that supports tutors in providing feedback.

In summary, existing solutions do not adequately combine the advanced features of LLMs with the specific needs of both text and programming exercises within a Learning Management System such as Artemis. This thesis addresses this gap by employing the latest advancements in LLM technology to develop a scalable, adaptable, and efficient feedback system, thereby making a significant contribution to educational technology and Artemis.

# Chapter 4

## Requirements Analysis

This chapter outlines the requirements for integrating large language models (LLMs) into Artemis to automate feedback generation. Using Bruegge and Dutoit’s RAD template [BD10], we first define the system’s purpose, objectives, scope, and success criteria. Next, we summarize relevant existing features of Artemis. We then specify the functional and non-functional requirements of our proposed system. The chapter concludes with system models, along with corresponding user interface (UI) mockups.

### 4.1 Overview

As introduced in Chapter 1, the primary purpose of this project is to enhance the assessment capabilities of Artemis for text and programming exercises. This is achieved by integrating LLMs, which, as outlined in Chapter 2, are a rapidly advancing technology with vast potential. We aim to leverage these capabilities to autonomously generate valuable, personalized feedback within Athena, the existing automatic assessment framework of Artemis.

Critical to the realization of this objective is the establishment of a research and development environment within Athena. This environment aims to be a testing ground for fine-tuning the LLM prompts, a crucial factor in the quality of generated feedback. The ability to rapidly iterate in this controlled setting is a foundational requirement for the project, enabling us to optimize prompt effectiveness and minimize error rates in the feedback generated. We will refer to this context as the *Research and Development Context*, while the LMS related context will be referred to as the *Learning Management Context*.

Addressing the project’s constraints is crucial for a focused scope. The development team consists of two people: one concentrating on integrating

LLM-enabled approaches for text and programming exercises, and the other primarily tasked with generalizing Athena beyond CoFee and its integration with Artemis. Therefore, it is vital to set realistic expectations regarding the scope of the system. Our primary focus is to deliver an integrated, well-documented, and high-quality system that establishes a solid foundation for future development and iteration. The specific design goals for the proposed system are outlined in Section 5.2.

The success of the system depends on its ability to consistently generate high-quality feedback for a diverse range of exercises, seamlessly integrate with the Artemis platform, and provide tangible benefits to the tutors.

## 4.2 Current System

Artemis (version 6.1.8), as it stands, offers a range of exercise types, including text, programming, modeling, among others. Those exercises are conducted online as part of academic assignments or exams. Artemis supports semi-automatic assessment for text and modelling exercises, while programming exercises are managed by manual assessment in addition to automatic testing and static code analysis. In this section, we discuss the features and limitations of the current system, focusing mainly on the assessment of text and programming exercises, Section 4.2.1 and Section 4.2.2, respectively.

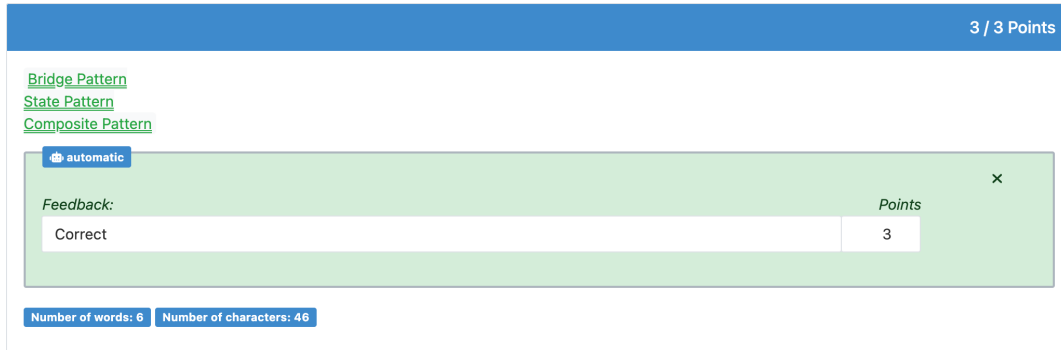
### 4.2.1 Assessment of Text Exercises

Artemis employs a manual assessment approach for text exercises, with limited automation through its Athena framework. Athena uses the CoFee approach to generate feedback suggestions [Ber22]. These suggestions are presented to the tutor when assessing a submission, ready for modification or deletion, if needed.

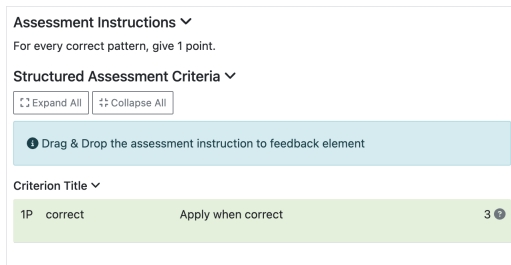
The assessment interface offers tutors two distinct options for feedback: referenced and unreferenced. Referenced feedback attaches directly to specific text blocks in a student’s submission and is displayed inline. If this type of feedback is generated by Athena, it is labeled with an “automatic” badge, as illustrated in Figure 4.1a. Conversely, unreferenced feedback, as shown in Figure 4.1c, is general and applies to the entire submission.

In terms of grading, both referenced and unreferenced feedback can assign points, ranging from negative to positive values. This scoring system operates within a defined range, established by the sum of maximum points and bonus points. To guide tutors during assessment, Artemis presents grading instructions in a sidebar, as depicted in Figure 4.1b. These grading

instructions come in two forms: unstructured text and structured grading instructions. For the latter, tutors can directly link feedback to a structured grading instruction through a drag-and-drop feature, making it easy to apply consistent graded feedback.



(a) Automated Feedback Suggestion in Text Exercises. (Screenshot)



(b) Grading Instructions in Text Exercises. (Screenshot)



(c) Unreferenced Feedback in Text Exercises. (Screenshot)

**Figure 4.1:** Text Exercise Assessment Features in Artemis.

Statistically, as of March 29, 2023, the majority of text exercises are conducted in English, accounting for 76.8% of all submissions, while German follows at 15.6%, as indicated in Figure A.4. Referenced feedback is the prevailing form, making up 86.6% of all feedback; unreferenced feedback fills the remaining 13.4%, as shown in Figure A.8a. In terms of structured grading instructions, they are increasingly being utilized; specifically, in the summer semester of 2021, 79.2% of all feedback was linked to these instructions, as depicted in Figure A.7. However, Athena’s automated feedback has yet to gain widespread use, as shown in Figure A.6.

Despite its utility, the current approach within Athena, CoFee, is not without limitations. It relies on historical feedback data within an exercise and propagates it to similar text blocks, limiting its ability to generate feedback for unique or dissimilar text segments. Additionally, it currently supports only English and offers exclusively referenced feedback.



In summary, while Artemis provides a comprehensive manual assessment framework for text exercises, it offers limited automation capabilities. The current system shows gaps in supporting multiple languages and generating feedback for diverse text segments. These are all areas where our proposed integration of LLMs could provide significant enhancements.

### 4.2.2 Assessment of Programming Exercises

Artemis deploys a range of techniques for assessing programming exercises, including manual assessment, automatic testing, and static code analysis. The assessment user interface, as shown in Figure 4.2, contains a code view with a file browser in the left sidebar, a grading instructions panel on the right, the test results at the top-right, and the build output followed by unreferenced feedback at the bottom. The interface indicates changed files in yellow, while added or modified lines are highlighted in green. Tutors have the option to give both referenced and unreferenced feedback, similar to text exercises.

The screenshot displays the Artemis assessment interface for a programming exercise titled "Sorting with the Strategy Pattern [100 Points]". The interface is divided into several sections:

- Top Bar:** Includes "Assessment" status, a lock indicator "You have the lock for this assessment", and buttons for "Save", "Submit", "Cancel", and "Exercise Dashboard".
- File Browser (Left):** Shows a directory structure with files like "BubbleSort.java", "Client.java", "Context.java", "MergeSort.java", "Policy.java", and "SortStrategy.java".
- Code Editor (Center):** Displays the source code for "BubbleSort.java". The code is highlighted in green, indicating it is the current file being viewed. A "Tutor Comment" section shows "Good job!" with a score of "10P".
- Instructions Panel (Right):** Contains "Assessment Instructions" and "Structured Assessment Criteria". It includes a table of feedback elements:
 

Points	Feedback	Score
3P	Correct implementation	1
2P	Slightly wrong	1
- Feedback Panel (Bottom):** Shows a score of "25" and the message "You have implemented everything correctly!".

**Figure 4.2:** Programming Exercise Assessment Features in Artemis. (Screenshot)

Grading options in programming exercises are similar to those in text exercises. The interface allows tutors to distribute points within a defined

range. Artemis also employs grading instructions, presented in the right sidebar, which come in both unstructured text and structured grading instructions, paralleling the text exercise assessment process.

Statistics as of March 29, 2023, reveal that Java leads the exercise submissions, accounting for 77.4% of all programming languages used, followed by C at 10.7% and OCaml at 5.3%, as shown in Figure A.9. This signals the importance of Java expertise in the feedback system. Automatic feedback significantly outnumbers manual feedback, as depicted in Figure A.10, emphasizing the system’s efficiency but also highlighting the need for improvements in the manual feedback process. When manual feedback is provided, it is mainly unreferenced, with 88.5% falling into this category, leaving only 11.5% as referenced feedback, as indicated in Figure A.12. Structured grading instructions are rarely used in manual feedback, as shown in Figure A.11, suggesting an area for potential improvement.

Artemis excels in automated assessments, leveraging automatic tests and static code analysis to validate code correctness and assess basic quality. However, these methods have limitations, such as an inability to confirm adherence to specific instructional criteria or handle non-executable code. Furthermore, the inherent complexity of student-submitted code complicates manual reviews, and the system provides insufficient support for tutors navigating this labor-intensive task, especially in large courses.

In summary, while Artemis excels in automated assessments, it has notable shortcomings in facilitating manual feedback for programming exercises, particularly in the context of complex code and large courses. Integrating LLMs could address these gaps, streamlining the manual assessment process and enhancing its quality.

## 4.3 Proposed System

Moving beyond the current system implemented in Artemis, we will outline the functional and non-functional requirements of our proposed system. We validate our requirements through Bruegge and Dutoit’s criteria: completeness, consistency, clarity, and correctness, as well as realism, verifiability, and traceability [BD10].

### 4.3.1 Functional Requirements

According to Bruegge and Dutoit, functional requirements describe the interactions between the system and its environment, irrespective of its implementation [BD10]. Here, we outline the functional requirements of our

proposed system, focusing on the Learning Management Context and the Research and Development Context, as defined in Section 4.1.

### Learning Management Context

Functional requirements in the Learning Management Context primarily aim to optimize the pedagogical process through automated, personalized feedback suggestions within an LMS, specifically Artemis.

#### Text Exercises

- FR T.1 **Generate Feedback Suggestions:** When a student's submission is received, the system automatically generates feedback suggestions using the problem statement, grading instructions, and other relevant information.
- FR T.2 **Review Feedback Suggestions:** After the system generates feedback suggestions, the interface provides tutors with options to accept or reject each suggestion, both unreferenced and referenced.
- FR T.3 **Learn from Feedback:** Once approved or modified feedback is received from tutors, the system updates its machine learning algorithms to refine future suggestions, restricted by available computational resources.
- FR T.4 **Link Structured Grading Instruction:** If structured grading instructions are available for a text exercise, the system aligns the generated feedback suggestions with these instructions to maintain consistency.
- FR T.5 **Handle Multiple Languages:** For each text submission, the system supports multilingual feedback suggestions, primarily in English and German.

#### Programming Exercises

- FR P.1 **Generate Feedback Suggestions:** Upon receiving a student's programming submission, the system automatically generates feedback suggestions using template and solution repositories, the problem statement, and grading instructions, among other relevant information.

- FR P.2 **Review Feedback Suggestions:** Once the system generates code-related feedback, the interface allows tutors to accept or reject each suggestion. This can be done directly within the code view for both unreferenced and referenced feedback.
- FR P.3 **Handle Multiple Programming Languages:** For each programming exercise, the system offers feedback generation in multiple languages, primarily Java, then C and OCaml.
- FR P.4 **Learn from Feedback:** After receiving approved or modified feedback from tutors, the system updates its machine learning algorithms to refine future suggestions, constrained by computational resources.
- FR P.5 **Link Structured Grading Instruction:** If structured grading instructions are provided for a programming exercise, the system aligns generated feedback with these instructions to maintain coherence.
- FR P.6 **Reference Test Results:** During feedback generation, the system incorporates test results to enrich feedback suggestions, subject to the availability of predefined tests and their results.
- FR P.7 **Reference Build Outputs:** While generating feedback, the system includes outputs from the build process, such as compiler errors or warnings, in the provided feedback, bounded by the availability of build logs.

### Research and Development Context

Functional requirements in this context primarily aim to facilitate research and iterative development. They are crucial for the adaptability and improvement of the feedback generation mechanism.

- FR R.1 **Use Multiple LLMs:** When initiating a research experiment, the system should provide options to employ various LLMs, enabling comparative analyses of their feedback generation capabilities, constrained by the system's computational limits.
- FR R.2 **Configure Feedback Generator:** Prior to feedback generation, researchers should be able to configure various parameters and modules through a user interface.
- FR R.3 **Evaluate Feedback Quality:** Once feedback suggestions are generated, the system should enable their evaluation against preset benchmarks, to continually refine the generation process' efficacy.

- FR R.4 **Track Token Usage and Generation Time:** During the execution of feedback generation, the system should automatically record the number of tokens used and the time taken by the LLM, to facilitate cost-benefit analysis.
- FR R.5 **Compare Modules:** After generating feedback suggestions, the system should facilitate the direct comparison of different feedback generation modules, to pinpoint the most effective strategies.
- FR R.6 **Define Experiment:** Before an experiment is run, the system should allow the definition of its scope and parameters, enabling the application of scientific rigor to the evaluation process.
- FR R.7 **Conduct Experiment:** After defining an experiment, the system should automate its execution, accommodating modes such as batch or incremental learning, constrained by computational resources.
- FR R.8 **Rate Feedback:** Following the generation of feedback suggestions while conducting an experiment, the system should permit manual rating of their quality, to further inform machine learning models.
- FR R.9 **Automatic Evaluation:** After an experiment concludes, the system should derive automatic ratings based on preset criteria, to speed up the evaluation process.
- FR R.10 **Import Configurations:** When initializing a research setting, the system should facilitate the importation of configuration settings, enabling repeatability.
- FR R.11 **Export Data:** Following an experiment, the system should permit the exportation of collected data and results, for external analysis and reporting.

### 4.3.2 Non-Functional Requirements

Non-functional requirements describe the system's properties or qualities that are not directly related to its functional behavior [BD10]. Here, we outline the non-functional requirements that are crucial for the proposed system's success.

### Usability

- NFR1 **User-Friendly Interface:** The system should provide an intuitive and easy-to-navigate user interface for tutors and researchers. It should be consistent with the existing Artemis interface to minimize the learning curve for tutors while reviewing feedback suggestions. researchers should be able to easily access the system's research and development features to facilitate experimentation and iteration.
- NFR2 **Documentation:** The system should be accompanied by detailed documentation, including user guides and technical manuals, to assist tutors, future developers, and researchers.
- NFR3 **Cognitive Load:** Feedback suggestions by the system should reduce the cognitive load on tutors and should not impose a burden during review.

### Reliability

- NFR4 **Fault Tolerance:** If the system fails to generate feedback suggestions, the failure should be graceful, allowing tutors to continue manual assessment without disruption.
- NFR5 **Submission Coverage:** The system should be capable of generating feedback suggestions for a significant portion of a submission, ideally covering at least 70% of the submission.
- NFR6 **Feedback Accuracy:** The system should generate feedback suggestions that are at least 80% accurate, meaning that they are accepted by tutors without modification.
- NFR7 **Feedback Quality:** The system should generate feedback suggestions that are at least the same quality as manual feedback.

### Performance

- NFR8 **Generation Speed:** Feedback suggestions should be generated within a tolerable time frame, ideally not exceeding 15 seconds for each submission.
- NFR9 **Response Time:** When a tutor starts assessing a submission, the system should respond within a few seconds with feedback suggestions, or immediately if the suggestions are pre-generated.

NFR10 **Scalability**: The system must be able to handle increased loads — from a large number of students or more complex exercises — without experiencing significant performance degradation.

### Supportability

NFR11 **Cost Efficiency**: Running the system should be more cost-effective than human labor, either by reducing the time spent by tutors or by enhancing the quality of feedback substantially.

NFR12 **Module Extensibility**: The system’s architecture must allow for the easy addition or modification of feedback generation processes to facilitate future development and research.

## 4.4 System Models

This section begins with the presentation of possible scenarios for the proposed system. These scenarios are then further detailed using use cases, which help us to define the analysis object model and dynamic model. The final part of this section outlines the expected changes in the Artemis user interface, demonstrated using mockups. We employ Unified Modeling Language (UML), a standard set of notations for representing models, to offer precise specifications [BD10].

### 4.4.1 Scenarios

A scenario is a narrative that describes the interaction between a user and a system, focusing on the execution of a specific feature. It serves as a valuable tool for eliciting system requirements [BD10]. While the following visionary scenario serves as an aspirational use-case for student interactions, the primary focus of this thesis lies in the two demo scenarios. These specifically address the system’s implications and functionalities for tutors.

#### Visionary Scenario

**Imminent Feedback for Interactive Learning for Students.** Alice, a freshman-student at the university, is working on her weekly programming assignment for “Fundamentals of Programming” via Artemis. Although, Artemis’ automatic tests and static code analysis have flagged some errors, she is still confused about some programming concepts while implementing a

sorting algorithm in Java. In need of guidance, Alice turns to a new feature called *Interactive Tutor*.

Upon activation, the Interactive Tutor immediately pinpoints the recursive function in her code and asks, “It looks like you’re having trouble with recursion. Would you like some tips?” Alice clicks “Yes”, and the Tutor elaborates, “Recursion requires a base case to prevent infinite loops. Your current code lacks that”. This information directly addresses the conceptual gap that neither automatic tests nor static code analysis could fill.

Alice, appreciative but still a bit puzzled, types in a follow-up question: “How do I actually write a base case in Java? Is it different from a regular ‘if’ statement?” The Tutor responds, “In Java, a base case is generally implemented using an ‘if’ statement at the beginning of the recursive function. The key is to return a value that doesn’t call the function again.” It then shows her a sample Java code snippet illustrating a proper base case for a recursive function.

Motivated by the example, Alice modifies her code and incorporates a fitting base case. The Interactive Tutor acknowledges her progress, stating, “Great, your base case looks solid now!” She clicks the provided “Learn More” link, leading her to lecture slides for further study.

As Alice reflects on her encounter with the Interactive Tutor, she feels immensely supported in her learning journey. The feature’s ability to provide immediate, personalized feedback elevates her understanding of complex programming concepts. Beyond mere comprehension, it instills in her a newfound confidence and independence, empowering her to tackle future challenges with greater self-assurance.

### Demo Scenarios

**Assisted Manual Assessment for Tutors.** Bob, a seasoned tutor for the “Fundamentals of Programming” course, is faced with the challenge of grading assignments for over 30 students each week. Though well-versed in programming, the sheer volume and repetition involved in manual assessments present a bottleneck. The latest version of Artemis introduces enhancements to Athena, its automated assessment framework, designed to alleviate such issues.

Bob initiates the assessment of Alice’s programming submission. The code is annotated with feedback suggestions from Athena, labeled as “Suggestion”. He reviews each suggestion and uses the “Accept” or “Reject” button to either incorporate or discard them. Additionally, Bob opts to edit some of the accepted suggestions, revising the feedback’s description and points to improve accuracy. Finally, he supplements Athena’s recommen-



dations with additional feedback and a personal note before finalizing the assessment by clicking “Submit”.

This hybrid assessment strategy improves Bob’s efficiency and allows him to offer nuanced feedback. Athena’s automation reduces his workload while amplifying the quality of the assessments. Thus, the combined strengths of human expertise and automated precision provide a robust, efficient, and practical assessment paradigm, enriching the educational experience for tutors and students alike.

**Research and Development Environment for Researchers.** Dr. Caro, a university researcher, anticipates the transformative potential of a newly released machine learning model called *AiThENA-3* for Athena. Excited to explore this opportunity, she quickly integrates AiThENA-3 into the codebase for text exercise assessment and utilizes the *Playground*, Athena’s research and development dashboard, to conduct her experiment.

Dr. Caro configures her experiment within the Playground, selecting historical Artemis data as the data source, a text exercise as the target, and correctness as the evaluation metric. She then configures the feedback generators by setting up the AiThENA-3 module and importing the CoFee module as a baseline using the “Import” button. Afterwards, she initiates the experiment by clicking “Conduct Experiment”.

Upon reviewing the experiment’s output, Dr. Caro can already eyeball AiThENA-3’s superior performance over CoFee. To validate her observations, she reviews the automated correctness estimates against her own ratings. While she finds the estimates generally accurate, there are some divergences which she corrects manually. She finalizes the experiment by clicking “Finish”.

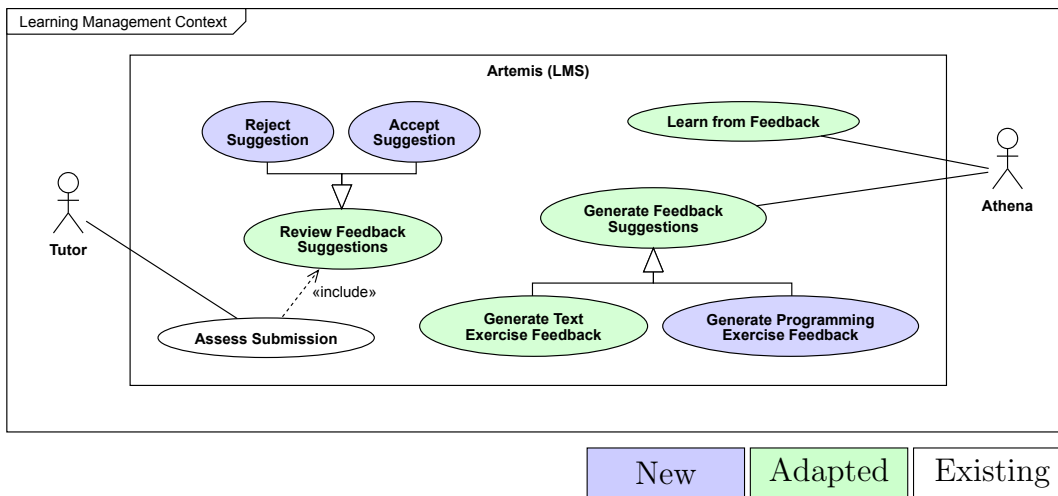
Dr. Caro then exports the experiment’s configuration and data by clicking “Export”, enabling further analysis and potential academic publication. Convinced by the experiment’s outcomes, she moves forward with integrating AiThENA-3 into Athena’s production environment, thus enhancing the system’s feedback generation capabilities.

#### 4.4.2 Use Case Model

Transitioning from scenarios, which offer specific instances of user interactions, we now explore use cases based on the demo scenarios. These serve as comprehensive representations of functionalities, capturing all possible scenarios initiated by different actors [BD10]. This approach allows for a structured analysis within both the Learning Management and Research and Development contexts.

### Learning Management Context

Figure 4.3 illustrates the use case diagram within the Learning Management Context, Artemis, featuring the interactions between tutors and Athena. Central to this interaction is the **Assess Submission** use case, an existing function within Artemis. Tutors begin the assessment process by examining student submissions, which leads them to the **Review Feedback Suggestions** use case (FR T.2, FR P.2).



**Figure 4.3:** UML Use Case Diagram for the Learning Management Context. Depicting the interaction of the Tutor and Athena, the automated feedback system within Artemis (LMS). New, adapted, and existing use cases are marked.

The **Review Feedback Suggestions** use case has been refined to accommodate both text and programming exercises. While it previously only supported referenced feedback suggestions, it now offers tutors a new option where they can accept or reject suggestions, including unreferenced ones. Previously, all suggestions were automatically accepted by default. Accepted suggestions are incorporated into the assessment, while rejected suggestions are discarded. After accepting a suggestion, tutors can edit or delete it as part of the **Assess Submission** use case.

On the side of Athena, the automated feedback system, there are several use cases aimed at enriching its capabilities. It **Generates Feedback Suggestions** for both text (FR T.1) and programming exercises (FR P.1). The use case related to text exercises is adapted, whereas the one for programming exercises is new. Athena can also adapt and **Learn from Feedback** (FR T.3, FR P.4) provided by tutors to improve future suggestions.

In the use case **Generate Feedback Suggestions**, Athena, incorporates advanced features to enrich the feedback suggestions. For text exercises, the feature **Handle Multiple Languages** (FR T.5) enables feedback suggestions for multilingual submissions. In the context of programming exercises, we include three key features: **Handle Multiple Programming Languages** (FR P.3), **Reference Test Results** (FR P.6), and **Reference Build Outputs** (FR P.7).

In summary, the Learning Management Context use cases streamline the collaboration between tutors and Athena. This enhances the assessment process in Artemis, contributing to a more effective educational environment.

### Research and Development Context

The Research and Development Context constitutes a crucial part of Athena's functionality and serves as a playground for researchers to experiment with feedback generation algorithms, including various LLMs. The focus here is on a set of use cases that enable researchers to establish rigorous research experiments, iterate on them, and evaluate the outcomes systematically.

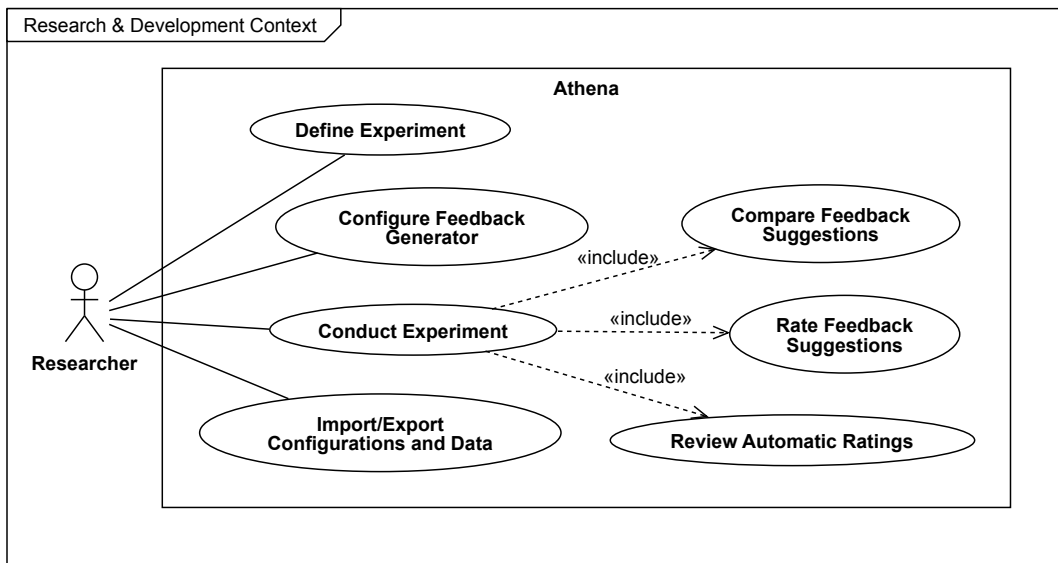
Figure 4.4 depicts a structured view of interactions between researchers and Athena in this context. A cornerstone use case is **Define Experiment** (FR R.6), where researchers can set the scope and evaluation metrics for an experiment. It serves as the precursor to the **Configure Feedback Generator** use case (FR R.2), where different LLMs can be selected and parameters can be adjusted to tailor the feedback generation process (FR R.1).

Upon configuring the experiment, researchers can initiate it using the **Conduct Experiment** use case (FR R.7). This use case encompasses critical functionalities such as tracking token usage and generation time (FR R.4). It includes the **Compare Feedback Suggestions** use case (FR R.5), where outputs from different feedback generation modules are directly compared, providing actionable insights into their efficacy.

As part of the experiment evaluation while a researcher is conducting an experiment, the **Rate Feedback Suggestions** use case (FR R.8) enables researchers to manually grade the quality of generated feedback. This manual rating serves as a valuable data point for the **Review Automatic Ratings** use case (FR R.9), where these manual ratings are compared with the system's own metrics for evaluating the feedback quality (FR R.3).

Finally, the **Import/Export Configuration and Data** use case facilitates the researcher's workflow by allowing the importation of pre-defined experiment configurations (FR R.10) and the exportation of experiment data for further analysis (FR R.11).

This array of use cases equips researchers with a robust set of tools to



**Figure 4.4:** UML Use Case Diagram for the Research and Development Context. Illustrating the interaction between the Researcher and Athena in a research and development environment.

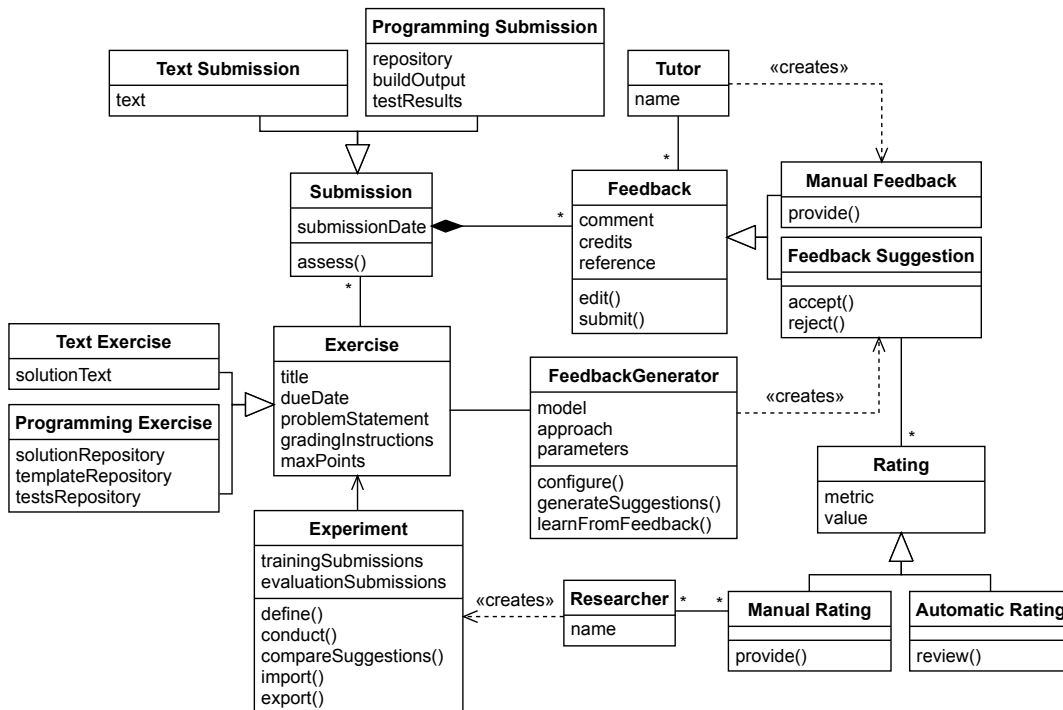
iteratively improve Athena’s feedback generation capabilities, thereby enhancing the effectiveness of the Learning Management System, Artemis, in which Athena is embedded.

### 4.4.3 Analysis Object Model

The Analysis Object Model (AOM) serves as a conceptual framework that captures the key entities, attributes, and relationships in the user’s view of the system. Rooted in UML class diagrams, it provides a high-level abstraction of the problem domain, essentially acting as a visual dictionary for comprehending main user-level concepts [BD10]. In the following, we present the AOM for the proposed system, as depicted in Figure 4.5.

In the AOM, an **Exercise** — with a title, due date, problem statement, grading instructions, and maximum achievable points — is either a **Text Exercise**, with a solution text, or a **Programming Exercise**, with solution, template, and test repositories. Depending on the exercise type, a **Submission** is either a **Text Submission**, with a submission text, or a **Programming Submission**, with a submission repository, build output and test results.

A **Tutor** can *assess* a submission with multiple pieces of **Feedback** — each containing a comment, credits, and a reference within the submission — and *submit* the feedback for the submission. Feedback is either **Manual**



**Figure 4.5:** UML Class Diagram for the Analysis Object Model of the Proposed System.

**Feedback**, that a tutor can *provide*, or an automatically generated **Feedback Suggestion**. The tutor can *accept* or *reject* a feedback suggestion and *edit feedback* before submitting it.

The **Feedback Generator** associated with the exercise can *generate suggestions* for its submissions, and *learn from feedback* submitted by tutors to improve future suggestions. For enhancing the feedback generation process, a **Researcher** can *define an experiment* for an exercise together with a set of submissions for training and evaluation. The researcher can then *configure the feedback generator* to select the model, approach, and parameters used for the experiment. Following that, the researcher can *conduct the experiment* generating feedback suggestions for the evaluation submissions.

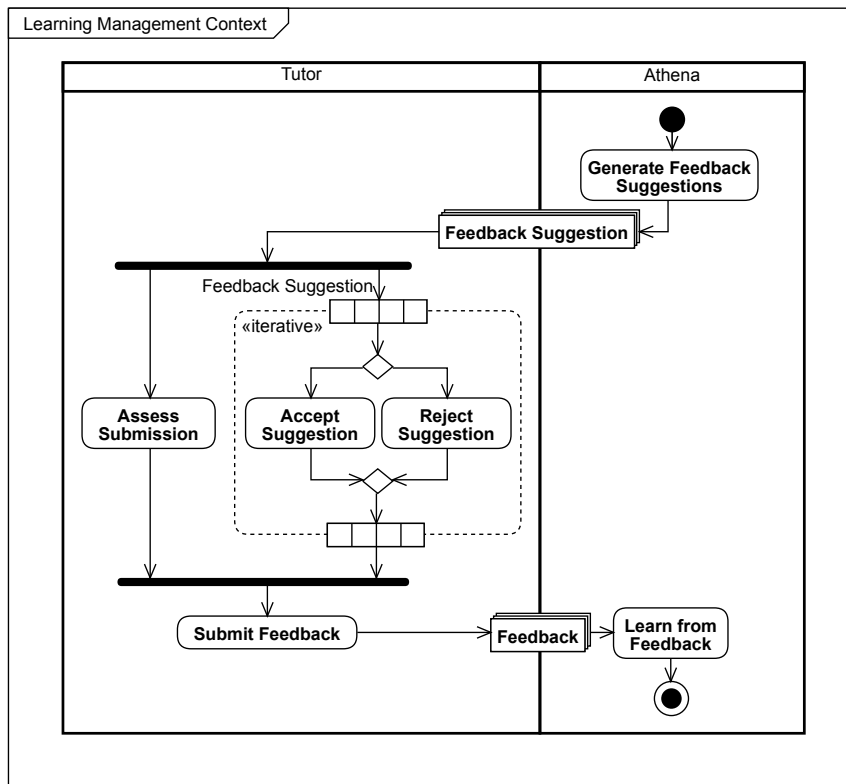
A **Rating** is a metric-value pair, which can be either an **Automatic Rating** or a **Manual Rating**. For evaluation purposes, the system automatically generates ratings for the feedback suggestions. Researchers can *compare suggestions* in order to efficiently *review automatic ratings* and *provide manual ratings* for the feedback suggestions. Finally, the researcher can *export* the experiment data and *import* experiment configurations for repeatability.

### 4.4.4 Dynamic Model

In the following subsection, we turn our attention to the Dynamic Model. Using UML activity diagrams, we analyze the behavior of the proposed system within the contexts of learning management and research and development.

#### Learning Management Context

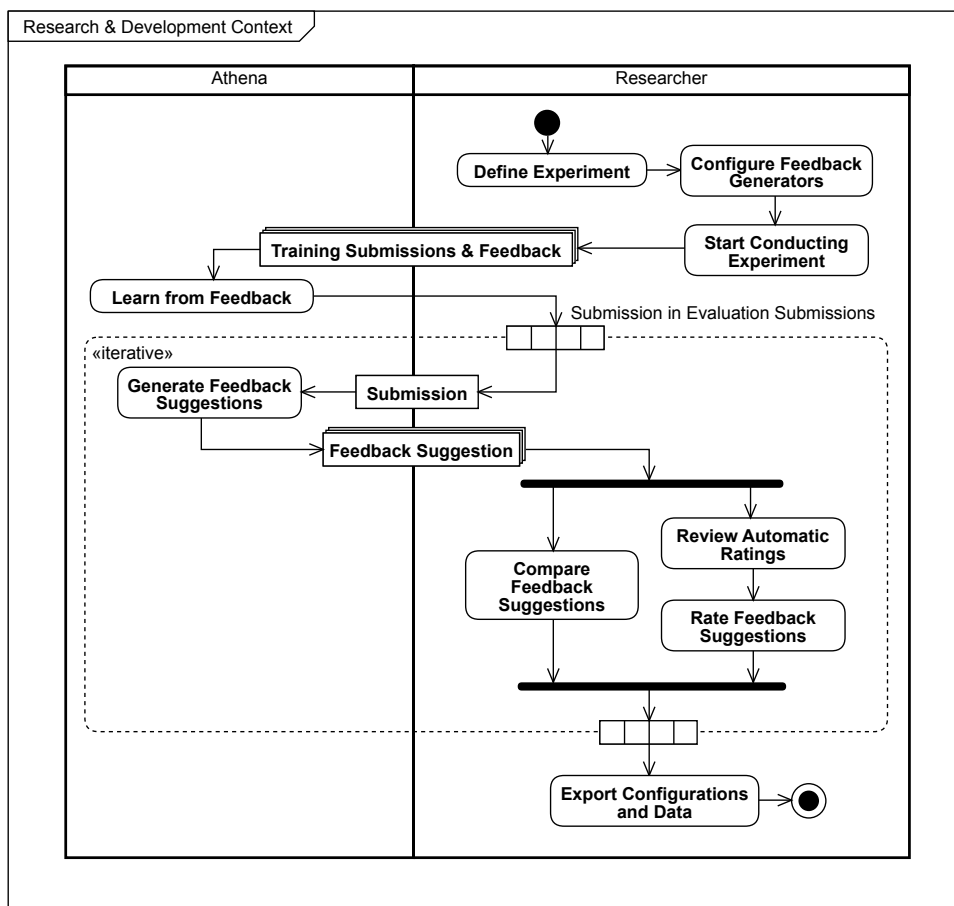
Figure 4.6 shows an activity diagram for the assessment workflow involving a tutor and Athena. Before the assessment by a tutor begins, Athena starts with generating feedback suggestions for a submission and passes them to the tutor. The tutor then starts to manually assess the submission as usual and reviews the suggestions in parallel. Each feedback suggestion can be either accepted or rejected by the tutor. If a suggestion is accepted, it is incorporated into the assessment. If a suggestion is rejected, it is discarded. In both cases, the tutor can edit the suggestions as part of the assessment, before submitting the feedback. Once the feedback is submitted, it gets passed to Athena, which learns from the feedback to improve future suggestions.



**Figure 4.6:** UML Activity Diagram for the Assessment Workflow Within the Learning Management Context.

### Research and Development Context

The activity diagram in Figure 4.7 illustrates the workflow of conducting an experiment within the research and development context. The researcher starts by defining an experiment, which includes the exercise, submissions, and evaluation metric. Then, the researcher configures the feedback generator by selecting the model, approach, and parameters. Afterwards, the researcher starts conducting the experiment, which first passes the training submissions with its feedback to Athena to learn from them. Then, for each evaluation submission, Athena generates feedback suggestions, which are passed to the researcher for review. The researcher can then compare the suggestions, while reviewing the automatic ratings and providing manual ratings. Finally, the researcher can export the configurations and data for further analysis.



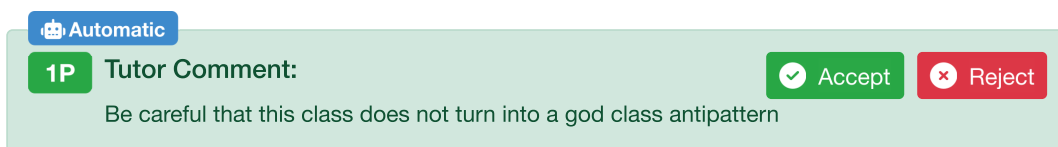
**Figure 4.7:** UML Activity Diagram for the Workflow of Conducting an Experiment Within the Research and Development Context.

### 4.4.5 User Interface

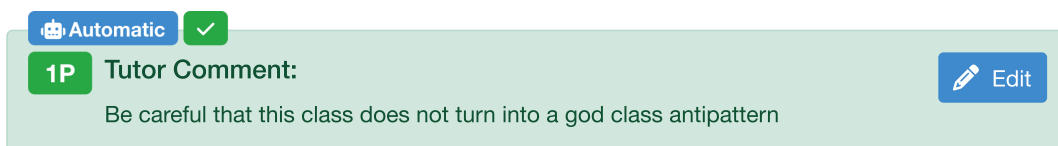
This subsection elaborates on the alterations needed in the Artemis user interface to effectively incorporate the new feedback suggestion system. We focus on designs within the Learning Management Context, leaving out the Research and Development Context due to its evolving nature. Our design mockups aim for coherence with the existing Artemis UI features.

Figure 4.8 outlines the design elements for introducing inline feedback suggestions in both text and programming exercises. Here, we adopt the established “Automatic” badge from existing text exercise suggestions, as seen earlier in Figure 4.1a. Contrary to the current setup where feedback for text exercises is auto-accepted, the new UI incorporates “Accept” and “Reject” buttons. These buttons are displayed in Figure 4.8a. Accepting a suggestion adds it to the assessment and allows for further editing, while rejection removes it. Accepted suggestions will carry a green checkmark in addition to the “Automatic” badge as shown in Figure 4.8b.

These buttons serve an essential function: they enable tutors to examine suggestions before incorporating them into assessments. This approach supports a more conscious decision-making process, contributing to the project’s research and development data collection objectives, while also reducing the risk of bias.



(a) Review referenced feedback suggestion



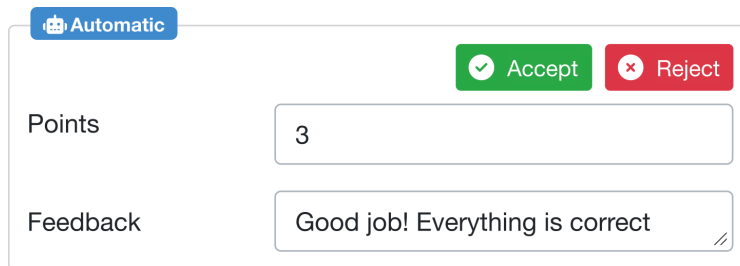
(b) Accepted Referenced Feedback Suggestion

**Figure 4.8:** Proposed Inline Feedback Suggestion UI. (Mockup)

Similarly, for unreferenced feedback suggestions, shown in Figure 4.9, we introduce “Accept” and “Reject” buttons. On accepting, the suggestion merges into the assessment and becomes editable. Rejected suggestions will be deleted. An accepted suggestion will display a green checkmark next to its badge, making it consistent with referenced feedback. This is particularly new for text exercises and is visualized in Figure 4.9b.



To clarify, the introduction of feedback suggestions for programming exercises is completely new, and for text exercises, the novelty lies in offering unreferenced feedback suggestions and a new choice architecture. These features enhance the existing framework, offering tutors an improved, thoughtful assessment experience.



Automatic

Accept Reject

Points 3

Feedback Good job! Everything is correct

This mockup shows a review interface for an unreferenced feedback suggestion. At the top left, there is a blue button labeled 'Automatic' with a small icon. To its right are two buttons: a green 'Accept' button with a white checkmark and a red 'Reject' button with a white 'x'. Below these buttons are two input fields. The first is labeled 'Points' and contains the number '3'. The second is labeled 'Feedback' and contains the text 'Good job! Everything is correct' followed by a double-slash icon.

(a) Review unreferenced feedback suggestion



Automatic

Points 3

Feedback Good job! Everything is correct

This mockup shows the state of the interface after the suggestion has been accepted. The 'Automatic' button is now greyed out and has a green checkmark next to it. A trash can icon is visible in the top right corner of the main container. The 'Points' and 'Feedback' fields remain the same as in mockup (a).

(b) Accepted Unreferenced Feedback Suggestion

**Figure 4.9:** Proposed Unreferenced Feedback Suggestion UI. (Mockup)

# Chapter 5

## System Design

In accordance with the System Design Document (SDD) framework outlined by Bruegge and Dutoit [BD10], this chapter describes the design of our system. We initiate by presenting an overview in Section 5.1. Next, we articulate the design goals in Section 5.2, drawing upon the non-functional requirements discussed in the previous chapter. Finally, we expand on subsystem decomposition and hardware-software mapping of our proposed system in Section 5.3 and Figure 5.5, respectively.

### 5.1 Overview

Aiming to augment the Artemis platform, our design introduces significant architectural modifications to the initial Athena system. Initially supporting only the CoFee approach for text exercises, Athena evolves to accommodate multiple feedback generation algorithms via a microservices architecture. The proposed LLM-based feedback generation modules are seamlessly integrated into this newly restructured architecture. Complementing this, we introduce the Research and Development (R&D) Playground, a dedicated component for iteratively experimenting with automated feedback mechanisms.

Importantly, for the integration with Artemis, the design capitalizes on the existing architectural foundation of Artemis. This approach mitigates the need for extensive refactoring and allows for quicker adoption by developers already familiar with the Artemis system. It thereby satisfies both architectural consistency and developmental efficiency.

## 5.2 Design Goals

Based on the outlined non-functional requirements in Section 4.3.2, we establish the design goals essential to the success of our proposed system. Additionally, we deliberate on the prioritization of these goals and consider potential trade-offs. For systematic evaluation, we adopt the design criteria classification proposed by Bruegge and Dutoit [BD10], categorizing our design goals into five groups: Performance, Dependability, Cost, Maintenance, and End-user criteria.

**Performance.** The primary performance criteria include response time (NFR9), scalability (NFR10), and feedback generation speed (NFR8). Our system aims to provide feedback suggestions immediately after a tutor initiates the assessment process, ideally within a few seconds. Scalability is crucial as the system must handle increased loads, such as a large number of students or more complex exercises, without significant degradation in performance. While feedback generation speed is not as critical as response time, it should be fast enough to be usable at scale, *i.e.* not exceeding 15 seconds for each submission, if the generation process is not parallelized.

**Dependability.** Key factors for system dependability include fault tolerance (NFR4), submission coverage (NFR5), feedback accuracy (NFR6), and feedback quality (NFR7). The system should minimize disruption in case of failure, allowing tutors to proceed with manual assessments. Aimed to cover at least 70% of a student’s submission, the system’s generated feedback should maintain an 85% accuracy level to ensure its usefulness and acceptance among tutors. Quality-wise, the system’s feedback should be on par with manual feedback provided by tutors.

**Cost.** The primary cost criteria involve cost efficiency (NFR11). Given that Artemis is an open-source project with limited funding, it is essential for our system — an intended extension of Artemis — to minimize operational costs. The proposed system aims to be more cost-effective than human labor, either by reducing the time tutors spend on feedback or by significantly enhancing feedback quality.

**Maintenance.** The key maintenance criteria include module extensibility (NFR12) and documentation (NFR2). Our system aims to be built on a flexible architecture that allows for easy addition or modification of feedback generation processes. This design goal will enable future development and research, making it easier for subsequent developers to contribute to the

project. Furthermore, comprehensive documentation is a priority to aid in understanding the system’s architecture and functionalities, thus simplifying maintenance efforts.

**End user criteria.** For the end user, the criteria focus on usability (NFR1) and cognitive load (NFR3). The system aims to offer an intuitive and easy-to-navigate user interface that is consistent with the existing Artemis interface. This is to ensure a minimized learning curve for tutors and facilitate ease of access to research and development features for researchers. Additionally, feedback suggestions should not add a cognitive burden on tutors, making it easier to review and apply. In terms of data protection, compliance with GDPR is essential to safeguard user data, ensuring that it does not leave German jurisdiction unless compliance can be confirmed.

**Prioritization and Trade-offs.** In prioritizing our design goals, dependability takes precedence because the system’s reliability directly impacts user trust and the overall success of the system. Next, we focus on performance and end-user criteria, as quick response times and usability significantly influence user experience. Cost and maintenance follow closely, emphasizing long-term sustainability.

Trade-offs are inevitable. For instance, achieving high dependability may increase operational costs, potentially conflicting with our cost-efficiency goal. Similarly, prioritizing rapid response times might compromise the quality of generated feedback, impacting dependability. These trade-offs are carefully considered and balanced to optimize system performance without sacrificing critical aspects such as dependability and user experience.

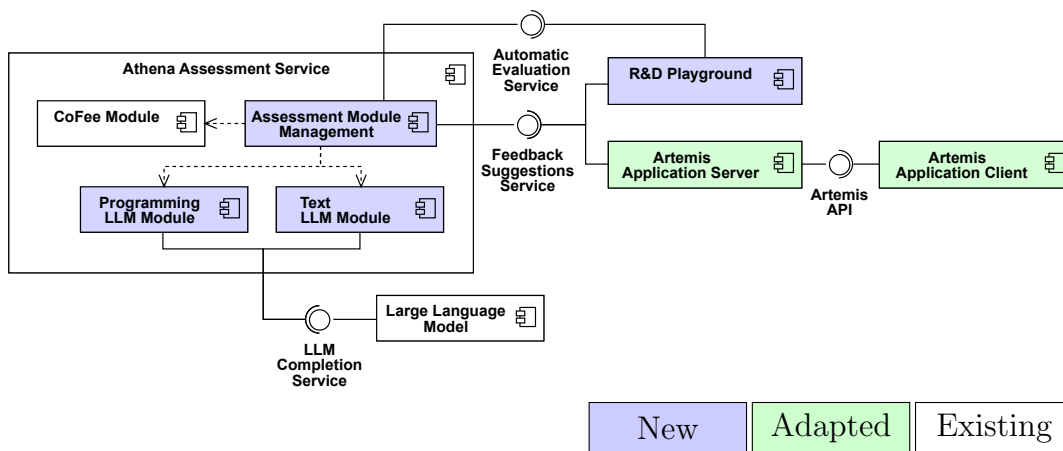
This strategic prioritization ensures the system’s robustness while accommodating budget and time constraints, thus aligning closely with both end-user needs and project objectives.

### 5.3 Subsystem Decomposition

To improve semi-automatic assessment in Artemis, it is necessary to break down the overall architecture into its main parts or subsystems. This makes it easier to understand the system and plan future changes. We focus on four main subsystems: *Athena Assessment Service* (Section 5.3.1), *Text/Programming LLM Module* (Section 5.3.2), *Research and Development Playground* (Section 5.3.3), and *Artemis* itself (Section 5.3.4).

### 5.3.1 Athena Assessment Service

The *Athena Assessment Service* is the heart of the system. It manages and runs the automated feedback for learning management systems such as *Artemis*. As shown in Figure 5.1, this subsystem connects with the *Artemis Application Server*, the *R&D Playground*, and the *Large Language Model* service. Inside the *Athena Assessment Service*, we find several components: *Assessment Module Management*, *CoFee Module*, *Text LLM Module*, and *Programming LLM Module*.



**Figure 5.1:** UML Component Diagram for the Top-Level Subsystem Decomposition. New, adapted, and existing components are marked.

The *Assessment Module Management* component serves as the orchestrator, streamlining communication between *Athena Assessment Service* and external systems such as the *Artemis Application Server* and the *R&D Playground*. When activated via a REST API, this module fetches assessment data from the connected system, routes it to the relevant internal module, and returns the generated feedback suggestions either to *Artemis* or the *R&D Playground*.

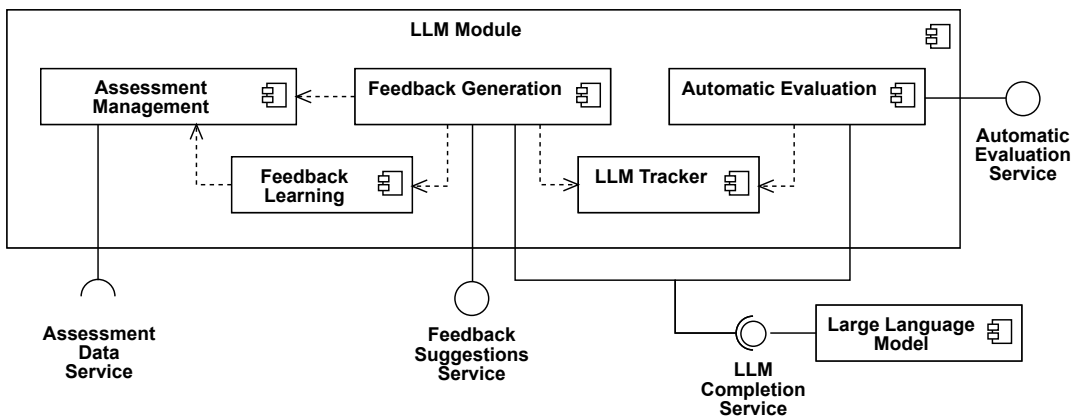
The *CoFee Module* stands as the existing module for generating feedback on text exercises, following the established method [Ber22]. *Assessment Module Management* invokes this module when the CoFee approach is specified for an exercise, yielding feedback suggestions accordingly.

Both the *Text LLM Module* and the *Programming LLM Module* introduce new avenues for feedback generation, leveraging the capabilities of large language models. These modules interact with an external *Large Language Model* service to produce natural language text completions to facilitate the feedback generation process.

A REST API underlies the interactions among these components, as well as with the *Artemis Application Server* and *R&D Playground*. This architecture ensures modularity, scalability, and flexibility, making it conducive to future extensions. It aligns seamlessly with both the immediate operational requirements and the long-term research goals, thereby fulfilling the system’s multi-faceted objectives.

### 5.3.2 Text/Programming LLM Module

The *LLM Module* encompasses two closely related subsystems: *Text LLM Module* and *Programming LLM Module*. These subsystems share the same foundational architecture, as illustrated in Figure 5.2. Their primary responsibility is to generate automated feedback for text and programming exercises using large language models.



**Figure 5.2:** UML Component Diagram for the Subsystem Decomposition of a LLM Module.

At its core, the *Assessment Management* component retrieves and manages data concerning the assessment process. When new feedback becomes available, the *Feedback Learning* component updates the feedback generation model accordingly, enhancing its performance over time.

Meanwhile, the *Feedback Generation* component is responsible for generating feedback suggestions. It incorporates insights from the *Feedback Learning* component and uses the *Large Language Model* to produce these suggestions, which are subsequently made accessible via a dedicated interface.

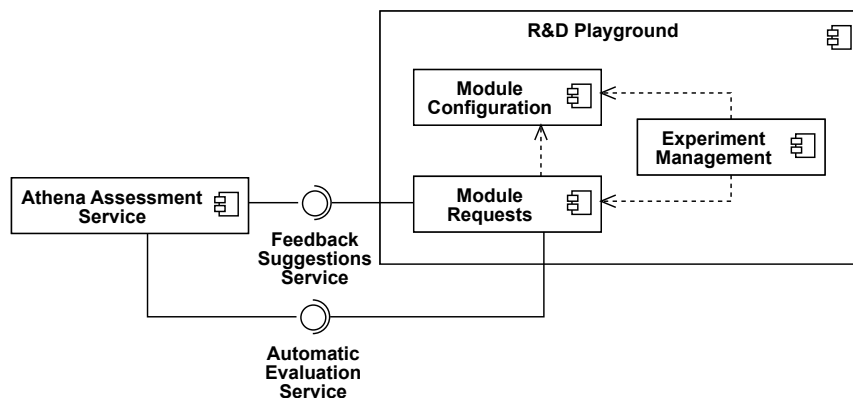
The *Automatic Evaluation* component serves a dual function. Primarily, it gauges the effectiveness of the generated feedback for ongoing quality assurance. Moreover, it employs the *Large Language Model* to make these judgments, providing an objective evaluation of the system’s output. The re-

sults of this evaluation are offered through another distinct interface, aiding in ongoing research and refinement processes.

Lastly, the *LLM Tracker* handles the system’s logging and monitoring needs, keeping track of LLM calls and their results. This information is crucial for debugging, performance optimization, and research and development, thus contributing to the system’s overall dependability.

### 5.3.3 Research and Development Playground

The *R&D Playground* serves as a hub for experimentation, tightly integrated with the *Athena Assessment Service* using both the *Feedback Suggestions Service* and *Automatic Evaluation Service*. As portrayed in Figure 5.3, it comprises three core components: *Module Requests*, *Module Configuration*, and *Experiment Management*.



**Figure 5.3:** UML Component Diagram for the Subsystem Decomposition of the R&D Playground.

A researcher initiates the experimentation cycle by configuring feedback algorithms via the *Module Configuration* component. This configured setup serves as an input for the *Experiment Management* component, which oversees the entire life cycle of an experiment. This involves managing the training data, evaluation submissions, and evaluation metrics. To execute the experiment, *Experiment Management* interacts with the *Module Requests* component, which in turn interfaces with the *Athena Assessment Service* to fetch feedback suggestions and automatic evaluations.

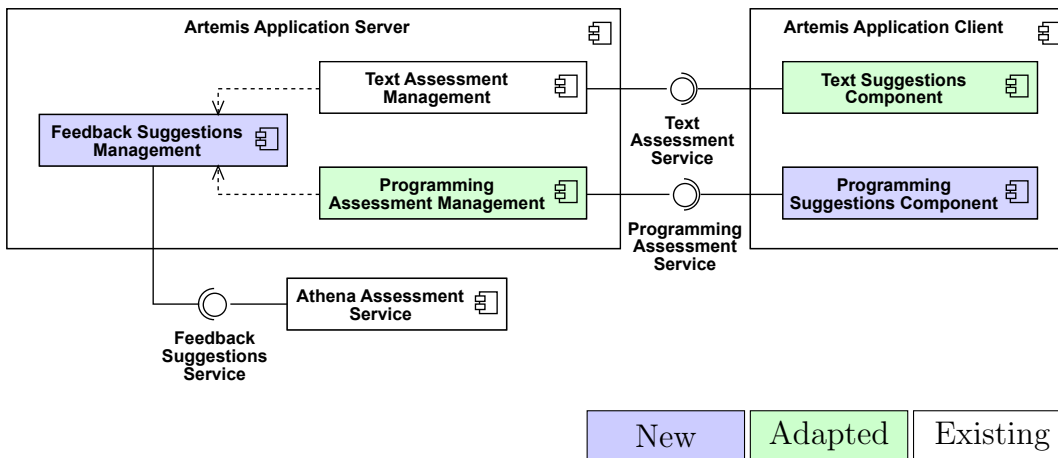
The design of *R&D Playground* underscores its role in providing a robust environment for iterative research. It allows for agile modifications to algorithms, enabling researchers to rapidly assess changes and ensure they align with the design goals. Consequently, this subsystem facilitates methodical

evaluation and optimization of feedback mechanisms, which is central to the Artemis platform’s objectives of enhanced learning outcomes.

To realize the *R&D Playground*, a web application will be developed using the Next.js<sup>1</sup> framework. This choice aligns with the goal of efficient development and easy integration, as Next.js offers server-side rendering capabilities within a React<sup>2</sup>-based environment.

### 5.3.4 Artemis

Artemis, built on a client-server architecture, serves as the primary foundation for integrating our proposed system. The client layer relies on the Angular<sup>3</sup> framework, while the server layer utilizes Spring Boot<sup>4</sup>. For clarity, Figure 5.4 outlines the simplified proposed architectural alterations.



**Figure 5.4:** UML Component Diagram for the Subsystem Decomposition of the Feedback Suggestions Within Artemis. New, adapted, and existing components are marked.

On the server side, the key component is *Feedback Suggestions Management*. This component serves as a replacement for the current component managing feedback suggestions using the CoFee approach. It orchestrates the management of feedback suggestions accommodating diverse exercise types. Its central role is to optimize system performance by pre-generating feedback suggestions, thereby decoupling feedback generation speed from system response time. This ensures that tutors can start an assessment immediately

<sup>1</sup><https://nextjs.org>

<sup>2</sup><https://reactjs.org>

<sup>3</sup><https://angular.io>

<sup>4</sup><https://spring.io/projects/spring-boot>



without waiting for feedback generation. Additionally, this component manages the balance between pre-generated and on-demand feedback, ensuring that the system learns from tutor feedback.

Building on the *Feedback Sugestions Management*, the *Text Assessment Management* and *Programming Assessment Management* component manage the semi-automatic assessment process, channelling the necessary assessment data to the client. On the client-side, two specific components assume responsibility for the user interface: the *Text Sugestions Component* and the *Programming Sugestions Component*. These components display the pre-generated or real-time feedback suggestions to the tutors and handle subsequent actions, such as accepting or rejecting the feedback suggestions.

These enhancements to Artemis contribute to achieving the design goals of performance and dependability, fulfilling the overarching aim of accelerating semi-automatic assessment in an efficient and dependable manner.

## 5.4 Hardware Software Mapping

Athena and Artemis are designed to function efficiently in a distributed computing environment, particularly suited for large educational institutions. As exemplary depicted in Figure 5.5, the *Artemis Application Server*, *Athena Assessment Service*, and *R&D Playground Application Server* are deployed on the computing infrastructure provided by the Technical University of Munich.

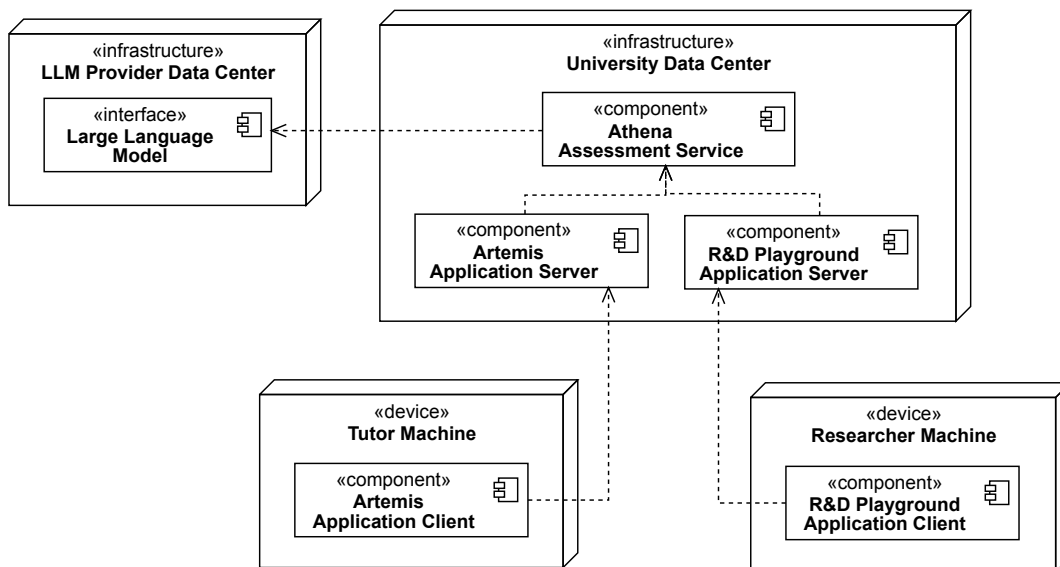


Figure 5.5: UML Deployment Diagram for the Hardware-Software Mapping.

The tutor interacts with the *Artemis Application Server* via a personal computer, gaining access through the *Artemis Application Client*. Similarly, the researcher employs another personal computer for interfacing with the *R&D Playground Application Client*. It is crucial to note that the *Large Language Model* service, a cornerstone for feedback generation, resides on a third-party cloud provider. This cloud-based service is connected to the *Athena Assessment Service*, providing the necessary computational power. Eventually, the LLMs can be hosted on the same infrastructure as the *Athena Assessment Service* if the computational power is sufficient.

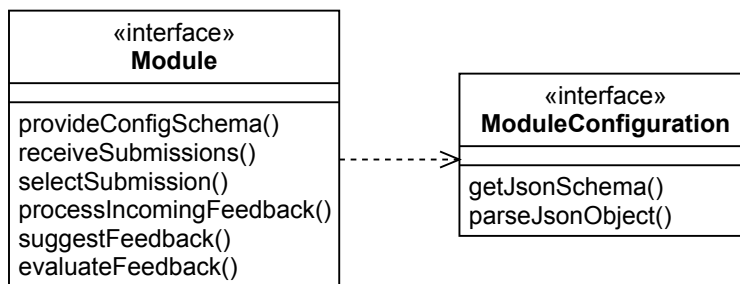
# Chapter 6

## Object Design

This chapter delineates the object design in the solution domain, describing the seamless integration of the system architecture illustrated in Chapter 5 with the Artemis learning platform. The focus here is on the Athena Assessment Service, which acts as the interface for modules such as the LLM modules for text and programming exercises. We also elaborate on the research and development environment established to facilitate future work.

### 6.1 Athena Assessment Service

The Athena Assessment Service serves as the cornerstone for integrating various feedback generation modules for different exercise types. It defines a standardized interface that each module must implement to ensure compatibility with Artemis. Figure 6.1 depicts a simplified UML Class Diagram of this interface. The complete interface definition, along with relevant documentation, resides in the Athena repository<sup>1</sup>.



**Figure 6.1:** UML Class Diagram for the Module Interface. (Simplified)

<sup>1</sup><https://github.com/ls1intum/Athena>

**The interface encompasses the following methods:**

- `receiveSubmissions()`: Accepts assessment data, including exercises and submissions, allowing modules to prepare for the feedback generation process.
- `selectSubmission()`: Selects a submission from the received data for generating feedback. This selection could be guided by various metrics, such as information gain or choosing a submission with pre-generated feedback suggestions.
- `processIncomingFeedback()`: Processes feedback from the Artemis system, allowing the module to learn and adapt its feedback generation process.
- `suggestFeedback()`: The core method that generates and returns feedback suggestions for a given submission.
- `provideConfigSchema()` (Optional): Delivers a JSON schema for dynamic module configuration.
- `evaluateFeedback()` (Optional): Provides metrics for evaluating the generated feedback.

## 6.2 Athena: Text LLM Module

The Text LLM Module aims to automate the feedback generation process for text exercises in the Artemis platform. Its operation pipeline is illustrated by the UML Activity Diagram shown in Figure 6.2. We detail the pipeline's steps in Section 6.2.1, and follow up with the process of generating evaluations in Section 6.2.2 for automatic evaluation using the LLM-as-a-judge methodology [ZCS<sup>+</sup>23].

### 6.2.1 Generating Feedback Suggestions

The Text LLM Module employs a basic three-step workflow when interacting with LLMs: Format, Predict, and Parse, as previously described in Section 2.3.3. First, a prompt is formatted to provide the LLM with a rich context for generating relevant feedback suggestions. The elements that can be incorporated into the prompt, next to the sentence numbered submission, include the problem statement, example solution, grading instructions, maximum points, and bonus points. However, it is essential to note that

due to the LLM’s token limit, not all elements can be included at all times. Specifically, the total token count, including the expected output, should not exceed 4096 tokens in the case of GPT-3.5-Turbo. For reference, the median submission length for text exercises in Artemis is 726 characters ( $\sim 180$  tokens) with the 95th percentile at 2170 characters ( $\sim 542$  tokens, in common English text), as detailed in Figure A.5.

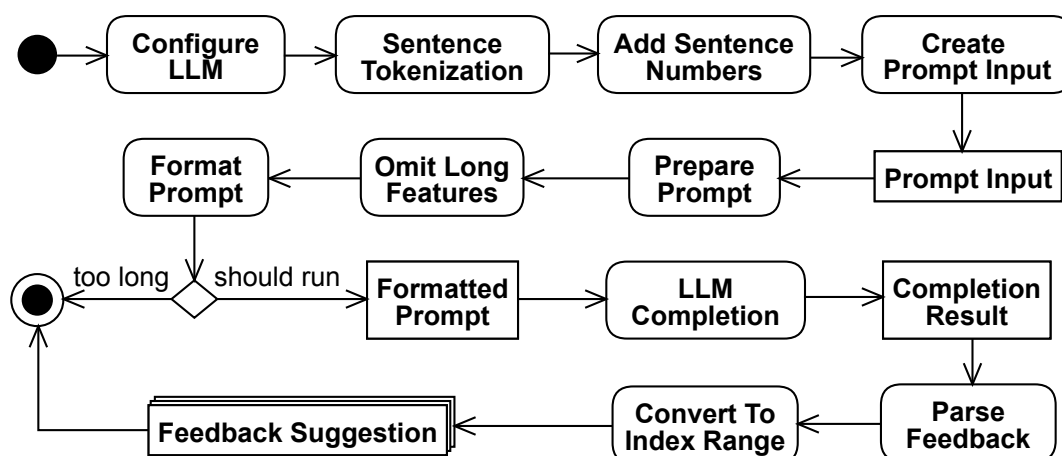


Figure 6.2: UML Activity Diagram for the Text LLM Module.

To address the token limit constraint, the module adopts a strategy where it first omits the problem statement, followed by the example solution, and finally the grading instructions until the total token count is within acceptable limits. Once the prompt, as seen in Figure 6.3, is successfully formatted and its length verified, it is passed to the LLM, which then returns a plain text completion.

The completion is subsequently parsed into feedback suggestions. This parsing process is either done through function calling if supported by the LLM, or through JSON parsing. The resulting feedback suggestions reference specific sentence ranges within the student’s submission. Next, we convert these sentence numbers to index ranges within the submission text characters, as required by Artemis.

The entire sequence of actions for generating feedback suggestions is depicted in Figure 6.2. This design ensures that the model gets as much context as possible to generate relevant feedback suggestions, while also adhering to the LLM’s token limit.

## System Message

```
1 You are an AI tutor for text assessment at a prestigious
  university .
2
3 # Task
4 Create graded feedback suggestions for a student's text
  submission that a human tutor would accept. Meaning, the
  feedback you provide should be applicable to the submission
  with little to no modification.
5
6 # Style
7 1. Constructive , 2. Specific , 3. Balanced , 4. Clear and
  Concise , 5. Actionable , 6. Educational , 7. Contextual
8
9 # Problem statement
10 {problem_statement}
11
12 # Example solution
13 {example_solution}
14
15 # Grading instructions
16 {grading_instructions}
17 Max points: {max_points} , bonus points: {bonus_points}
```

## Human Message

```
1 Student's submission to grade (with sentence numbers <
  number>: <sentence>):
2 """
3 {submission}
4 """
```

**Figure 6.3:** Prompt for Generating Feedback Suggestions for Text Exercises. The highlighted sections are placeholders for the respective elements.

## 6.2.2 Generating Evaluations

To facilitate an automatic evaluation of the feedback suggestions, in Chapter 7, the Text LLM Module integrates the LLM-as-a-judge methodology, a technique adapted from recent research [ZCS<sup>+</sup>23]. The methodology leverages a strong LLM, such as GPT-4, to approximate human preferences.

## System Message

```

1 You are now an evaluator for feedback accuracy generated by
  a machine-learning system.
2
3 # Task
4 Your task is to estimate if a human tutor would accept or
  reject the feedback suggestion and how much modification is
  needed to make the feedback useful.
5
6 # Score Criteria
7 Accept feedback that is useful to the tutor, meaning that
  it can be applied to the submission with minor or no
  modification. Our goal is to reduce the workload of tutors
  and reduce their cognitive load. Reject feedback that is
  not useful and would burden the tutor.
8
9 Put the focus on the description of the feedback, the title
  is optional. The 'line_start' and 'line_end' should make
  sense with respect to the submission but do not need to be
  exact. Credits should make sense with respect to the
  feedback and the submission but also do not need to be
  exact.
10
11 # Submission (with sentence numbers <number>: <sentence>):
12 {submission}
13
14 # Example (Human) Feedback:
15 {true_feedbacks}

```

## Human Message

```

1 ### Model Output:
2 {predicted_feedbacks}

```

**Figure 6.4:** Prompt for Generating Evaluations for Text Exercises. The highlighted sections are placeholders for the respective elements.

We use this method to assess the applicability, *i.e.* accuracy, of the generated feedback. In this thesis, we define accuracy as the likelihood that a human tutor would accept a feedback suggestion with minimal or no modifications. This evaluation mechanism is critical for validating the efficacy of the system and iteratively refining its performance.

The evaluation process follows the established format, predict, and parse workflow. A specialized prompt is constructed, which includes the student's submission annotated with sentence numbers, the original feedback provided by human tutors as a gold standard, and the machine-generated feedback suggestions. The prompt used for this task is detailed in Figure 6.4. Upon receiving the formatted prompt, the LLM produces a completion that categorizes each feedback suggestion based on its acceptability and the degree of modification required for its utility.

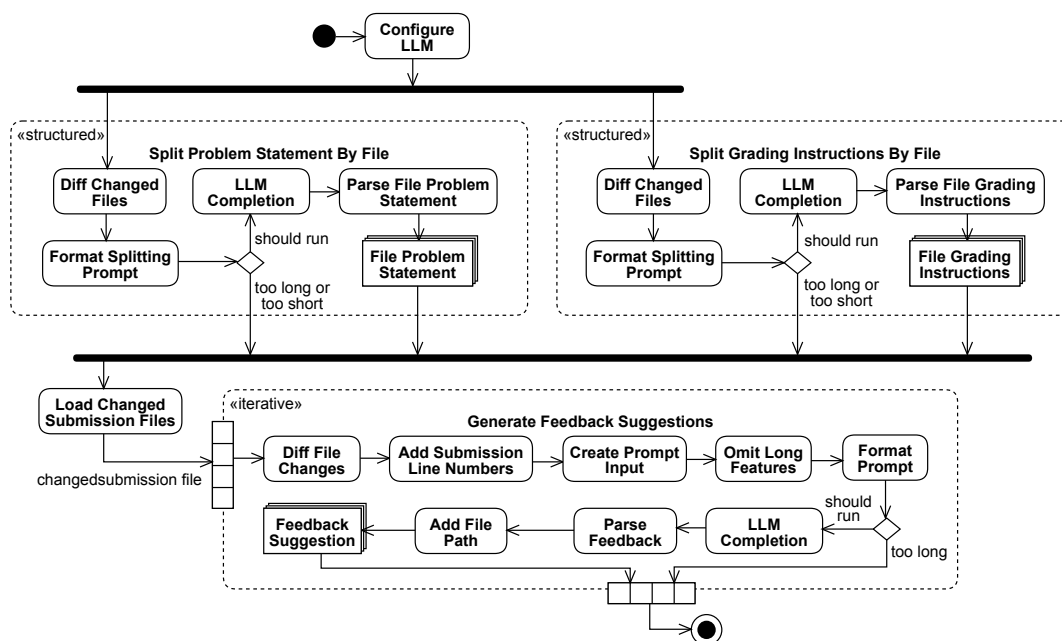
The output from the LLM is then parsed into a structured format. This format contains a list of feedback suggestions, each tagged with an identifier, an *accepted* or *rejected* label, and the level of modifications needed — categorized as *no*, *minor*, or *major*. Additionally, the output also includes the reason for the evaluation as reflection step. Overall, this evaluation output serves as an invaluable resource for both system diagnostics and future research, offering insights into the specific strengths and weaknesses of the automated feedback generation mechanism.

### 6.3 Athena: Programming LLM Module

The Programming LLM Module navigates the complex landscape of programming exercises, which are inherently more complex than text exercises. This complexity stems from multifaceted problem statements, extended grading instructions, and multiple submission files that form a complete code repository. The module's design adapts the approach from text exercises to accommodate these complexities. This detailed operation pipeline is illustrated as UML Activity Diagram in Figure 6.5.

To distill relevant information for feedback generation, the Programming LLM Module introduces two critical steps: *Split Problem Statement By File* and *Split Grading Instructions By File*. These steps utilize a LLM to restructure the general problem statement and grading instructions into file-specific variants. The restructured versions serve as concise context for the feedback generation step, focusing on the files altered by students. This is achieved through specialized prompts, as outlined in Figures 6.6 and 6.7, which guide the LLM to produce file-specific problem statements and grading instructions





**Figure 6.5:** UML Activity Diagram for the Programming LLM Module.

based on the repository diffs<sup>2</sup> between the template, solution, and student submission.

For the actual feedback generation step, *Generate Feedback Suggestions*, the module operates on a per-file basis, ignoring context of other files. To manage a large number of changed files, the module adopts a heuristic approach to prioritize files more likely to require feedback. This is determined by analyzing repository diffs, the programming language used, and file extensions. The number of files to generate feedback for is limited, arbitrarily, to 25. Additionally, if the general problem statement and grading instructions are sufficiently short, less than 250 tokens, the module includes them in their entirety to preserve context.

The prompt for generating file-specific feedback, as shown in Figure 6.8, integrates multiple elements: the file-specific problem statement, grading instructions, repository diffs, and the student’s submission file annotated with line numbers to reference. This prompt equips the LLM with a rich context for generating feedback suggestions for programming exercises.

In summary, the Programming LLM Module’s design ensures a context-aware feedback generation process per-file. The module adapts to the complexities of programming exercises while aiming to retain the critical elements within the input context.

<sup>2</sup><https://git-scm.com/docs/git-diff>

System Message

```
1 You are an AI tutor for programming assessment at a
  prestigious university.
2
3 # Task
4 Restructure the grading instructions by student changed
  file to show relevant information for each file to the
  tutor. Make it as easy as possible for the tutor to grade
  the assignment when looking at the changed file. Some
  instructions may be relevant for multiple files.
```

Human Message

```
1 Grading instructions:
2 {grading_instructions}
3
4 Changed files from template to sample solution:
5 {changed_files_from_template_to_solution}
6
7 Changed files from template to student submission (Pick
  from this list , very important!):
8 {changed_files_from_template_to_submission}
9
10 Grading instructions by file:
```

**Figure 6.6:** Prompt for Splitting the Grading Instructions by File for Programming Exercises. The highlighted sections are placeholders for the respective elements.

System Message

```
1 You are an AI tutor for programming assessment at a
2 prestigious university.
3 # Task
4 Restructure the problem statement by student changed file
  to show relevant information for each file to the tutor.
  Make it as easy as possible for the tutor to grade the
  assignment when looking at the changed file. Some parts of
  the problem statement may be relevant for multiple files.
```

Human Message

```
1 Problem statement:
2 {problem_statement}
3
4 Changed files from template to sample solution:
5 {changed_files_from_template_to_solution}
6
7 Changed files from template to student submission (Pick
  from this list , very important!):
8 {changed_files_from_template_to_submission}
9
10 Problem statement by file:
```

**Figure 6.7:** Prompt for Splitting the Problem Statement by File for Programming Exercises. The highlighted sections are placeholders for the respective elements.

## System Message

```
1 You are an AI tutor for programming assessment at a
  prestigious university.
2
3 # Task
4 Create graded feedback suggestions for a student's
  programming submission that a human tutor would accept.
  Meaning, the feedback you provide should be applicable to
  the submission with little to no modification.
5
6 # Style
7 1. Constructive , 2. Specific , 3. Balanced , 4. Clear and
  Concise , 5. Actionable , 6. Educational , 7. Contextual
8
9 # Problem statement
10 {problem_statement}
11
12 # Grading instructions
13 {grading_instructions}
14 Max points: {max_points}, bonus points: {bonus_points} (whole
  assessment, not just this file)
15
16 # Diff between solution (deletions) and student's
  submission (additions):
17 {solution_to_submission_diff}
18
19 # Diff between template (deletions) and student's
  submission (additions):
20 {template_to_submission_diff}
```

## Human Message

```
1 Student's submission file to grade (with line numbers <
  number>: <line >):
2 """
3 {submission_file}
4 """
```

**Figure 6.8:** Prompt for Generating Feedback Suggestions for Programming Exercises. The highlighted sections are placeholders for the respective elements.

## 6.4 Research and Development Environment

The Research and Development (R&D) Environment, or also R&D Playground, serves as an instrumental platform for rapid experimentation and evaluation in the realm of automated feedback generation. This environment serves as a multi-faceted tool, comprising three main components: module configuration, module requests, and evaluation management.

### 6.4.1 Module Configuration

The R&D Playground introduces a dynamic configuration mechanism for feedback generation modules. Each module can provide a JSON schema that outlines its configuration parameters, which the Playground renders into a user interface using `react-jsonschema-form`<sup>3</sup>, an example for the Text LLM Module is illustrated in Figure 6.9. This configuration is incorporated into the request header when the Athena Assessment Service is invoked, enabling dynamic adjustments to module behavior. This feature paves the way for quick, iterative development and experimentation, an essential aspect for improving automated feedback processes.

### 6.4.2 Module Requests

The Playground also features a request mode, providing a convenient way for testing individual module requests. Users can select between two types of datasets: example data for quick setup and evaluation data for a more realistic setup. The interface displays both the request and response with all the necessary detail views, as for example seen in Figure 6.10, offering a simple way to understand the system's inner workings. This mechanism accelerates the debugging and testing processes, streamlining the development pipeline.

### 6.4.3 Evaluation Management

Transitioning from module testing to systematic evaluation is seamless, due to the Playground's evaluation mode. Here, researchers can define and conduct experiments on different module configurations in parallel.

#### Define Experiment

Researchers have the liberty to design experiments by selecting the exercise type, exercise and submission data for training and evaluation, and whether

---

<sup>3</sup><https://github.com/rjsf-team/react-jsonschema-form>

or not to enable automatic evaluation, as seen in Figure 6.11. The exercise configuration can be exported and imported, offering a persistent research setup. Once the settings are locked in, the stage is set for the experiment.

### **Configure Modules**

Prior to conducting an experiment, researchers configure the modules involved. These configurations are experiment-agnostic, allowing for their reuse across different experimental setups. The Playground's UI for configuring the modules can be seen in Figure 6.12.

### **Conduct Experiment**

The actual experiment takes place within a multi-column interface. It displays the exercise details, tutor feedback, and module results, facilitating a side-by-side comparison. The UI of the exercise details and tutor feedback is shown in Figure 6.13. For text exercises, the feedback suggestions are displayed in the text submission itself, as seen in Figure 6.14. Similarly, for programming exercises, an integrated code editor enables users to navigate through inlined feedback within code files, as seen in Figure 6.15. The environment supports the manual rating of feedback suggestions and the export of experiment data for further analysis.

## 6.4. RESEARCH AND DEVELOPMENT ENVIRONMENT

Enable debug mode.

Debug

### Approach

This approach uses a LLM with a single prompt to generate feedback in a single step.

**Max Input Tokens**

Maximum number of tokens in the input prompt.

**Model**

OpenAI

### OpenAI

OpenAI LLM configuration.

**OpenAIModel**

The name of the model to use.

**Max Tokens**

The maximum number of **tokens** to generate in the chat completion.

The total length of input tokens and generated tokens is limited by the model's context length. [Example Python code](#) for counting tokens.

**Temperature**

What sampling temperature to use, between 0 and 2. Higher values like 0.8 will make the output more random, while lower values like 0.2 will make it more focused and deterministic.

We generally recommend altering this or `'top_p'` but not both.

**Top P**

An alternative to sampling with temperature, called nucleus sampling, where the model considers the results of the tokens with top\_p probability mass. So 0.1 means only the tokens comprising the top 10% probability mass are considered.

We generally recommend altering this or `'temperature'` but not both.

**Presence Penalty**

Number between -2.0 and 2.0. Positive values penalize new tokens based on whether they appear in the text so far, increasing the model's likelihood to talk about new topics.

[See more information about frequency and presence penalties.](#)

**Frequency Penalty**

Number between -2.0 and 2.0. Positive values penalize new tokens based on their existing frequency in the text so far, decreasing the model's likelihood to repeat the same line verbatim.

[See more information about frequency and presence penalties.](#)

### Generate Suggestions Prompt

Features available: `{problem_statement}`, `{example_solution}`, `{grading_instructions}`, `{max_points}`, `{bonus_points}`, `{submission}`

Note: `{problem_statement}`, `{example_solution}`, or `{grading_instructions}` might be omitted if the input is too long.

**System Message**

Message for priming AI behavior and instructing it what to do.

```
1 You are an AI tutor at a prestigious university tasked with grading and providing high quality feedback to text assignments.
2
3 VERY IMPORTANT: Effective feedback for text assignments should be:
4 1. Constructive, 2. Specific, 3. Balanced, 4. Clear and Concise, 5. Actionable, 6. Educational, 7. Contextual
5
6 Ignore all remarks about plagiarism.
```

**Human Message**

Message from a human. The input on which the AI is supposed to act.

```
1 Problem statement:
2 {problem_statement}
3
4 Example solution:
5 {example_solution}
6
7 Grading instructions:
8 {grading_instructions}
9 Max points: {max_points}, bonus points: {bonus_points}
10
11 Student's submission to grade (with sentence numbers <number>: <sentence>):
12 {submission}
13
```

Figure 6.9: Module Configuration UI of the R&D Playground for the Text LLM Module. (Screenshot)

## CHAPTER 6. OBJECT DESIGN

### Request Feedback Suggestions from Athena

Request a list of feedback suggestions from Athena for the selected submission. The LMS would usually call this when a tutor starts grading a submission. You should get a list of all submissions that are not graded yet. The matching module for the exercise will receive the request at the function annotated with @Feedback\_provider.

**Exercise**  
4160 text: H04E01 Coupling and Cohesion (SS21)

**Submission**  
796213 - Coupling evaluates dependency between classes within one system. - Cohesion e

▼ Exercise Detail  
3 Points and 0 Bonus Points

▼ Problem Statement

Markdown Plain Text

Explain the difference between coupling and cohesion. Why are these terms important in system design? Provide an example.

Requirement: Use your own words and your own example and do not just copy the lecture slides nor internet sources! Limit yourself to around 250 words

► Grading Instructions  
► Example Solution

▼ Submission

```
1 - Coupling evaluates dependency between classes within one system.
2
3 - Cohesion evaluates dependency between multiple subsystems.
4
5 They are important terms since coupling and cohesion are goals for a good system design that is less complex while changes are feasible.
6
7 For instance, an insurance service system has one subsystem for client portals and one subsystem for internal operations. Within each subsystem, its components are closely connected (high cohesion). Whereas, both subsystems are loosely connected (low coupling). This allows individual upgrades possible for either system with minimal effect to the other. Within each subsystem, functions work smoothly and have appropriate triggers.
```

**Response**

Used module:  
module\_text\_lim

Status code:  
200

▼ Meta:  
{}

▼ Data:

```
[
  {
    "id": 1694178593052080,
    "title": "Assessment of the Difference between Coupling/Cohesion",
    "description": "You correctly explained the difference between coupling and cohesion, well done!",
    "credits": 1,
    "grading_instruction_id": null,
    "meta": {},
    "exercise_id": 4160,
    "submission_id": 796213,
    "index_start": 0,
    "index_end": 128,
    "type": "text",
    "isSuggestion": true
  },
  {
    "id": 1694178593052081,
    "title": "Assessment of the Explanation why Coupling/Cohesion are important",
    "description": "You correctly explained why coupling and cohesion are important, well done!",
    "credits": 1,
    "grading_instruction_id": null,
    "meta": {},
    "exercise_id": 4160,
    "submission_id": 796213,
    "index_start": 130,
    "index_end": 266,
    "type": "text",
    "isSuggestion": true
  },
  {
    "id": 1694178593052082,
    "title": "Assessment of the Example",
    "description": "Great example, well done!",
    "credits": 1,
    "grading_instruction_id": null,
    "meta": {},
    "exercise_id": 4160,
    "submission_id": 796213,
    "index_start": 268,
    "index_end": 785,
    "type": "text",
    "isSuggestion": true
  }
]
```

▼ Submission with Feedback Suggestions

```
1 - Coupling evaluates dependency between classes within one system.
2
3 - Cohesion evaluates dependency between multiple subsystems.
References 0-128
1P Assessment of the Difference between Coupling/Cohesion
You correctly explained the difference between coupling and cohesion, well done! Suggestion
4
5 They are important terms since coupling and cohesion are goals for a good system design that is less complex while changes are feasible.
References 130-266
1P Assessment of the Explanation why Coupling/Cohesion are important
You correctly explained why coupling and cohesion are important, well done! Suggestion
6
7 For instance, an insurance service system has one subsystem for client portals and one subsystem for internal operations. Within each subsystem, its components are closely connected (high cohesion). Whereas, both subsystems are loosely connected (low coupling). This allows individual upgrades possible for either system with minimal effect to the other. Within each subsystem, functions work smoothly and have appropriate triggers.
References 268-705
1P Assessment of the Example
Great example, well done! Suggestion
```

Request feedback suggestions

Figure 6.10: Module Requests UI of the R&D Playground for Feedback Suggestions from Athena. (Screenshot)



## 6.4. RESEARCH AND DEVELOPMENT ENVIRONMENT

**Define Experiment** Export Import

**Execution Mode**  
Batch mode (all submissions at once)

Run automatic evaluation (if available, can be costly)

**Exercise Type**  
text

**Exercise**  
4160 text: H04E01 Coupling and Cohesion (SS21)

▶ Exercise Detail

**Submissions**  
 Enable training data  
Select  random  next 25 submissions

Move 25 to Training →    Move 25 to Evaluation →    ← Move 25 to Excluded    Move 25 to Evaluation →    ← Move 25 to Excluded    ← Move 25 to Training

Excluded (1187 Submissions)	Training (50 Submissions)	Evaluation (25 Submissions)
Submissions that are not used in the experiment. ▶ 1187 Submissions	Sent for training before running evaluation. ▶ 50 Submissions	Run the experiment on the evaluation submissions. ▶ 25 Submissions

Define Experiment

Figure 6.11: Define Experiment UI of the R&D Playground. (Screenshot)

**Configure Modules** Export Import Prev Next

**Configuration 1** Valid ← →  
GPT-4 Duplicate Delete

**Configuration 2** Valid ← →  
LLaMA-13b Duplicate Delete

**Approach**  
This approach uses a LLM with a single prompt to generate feedback in a single step.

**Max Input Tokens**  
Maximum number of tokens in the input prompt.  
3000

**Model**  
OpenAI

**OpenAI**  
OpenAI LLM configuration.  
**OpenAIModel**  
The name of the model to use.  
azure\_openai\_gpt-4

**Human Message**  
Message from a human. The input on which the AI is supposed to act.

```
1 You are an AI tutor at a prestigious university tasked with grading
2 and providing high quality feedback to text assignments.
3
4 VERY IMPORTANT: Effective feedback for text assignments should be:
5 1. Constructive, 2. Specific, 3. Balanced, 4. Clear and Concise, 5.
6 Actionable, 6. Educational, 7. Contextual
7
8 Ignore all remarks about plagiarism.
9
10
11 Grading instructions:
12 {grading_instructions}
13 Max points: {max_points}, bonus points: {bonus_points}
14
15 Student's submission to grade (with sentence numbers <number>:
16 <sentence>):
17 {submission}
```

Define Configurations

Figure 6.12: Configure Modules UI of the R&D Playground. (Screenshot)

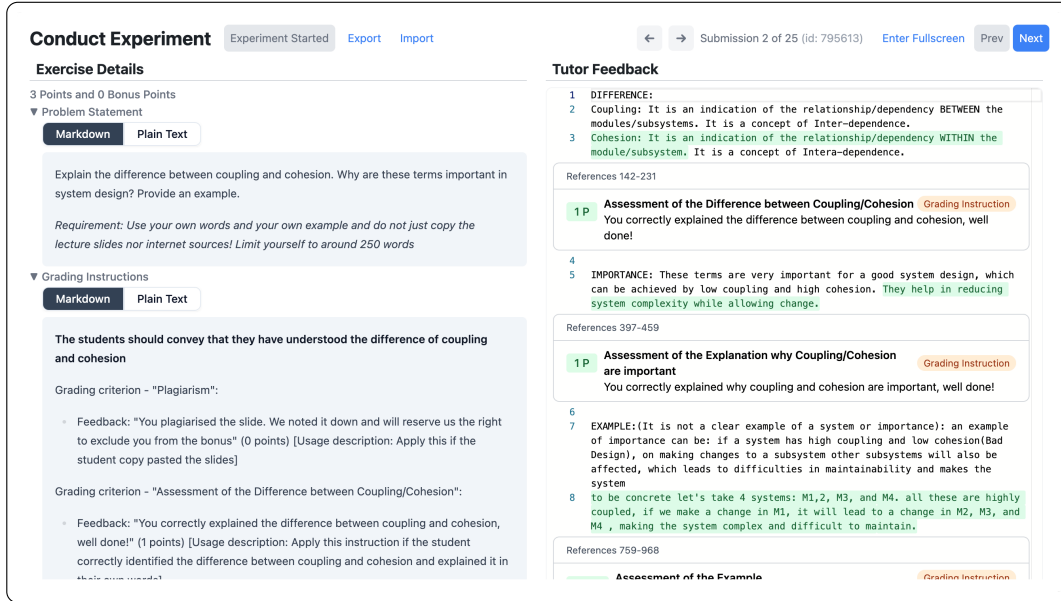


Figure 6.13: Conduct Experiment Exercise Details and Tutor Feedback UI of the R&D Playground. (Screenshot)

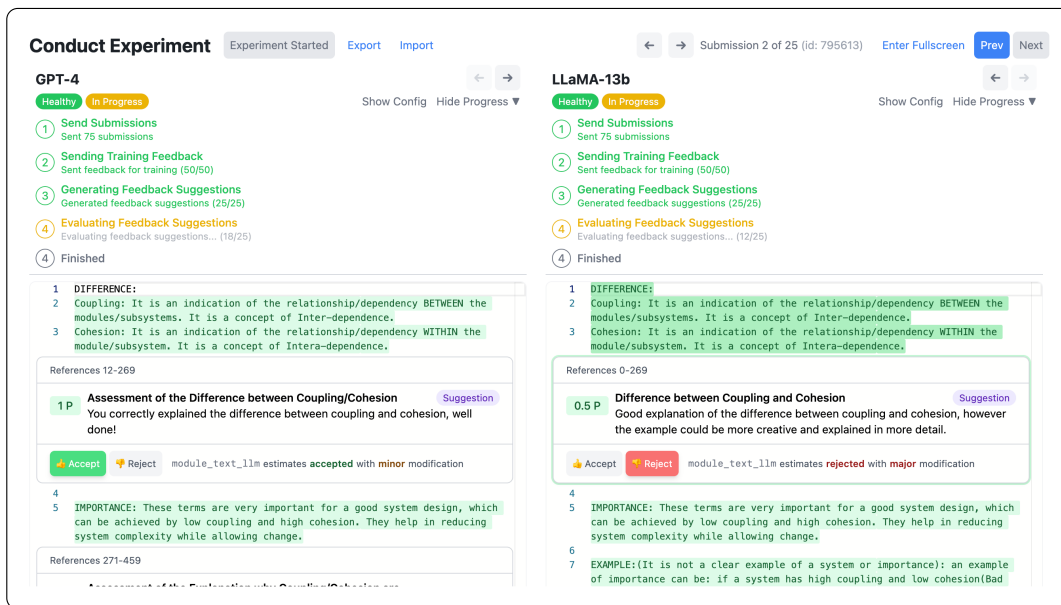
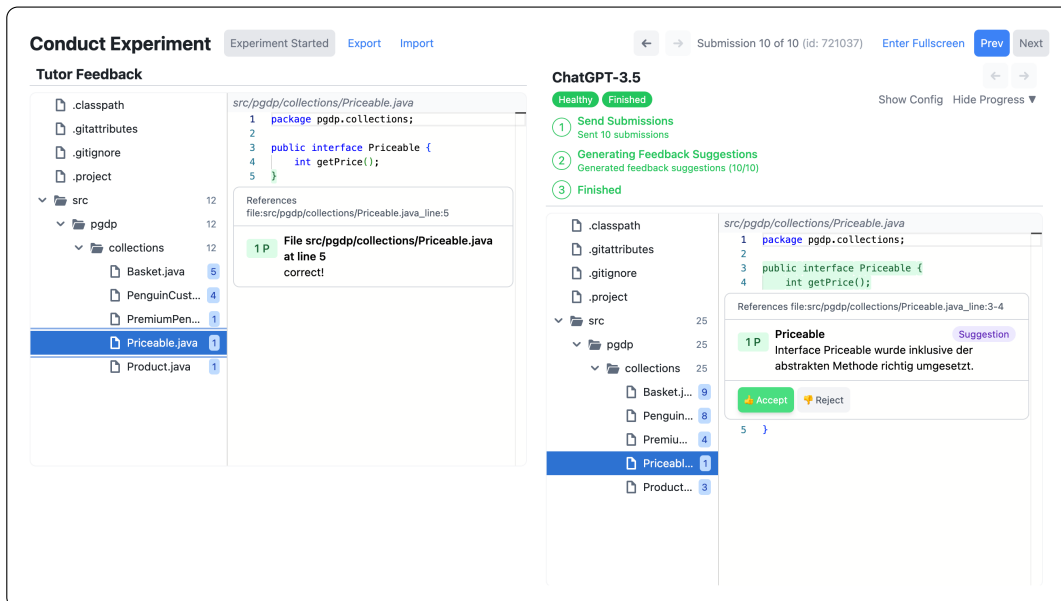


Figure 6.14: Conduct Experiment Module Comparison UI of the R&D Playground for a Text Module. (Screenshot)



**Figure 6.15:** Conduct Experiment Module Comparison UI of the R&D Playground for a Programming Module. (Screenshot)

# Chapter 7

## Evaluation

This chapter focuses on the assessment of the LLM-based feedback generation system integrated into the Artemis learning platform. The evaluation aims to quantify and understand the system’s performance across text and programming exercises.

### 7.1 Design

Evaluating LLMs for specialized tasks, such as automated feedback generation in an educational context, is a non-trivial endeavor. These challenges stem from multiple metrics that require consideration: quality, accuracy, and coverage of the feedback suggestions. To navigate these complexities, we employ a mixed-method approach.

For text exercises, we apply the *LLM-as-a-judge* methodology [ZCS<sup>+</sup>23]. This approach leverages GPT-4’s capabilities to approximate human judgment, thereby offering a scalable and explainable method for quality and accuracy assessment. To cross-verify the results, we conduct a manual evaluation on a single exercise.

In contrast, for programming exercises, the LLM-as-a-judge method does not easily translate due to the complexity of coding tasks. As a result, manual evaluation becomes indispensable for these exercises. Due to time-constraints, we did not conduct manual evaluation for programming exercises. However, we conduct a quantitative analysis of the generated feedback suggestions, focusing on metrics such as token usage, generation time, and the number of generated feedback suggestions.

Experiments are conducted in a research and development environment, as previously detailed in Section 6.4. Metrics such as token usage, generation time, the number of generated feedback suggestions, and their accuracy are

rigorously tracked. The emphasis on accuracy aims to measure how readily the LLM-generated feedback can be applied to student submissions without extensive modifications.

### 7.1.1 Data Selection

For empirical validation, we selected a subset of previously conducted text and programming exercises from the Technical University of Munich’s instance of the Artemis Learning Management System. The data for this study was acquired through an anonymized database dump provided by the Artemis team, snapshot dated March 29, 2023.

#### Text Exercises

We select data from Artemis courses with the lowest rates of empty feedback for text exercises, as detailed in Table 7.1. The reason for the empty feedback is unclear, but it is likely due to the deletion of structured grading instructions associated with the feedback.

ID	Course	Semester	Feedback	
			Total	Empty (%)
29	Introduction to Software Engineering	SS19	92411	45.37
30	Project Organization and Management	SS19	5432	45.43
41	Patterns in Software Engineering	WS19/20	131	76.34
48	Introduction to Software Engineering	SS20	77629	18.17
67	Networks for Monetary Transactions	SS20	2754	97.02
88	Patterns in Software Engineering	WS20/21	6177	19.48
121	Introduction to Software Engineering	SS21	123919	<b>0.28</b>
139	Advanced Topics of Software Testing	SS21	5766	<b>1.23</b>
167	Functional Programming and Verification	WS21/22	9805	<b>0.44</b>
169	Introduction to Software Engineering	SS22	45546	<b>1.03</b>

**Table 7.1:** Empty Feedback Rates in Artemis Courses for Text Exercises as of March 29, 2023. The first block details courses used by Bernius [Ber22] for validation. The second block lists additional suggested courses for future evaluation. A gray background indicates a low empty feedback rate.

For our evaluation, we focus on the “Introduction to Software Engineering” course of the summer semester 2021. It has the lowest empty feedback rate of all courses, making it an ideal candidate for comparing the LLM-generated feedback against actual tutor feedback. Within this course, we

select five exercises for evaluation, as depicted in Table 7.2. Note that, for simplicity and due to time constraints, the main focus of this evaluation is on the “H04E01 Coupling and Cohesion” exercise. The remaining exercises are automatically evaluated to provide insight into a broader perspective on the LLM’s performance across different exercises within the selected software engineering course.

ID	Exercise Title	Submissions
4082	H01E02 Change in Software Development	1361
4101	H03E03 Bumpers Visionary Scenario	1284
4160	H04E01 Coupling and Cohesion	1260
4162	H04E03 Design Goal Trade-offs	1222
4238	H06E04 Inheritance vs. Delegation	1135

**Table 7.2:** Text Exercises Selected for Evaluation from “Introduction to Software Engineering” of summer semester 2021. The gray background indicates our selection for manual evaluation.

### Programming Exercises

The selection of programming exercises for evaluation was guided by multiple criteria, such as the programming language used, the volume of manual feedback provided, and a high incidence of referenced feedback. The exercises chosen for this study are detailed in Table 7.3.

Due to time constraints, this research focuses on the “Shoppende Pinguine” exercise, which is part of the course “Nachklausur zum Praktikum: Grundlagen der Programmierung WS20/21”. It is worth noting that this exercise has an atypically high rate of referenced feedback; almost all feedback is referenced, diverging from the Artemis average of 11.5% as shown in Figure A.12. Written in Java, the exercise aligns with the most commonly used programming language on Artemis, as indicated in Figure A.9. This focus is further justified by the manageable number of submissions for the exercise, since the current generation approach is rather costly.

The remaining exercises listed in Table 7.3 serve as promising candidates for future research and subsequent evaluation. They are selected based on the criteria mentioned above, but due to time-constraints, we did not evaluate them in this study.

ID	Exercise Title	Language	Exam	Submissions
3184	Objektorientierung mit Verkehrsmitteln	Java	Yes	755
3185	Objektorientierung mit Transportmitteln	Java	Yes	740
3908	Aktor-Threads	Java	Yes	81
3913	Shoppende Pinguine	Java	Yes	97
3914	Konsumierende Eisbaeren	Java	Yes	102
2104	H03 - Grafikspeicher	C	No	706
2111	H02 - Festkommarechnung	C	No	835
2610	P01 - Raycasting mit Festkommazahlen	Assembly	No	806
3782	ASM Backpack	Assembly	Yes	597
3187	P02 - Taschenrechner	VHDL	No	693
3781	Ampelsteuerung	VHDL	Yes	621
6344	Textline Editor	OCaml	Yes	214
6433	Interpreters	OCaml	Yes	93
3693	Shapes	Haskell	Yes	270
3942	Expressions	Haskell	Yes	213
4864	Free Shipment	Python	Yes	113
4896	Free Delivery	Python	Yes	113

**Table 7.3:** Potential Programming Exercises Selected for Evaluation. The gray background indicates our selection for evaluation due to time-constraints.

## 7.1.2 Experiments

The evaluation employs a rigorous experimental methodology designed to address specific hypotheses about the LLM-based feedback generation system’s performance. We conducted a combination of four experiments for text exercises (T1-4) and one experiment for programming exercises (P1).

### Text Exercises

For text exercises, the first experiment (T1) aims to optimize the prompt for the LLM-based system. The subsequent experiments (T2-4) focus on model comparison, method validation, and exercise comparison, respectively.

**Experiment T1: Prompt Optimization.** This experiment is dedicated to optimizing the prompts that guide the LLM in generating feedback for text exercises. Specifically, we focus on the “H04E01 Coupling and Cohesion” exercise for this purpose. We utilize two versions of the exercise: one containing the original structured grading instructions (SGI) and another with simplified

grading instructions. The simplification process involves GPT-4 summarizing the original grading instructions. This step is crucial as it prevents the LLM from merely copying the detailed grading instructions, thus enabling a more nuanced evaluation of prompt performance.

Five prompt variants are created for this experiment. The first variant is the original prompt used during the system’s development phase. The second variant is meticulously crafted, adhering to OpenAI’s best practices guidelines<sup>1</sup>; this variant is further illustrated in Figure 6.3. The remaining three variants are derivatives of the second prompt, integrating subtle modifications in context cues.

We perform the evaluation with GPT-3.5-Turbo for each prompt on a fixed small sample of 25 randomly selected submissions of the “H04E01 Coupling and Cohesion” exercise for both exercise variants.

**Experiment T2: Model Comparison.** Building upon the optimized prompt from Experiment T1, the second experiment seeks to compare the performance of different LLMs, namely GPT-3.5-Turbo, GPT-4, and LLaMA-13b-Chat in generating feedback for the “H04E01 Coupling and Cohesion” exercise. Note that GPT-3.5-Turbo and GPT-4 use structured output parsing through OpenAI’s function calling, and LLaMA-13b-Chat uses an unoptimized custom parsing approach, injecting formatting instructions into the prompt. The OpenAI models are deployed via Azure, whereas LLaMA-13b-Chat is deployed through the cloud computing platform Replicate<sup>2</sup>.

We benchmark using a fixed, randomly selected sample of 100 submissions due to the high computational cost of generating feedback suggestions and automatically evaluating them with GPT-4 as a judge

**Experiment T3: LLM-as-a-Judge Evaluation.** This experiment is geared towards cross-validating the estimated accuracy by the LLM-as-a-judge methodology with manual evaluation. We use the automatic evaluation results from the previous experiment (T2) for the GPT-3.5-Turbo model and have one person that is also not an expert in software engineering, manually rate the 303 feedback suggestions similarly how a tutor would.

The person is given each submission, with the feedback suggestions and the previous tutor’s feedback. The instructions for the acceptance and rejection of the feedback suggestions are similar to the prompt used for the LLM-as-a-judge, as detailed in Figure 6.4. The non-expert is asked to rate

---

<sup>1</sup><https://platform.openai.com/docs/guides/gpt-best-practices>

<sup>2</sup><https://replicate.com>



the feedback suggestions as either “Accept” or “Reject” based on the instructions.

**Experiment T4: Exercise Comparison.** This experiment evaluates the LLM-generated feedback across multiple text exercises, as listed in Table 7.2. For “H04E01 Coupling and Cohesion” we use the previous results of experiment T2, and for the remaining exercises we select a random sample of 100 submission within each exercise. We are using GPT-3.5-Turbo for this experiment, since it seems to be the most cost-effective model for the feedback generation task when dealing with thousands of submissions at the moment.

### Programming Exercises

Due to time-, cost-, and usability-constraints we only conduct one experiment for model comparison.

**Experiment P1: Model Comparison.** In this experiment, we focus on the “Shoppende Pinguine” exercise, comparing GPT-3.5-Turbo and GPT-4. Taking what we have learned though experiment T1, we crafted the prompts for this experiment similarly to Figure 6.3. For splitting the problem statement and grading instructions by file, we use the prompts as seen in Figure 6.7 and Figure 6.6, respectively. For the generation process, we use the prompt detailed in Figure 6.8.

For benchmarking, we randomly select 48 submissions from the “Shoppende Pinguine” exercise. Due to the complexity of programming tasks, the metrics monitored are limited to token usage, generation time, and the number of generated feedback suggestions. We do not evaluate accuracy and quality.

## 7.2 Objectives

The objectives of this evaluation are designed to rigorously assess the performance, scalability, and adaptability of the LLM-based feedback generation system in the educational context of Artemis. These objectives are aligned with our specific experiments for both text and programming exercises, detailed previously in Section 7.1.2.

### 7.2.1 Text Exercises

For text exercises, the first objective aims to validate the research and development environment’s capabilities for systematic iteration on feedback generation configurations. This is tied to Experiment T1 on Prompt Optimization. The environment should demonstrate its effectiveness in identifying an optimal prompt that enhances both the quality and accuracy of the generated feedback.

The second objective, corresponding to Experiment T2 on Model Comparison, aims to identify the most efficient, cost-effective, and accurate LLM for deployment in a large-scale educational setting. The focus here is to provide actionable insights that balance generation time, computational costs, and high feedback accuracy.

Experiment T3 aims to validate the LLM-as-a-judge methodology within the research and development environment. It aims to substantiate that LLMs can approximate human judgment for automatic evaluation, thereby offering a scalable method for assessing the quality and accuracy of generated feedback suggestions.

The final objective for text exercises associated with Experiment T4 is to establish that the LLM-based system can generalize across different types of text exercises. The system should demonstrate adaptability and flexibility in accommodating diverse educational content and grading criteria.

### 7.2.2 Programming Exercises

The primary objective for programming exercises is centered around a rough quantitative feasibility assessment through Experiment P1 on Model Comparison. This experiment deploys two LLMs, GPT-3.5-Turbo and the more computationally demanding GPT-4, to investigate whether automated feedback generation for programming exercises can be viable. Specifically, the focus is on optimizing a balance between computational cost and the volume of feedback suggestions while maintaining a minimum threshold of educational value in the generated feedback. This evaluation serves as an initial, foundational step to inform future research and evaluations on the scalability and effectiveness of automated feedback in complex programming tasks.

## 7.3 Results

The subsequent sections present the empirical outcomes of the evaluation, categorizing the results into text and programming exercises. These results,

devoid of interpretation at this stage, set the groundwork for subsequent discussions of findings and limitations.

### 7.3.1 Text Exercises

**Experiment T1: Prompt Optimization.** The primary experiment focusing on prompt optimization yielded varying performance metrics across the five different prompts tested. The most compelling result is observed for Prompt 2 with Structured Grading Instructions (SGI), which demonstrated the highest estimated accuracy of 83.3%, as seen in Table 7.4. Interestingly, Prompt 4 without SGI scored an even higher estimated accuracy of 89.4%. In total, three out of five prompts achieved a higher estimated accuracy with summarized SGIs than with the original detailed SGIs while producing more feedback suggestions with fewer tokens. Despite these results, we choose to use Prompt 2 for the subsequent experiments due to having the highest estimated accuracy with SGIs, which is more in line with the exercises on Artemis and the original intent of the system.

Name	SGI	Tokens	Feedbacks	Est. Acc. (%)
Prompt 1	Yes	1539	3.12	71.8%
	No	1274	4.32	50.0%
Prompt 2	Yes	1549	3.12	<b>83.3%</b>
	No	1167	3.24	85.2%
Prompt 3	Yes	1484	3.04	81.6%
	No	1155	3.72	74.2%
Prompt 4	Yes	1550	3.16	77.2%
	No	1178	3.40	<b>89.4%</b>
Prompt 5	Yes	1587	3.16	79.7%
	No	1212	3.32	88.0%

**Table 7.4:** Comparative Analysis of Five Different Prompts in Experiment T1. Results are segmented based on the inclusion (*Yes*) or exclusion (*No*) of Structured Grading Instructions (SGI). Metrics include the average token usage, the average number of feedback suggestions produced, and the estimated accuracy as assessed by the LLM-as-a-judge methodology.

**Experiment T2: Model Comparison.** In model comparison, GPT-4, although computationally demanding, did not significantly outperform GPT-3.5-Turbo in terms of estimated accuracy, scoring 80.1% and 82.2% respectively. LLaMA-13b, however, lagged notably behind, achieving only 44.5% estimated accuracy. As for generation speed, GPT-3.5-Turbo was the fastest, clocking at 5.58 seconds per submission, followed by GPT-4 at 10.09 seconds, and LLaMA-13b at 22.21 seconds. The results are summarized in Table 7.5.

Model	Tokens	Time (s)	Feedbacks	Est. Acc. (%)
GPT-3.5-Turbo	1522	5.58	3.03	82.2%
GPT-4	1406	10.09	3.32	80.1%
LLaMA-2-13b-Chat	2008	22.21	1.91	44.5%

**Table 7.5:** Comparative Analysis of LLMs in Experiment T2. Metrics include average token usage, generation time in seconds, the average number of feedback suggestions produced, and the estimated accuracy assessed by the LLM-as-a-judge methodology.

**Experiment T3: LLM-as-a-Judge Evaluation.** In a manual evaluation process for cross-validating the LLM-as-a-judge methodology, a non-expert reviewed 303 pieces of feedback. The manual acceptance rate was 79.5%, closely aligning with the LLM-as-a-judge’s 82.2%. Agreement between the two evaluation methodologies was observed in 95% of the cases, strengthening the credibility of the LLM-as-a-judge methodology.

**Experiment T4: Exercise Comparison.** In a broader check across multiple text exercises, the estimated accuracy ranged from 64.8% to 98.0%, with the exercise “Design Goal Trade-offs” remarkably scoring 98.0%. This suggests that the LLM-based feedback generation system can generalize well across different types of text exercises within the same educational context. Generation time and token usage varied across exercises, with all taking, on average less than 2 196 tokens and 10.24 seconds per submission. Results are summarized in Table 7.6.

### 7.3.2 Programming Exercises

**Experiment P1: Model Comparison.** For the “Shoppende Pinguine” programming exercise, as summarized in Table 7.7, GPT-4 generated almost double the amount of feedback suggestions compared to GPT-3.5-Turbo,

Exercise Title	Tokens	Time (s)	Feedbacks	Est. Acc. (%)
H01E02 Change in Software Development	1522	5.58	3.03	82.2
H03E03 Bumpers Visionary Scenario	1813	3.43	1.22	64.8
H06E04 Inheritance vs. Delegation	2145	9.41	6.75	78.3
H01E02 Change in Software Development	2196	10.24	7.04	73.6
H04E03 Design Goal Trade-offs	1961	8.30	3.01	98.0

**Table 7.6:** Comparative Analysis of Selected Text Exercises in Experiment T4. Metrics include average token usage for feedback generation, computational time in seconds required for each submission, the average number of feedback suggestions produced, and the estimated accuracy as assessed by the LLM-as-a-judge methodology.

with 18.70 and 10.81 respectively. However, this increase came at the cost of significantly higher computational time, clocking at 160.85 seconds for GPT-4 versus 57.08 seconds for GPT-3.5-Turbo. Surprisingly, considering the number of feedback suggestions, the token usage was lower for GPT-4, standing at 11409 as opposed to GPT-3.5-Turbo’s 12953 per submission.

Model	Tokens	Time (s)	Feedbacks
GPT-3.5-Turbo	12953	57.08	10.81
GPT-4	11409	160.85	18.70

**Table 7.7:** Comparative Analysis of GPT-3.5-Turbo and GPT-4 in Experiment P1. Metrics include the average token usage for each model, the computational time in seconds required for feedback generation, and the average number of feedback suggestions produced.

## 7.4 Findings

The evaluation reveals critical insights into the LLM-based feedback generation system’s performance, scalability, and adaptability within Artemis’ educational context.

### 7.4.1 Text Exercises

**Prompt Optimization.** The research and development playground proved effective for prompt optimization, demonstrating that carefully engineered prompts could significantly improve the estimated accuracy of feedback suggestions. Prompts with the exercise using summarized structured grading

instructions surprisingly showed higher performance in some cases, challenging the assumption that detailed SGIs are always superior for feedback generation.

**Model Comparison.** GPT-3.5-Turbo emerged as the most efficient and nearly as accurate as GPT-4, suggesting that the latter’s computational overhead may not justify its use in this specific context. LLaMA-13b lagged considerably, indicating that not all LLMs are equally suited for the task at hand and need to be optimized.

**Method Validation.** The LLM-as-a-judge methodology successfully approximated human judgment, with a high degree of agreement between manual and automated evaluations. This validates the methodology as a scalable and reliable approach for quality and accuracy assessment.

**Exercise Generalization.** The system exhibited a high degree of adaptability across different types of text exercises. However, the estimated accuracy varied significantly, highlighting the importance of tailoring the system to specific educational contexts and grading criteria. In our evaluation, “Bumpers Visionary Scenario” achieved the lowest estimated accuracy of 64.8%, while on average, the system scored 79.38% over all exercises. This finding underlines the system’s potential to generalize across different educational content.

## 7.4.2 Programming Exercises

**Feasibility Assessment.** While GPT-4 generated a significantly higher number of feedback suggestions, it did so at a much higher computational cost than GPT-3.5-Turbo. This calls into question the scalability of using more advanced models for programming exercises, especially given that the quality of these suggestions could not be evaluated due to time constraints.

## 7.5 Discussion

The experiments and results have unfolded several dimensions of the LLM-based feedback generation system’s capabilities and limitations. This section interprets these findings, linking them to the broader context of Artemis and educational technology.

### 7.5.1 Text Exercises

**Prompt Engineering.** Prompt optimization emerged as a pivotal factor for enhancing system performance. The experiment revealed that even minor variations in the prompt could yield different outcomes. The use of summarized SGIs achieving higher estimated accuracy in some cases suggests the LLM’s ability to distill essential aspects of grading. This could be valuable for Artemis when SGIs can be complex and lengthy.

**Model Suitability.** The model comparison experiment laid bare the trade-offs between computational efficiency and performance. While GPT-4 didn’t significantly outperform GPT-3.5-Turbo in accuracy, it was computationally more demanding, raising questions about its practicality in large-scale deployments within Artemis. This finding underscores the importance of a sensible selection of LLMs tailored for specific educational settings.

**Method Validation and Scalability.** The LLM-as-a-judge methodology offers a scalable avenue for system evaluation. Its high degree of agreement with human judgment underlines its utility for initial quality assurance in large-scale deployments. This method could also be integrated into Artemis to assist in ongoing evaluations.

**Generalization Across Exercises.** The system’s ability to generalize across different text exercises reaffirms its potential for broader applications within Artemis. However, the variability in estimated accuracy suggests that while the system is flexible, it may require specific tuning for different types of exercises. To meet this need for customization, we could allow tutors to configure custom instructions for the LLM-based system, similar to how OpenAI is doing it with ChatGPT<sup>3</sup>.

### 7.5.2 Programming Exercises

**Feasibility and Scalability.** The single experiment for programming exercises revealed that while generating feedback is technically feasible, scalability remains a concern. GPT-4’s higher computational demands and longer generation times suggest that more optimized approaches are needed for practical deployments in Artemis, especially given the usually higher volume of submissions in programming exercises in combination with the size of each submission.

---

<sup>3</sup><https://openai.com/blog/custom-instructions-for-chatgpt>

### 7.5.3 Implications for Artemis

The findings offer actionable insights for the future development of Artemis. The research and development environment’s effectiveness in prompt optimization can serve as a blueprint for iterative enhancements. Moreover, the LLM-as-a-judge methodology could become an integral part of Artemis’ semi-automatic assessment process, aiding in quality assurance. Finally, the data on model efficiency and estimated accuracy could inform strategic decisions on LLM deployments for both text and programming exercises within Artemis.

## 7.6 Limitations

**Limited Scope.** The study’s scope was limited to one type of programming exercise and a few text exercises with few randomly selected submissions, constraining the generalizability of the findings. Future work could extend the evaluation to a broader range of exercises and educational contexts with representative samples of submissions.

**Time and Computational Constraints.** The constraints on time and computational resources limited the depth of the experiments, particularly for programming exercises. Future research should aim for more exhaustive evaluations.

**Quality Metrics.** While estimated accuracy served as a quantitative metric, qualitative aspects of the feedback were not evaluated. Additionally, we only assessed the accuracy of whether the feedback was accepted with minor to no modifications. We did not account for the accuracy of the points within the feedback and did not differentiate between points and feedback text. Furthermore, we did not evaluate qualitative aspects of the feedback. Future work could involve a comprehensive evaluation of feedback quality, involving students and tutors.

**LLM-as-a-Judge Methodology.** While the LLM-as-a-judge methodology showed a high degree of agreement with the human judgment of a non-expert on a relatively small sample, it is still an automated method. The actual quality of feedback suggestions may vary when evaluated by multiple human assessors, especially those who are domain experts in software engineering.



**Fixed Configuration.** The experiments employed fixed configurations for each LLM, which might not be optimal. For example, tuning hyperparameters such as temperature, presence penalty, and frequency penalty could potentially impact the quality and quantity of the generated feedback.

# Chapter 8

## Summary

In the following sections, the status of the thesis, significant conclusions, and discuss potential directions for future work.

### 8.1 Status

Tables 8.1 and 8.2 show the status of the functional requirements for text and programming exercises within the learning management context. For the research and development context, an overview of the status of the functional requirements is given in Table 8.3. We indicate the status of each functional requirement with ● for realized, ◐ for partially realized, and ○ for not realized. We discuss the realized and open goals in the following sections.

Status	Functional Requirement	FR
●	Generate Feedback Suggestions	FR T.1
●	Review Feedback Suggestions	FR T.2
○	Learn from Feedback	FR T.3
●	Link Structured Grading Instruction	FR T.4
◐	Handle Multiple Languages	FR T.5

**Table 8.1:** Status of Functional Requirements for Text Exercises within the Learning Management Context.

#### 8.1.1 Realized Goals

We have realized most of the main goals of this thesis regarding the learning management context and the research and development context. The following sections summarize the realized goals for each context.

Status	Functional Requirement	FR
●	Generate Feedback Suggestions	FR P.1
●	Review Feedback Suggestions	FR P.2
◐	Handle Multiple Programming Languages	FR P.3
○	Learn from Feedback	FR P.4
◐	Link Structured Grading Instruction	FR P.5
○	Reference Test Results	FR P.6
○	Reference Build Outputs	FR P.7

**Table 8.2:** Status of Functional Requirements for Programming Exercises within the Learning Management Context.

Status	Functional Requirement	FR
●	Use Multiple LLMs	FR R.1
●	Configure Feedback Generator	FR R.2
○	Evaluate Feedback Quality	FR R.3
●	Track Token Usage and Generation Time	FR R.4
●	Compare Modules	FR R.5
●	Define Experiment	FR R.6
●	Conduct Experiment	FR R.7
●	Rate Feedback	FR R.8
◐	Automatic Evaluation	FR R.9
●	Import Configurations	FR R.10
●	Export Data	FR R.11

**Table 8.3:** Status of Functional Requirements for the Research and Development Context.

## Learning Management Context

**Text Exercises.** We have developed a feedback generation module for text exercises, as outlined in Section 6.2. This module effectively produces feedback suggestions, fulfilling the requirements detailed in FR T.1. Our evaluation, discussed in Chapter 7, confirms that the module generates feedback rapidly (NFR8), cost-effectively (NFR11), and accurately (NFR6) using GPT-3.5-Turbo. It also scales well to thousands of submissions (NFR10). The system offers some multilingual support (FR T.2), particularly for languages that the LLM is trained on, although further enhancements can be made. During the manual evaluation, we found that the LLM consistently and accurately applies structured grading instructions (FR T.4). Addition-

ally, a collaborator has integrated the review feature for these feedback suggestions into Artemis (FR T.2), ensuring seamless compatibility with our module.

**Programming Exercises.** The feedback generation module for programming exercises achieves the core objective, fulfilling FR P.1. This module has been seamlessly integrated into Artemis through a review feature implemented by a collaborator (FR P.2), enabling tutors to easily accept or reject generated feedback suggestions. Although the module currently supports multiple file types and programming languages (FR P.3), it should be considered with more care in future work. We observed that the system’s ability to map structured grading instructions to specific code snippets is not yet fully reliable (FR P.5). This limitation warrants further investigation and refinement to enhance the module’s applicability and accuracy.

### Research and Development Context

We successfully setup the research and development environment, covering most of the functional requirements. It allows for the use of multiple LLMs (FR R.1) and other configurable feedback generator settings (FR R.2). A researcher can define an experiment (FR R.6) and subsequently conduct it (FR R.7). While conducting an experiment, the researcher can compare the result of multiple modules (FR R.5) and rate the feedback suggestions (FR R.8) by either accepting them or rejecting them, similar to how a tutor would. Additionally, we generate automatic ratings to gauge the system’s accuracy for text exercises (FR R.9), although this is not yet implemented for programming exercises. The environment also tracks token usage and generation time (FR R.4). Moreover, it facilitates data and configuration imports and exports (FR R.11, FR R.10) for enhanced reproducibility and iterative development.

#### 8.1.2 Open Goals

Some of our goals are not yet fully realized, primarily due to time constraints and the complexity of the problem. In the following sections, we discuss the open goals for each context.

### Learning Management Context

**Text Exercises.** The feedback generation system does not yet include a mechanism for learning from feedback (FR T.3). This capability could help

improve the system’s performance based on tutors’ past feedback. We would have liked to explore this feature for GPT-3 and GPT-3.5-Turbo, as it gained fine-tuning support just recently, but due to time constraints, we could not do so in this thesis. Another open goal is full multilingual support (FR T.5). While the system can generate feedback in languages the LLM is trained in, it might still generate feedback in the wrong language. In the future, we could adjust the system to only generate feedback in the language of the submission.

**Programming Exercises.** Similar to text exercises, the system lacks a learning mechanism (FR P.4). Additionally, the system’s ability to handle multiple programming languages is partial and needs refinement (FR P.3). The linking of structured grading instructions to specific code snippets is not yet fully reliable (FR P.5), and other non-functional requirements such as scalability (NFR10), cost-efficiency (NFR11), and feedback accuracy (NFR6) need further investigation. The system does not yet reference test results (FR P.6) or build outputs (FR P.7) when generating feedback, limiting its ability to generate feedback suggestions that require such references.

### Research and Development Context

The research environment does not yet provide a robust method for evaluating the quality of generated feedback (FR R.3). This evaluation is vital for validating the system’s efficacy and should be incorporated in future work. The ability to automatically rate feedback is partially implemented (FR R.9) for text exercises but remains unaddressed for programming exercises due to its associated complexity.

## 8.2 Conclusion

This thesis addressed the critical challenge of providing individualized, meaningful feedback in large educational settings, specifically within the Learning Management System (LMS) Artemis at the Technical University of Munich. The primary contribution is developing and evaluating a feedback generation system that leverages large language models to semi-automate the assessment process for text and programming exercises. This system aims to enhance both the quality of education for students and the efficiency of the grading process for tutors.

The feedback generation system for text exercises has shown promise, with LLMs such as GPT-3.5-Turbo generating rapid, cost-effective, and ac-

curate feedback. For programming exercises, the system successfully generates feedback suggestions but requires further evaluation and refinement. To support ongoing research and development, we also established an environment that allows for the easy configuration of LLM modules, evaluation of generated feedback, and tracking various performance metrics. This environment is the foundation for future improvements and research into feedback generation and automated assessment methods.

The thesis also contributes to the academic discourse on the applicability of LLMs in educational technology. It provides empirical data and analysis that shed light on the capabilities and limitations of using LLMs for feedback generation and presents valuable insights for future research in this area.

## 8.3 Future Work

After addressing the open goals in Section 8.1.1, we identify further steps to improve automated feedback in Artemis. The section will cover three main areas: fine-tuning large language models, introducing a more sophisticated approach for programming exercises, and the extension to modeling exercises.

### 8.3.1 Fine-Tuning Large Language Models

Fine-tuning LLMs such as GPT-3 or GPT-3.5-Turbo offers a robust method for elevating the automated feedback system in Artemis. Fine-tuning enables models to generate more relevant, consistent, and precise feedback by learning from past feedback, similar to CoFee [Ber22]. This aligns with OpenAI's guidelines on fine-tuning, which advocate for its utility in producing higher quality results and in making the system more cost-efficient<sup>1</sup>.

Secondly, fine-tuning simplifies the complexity tied to prompt optimization. The model learns the desired style and format, thereby mitigating the need for intricate prompts. Next to the model's increase in reliability and robustness, the token consumption also benefits, allowing for shorter prompts that reduce both latency and costs, as reiterated by OpenAI's guidelines.

For the feedback generation of text exercises, as detailed in Section 6.2, fine-tuning can readily be applied by simply fine-tuning on incoming feedback and subsequently using the fine-tuned model for feedback generation. For programming exercises, as detailed in Section 6.3, the first step is to find better ways to provide high-quality context to the LLM before applying fine-tuning for the feedback generation process.

---

<sup>1</sup><https://openai.com/blog/gpt-3-5-turbo-fine-tuning-and-api-updates>

Exploring open-source alternatives such as LLaMA-2 [TMS] also holds potential, specifically for research purposes. Comparable in performance to commercial models, LLaMA-2 allows fine-tuning on in-house infrastructure, enabling more control over the training process.

In summary, fine-tuning is a pragmatic approach to enhance automated feedback capabilities of Artemis. It sets a promising direction for future research, aiming to produce more contextual, consistent, and task-oriented feedback.

### 8.3.2 Agentic Approach for Programming Exercises

The feedback generation process for programming exercises discussed in Section 6.3 has certain limitations, most notably in its context handling. The current model tends to flood the LLM with abundant information from a changed file, while ignoring valuable context from other files. This not only increases computational demands but also compromises the relevance of the feedback generated. To overcome these issues, we propose an agentic approach to feedback generation.

In this approach, the LLM functions less like a simple query-response mechanism and more like an intelligent agent. This agent emulates a tutor's thought process when evaluating a piece of code. It decomposes the complex task of code assessment into smaller, manageable tasks and outlines a series of actions to reach a meaningful conclusion. These actions may range from reading the problem statement, and consulting test results to examining specific lines of code or tracing function calls.

As the agent progresses through these tasks, it accumulates notes to build context. Subsequently, we use this context to generate feedback suggestions for the assessment. This approach allows for more targeted feedback, focusing on the context of specific code blocks and functions rather than the entire file. It might still be computationally expensive but potentially yield more relevant and higher-quality feedback suggestions.

### 8.3.3 Modeling Exercises

While this thesis focuses on using LLMs for text and programming exercises in the Artemis Learning Management System, extending this framework to modeling exercises is a promising avenue for future work. The application of LLMs in this area remains largely unexplored and could offer significant advantages, such as generating context-aware feedback or recognizing patterns in Unified Modeling Language (UML) diagrams.

Artemis already employs a semi-automatic, machine learning-based assessment for modeling exercises [Kru22]. This existing framework serves as a solid foundation for incorporating LLM-generated feedback. One significant challenge is the graphical nature of modeling exercises, which contrasts with the text-based focus of LLMs. Nevertheless, this challenge is not impossible. UML diagrams can be converted into structured textual formats such as JSON or Markdown, formats LLMs can process effectively. This would allow LLMs to offer feedback suggestions for modeling exercises in a manner similar to our contributions to text and programming exercises.

In summary, extending the application of LLMs to modeling exercises within Artemis represents a promising avenue for future research, seamlessly integrating with existing structures and processes.



# Appendix A

## Data Exploration

This chapter examines the specific data within Artemis pertinent to the thesis, concentrating on text and programming exercises related to assessment. It begins with data acquisition and selection procedures and progresses to exploratory data analysis (EDA) concerning text and programming exercises.

### A.1 Data Acquisition and Selection

The Artemis team supplied the data utilized in this exploration, delivered as an anonymized database dump from the Technical University of Munich’s Artemis instance, dated *March 29, 2023*.

We confine our examination to text and programming exercises that fulfill certain conditions: they must not be part of a test course, must contribute to the cumulative grade, and must be either an exam or non-exam exercise.

Concerning the Artemis data model, as depicted in Figure A.1, the focus is on participations with at least one submitted submission and a corresponding rated result. We omit all others. For this analysis, only the final submission and the latest corresponding rated result for each participation are considered. This ensures that only the final submissions and outcomes are assessed, rather than preliminary ones. For text exercises specifically, we disregard any submissions lacking text.

### A.2 Exploratory Data Analysis (EDA)

The aim of this section is to provide a foundational understanding of the Artemis data, particularly focusing on text and programming exercises that align with the objective of this thesis: generating automatic feedback to support tutors during assessment. We’ll explore overarching trends within

APPENDIX A. DATA EXPLORATION

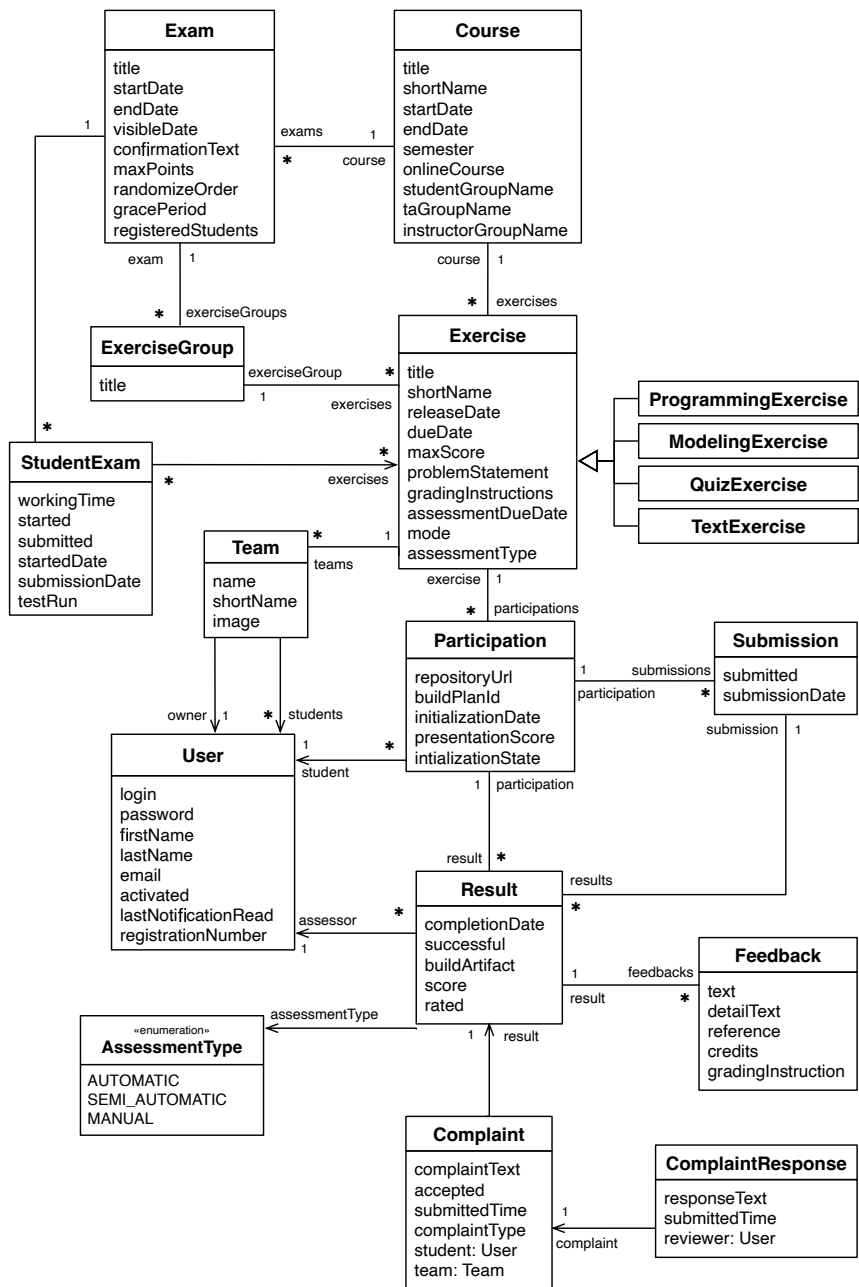
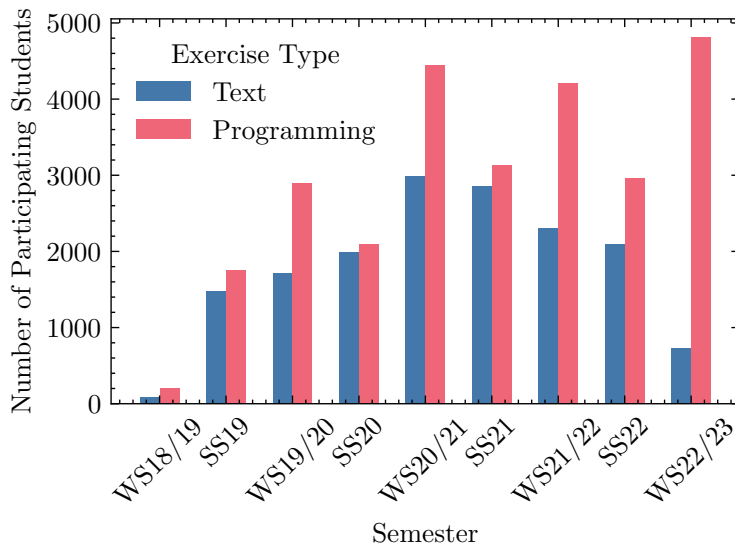


Figure A.1: Simplified Data Model of the Artemis Application Server.  
 Source: <https://docs.artemis.cit.tum.de/dev/system-design>

Artemis, followed by specific analyses of text and programming exercises. These insights are not meant to be exhaustive but rather establish a well-informed basis for establishing the automated feedback generation processes in this thesis.

### A.2.1 Overview

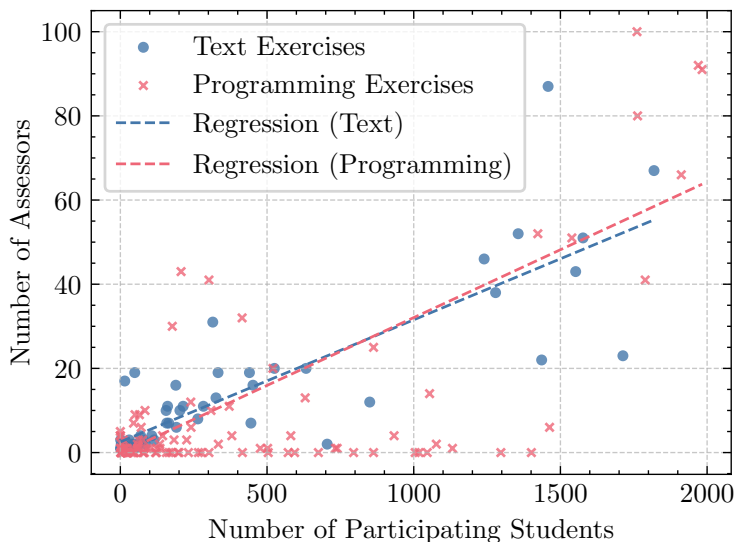
Artemis has amassed significant usage data over its period of deployment at the Technical University of Munich. The volume of student participation has been trending upward for both text and programming exercises, as captured in Figure A.2. Specifically, 4813 students engaged in programming exercises during the winter semester of 2022/23, marking a new peak. Conversely, participation in text exercises peaked at 2992 during the winter semester of 2020/21 but has since experienced a reduction. The data indicates that programming exercises play a larger role than text exercises, substantiating their role as a critical part of the Artemis platform.



**Figure A.2:** Student Participation Trends in Text and Programming Exercises Across Semesters.

Equally important is the tutors’ engagement with Artemis, as seen in Figure A.3. For programming exercises, the largest course in the dataset, “Praktikum: Grundlagen der Programmierung WS21/22”, had a student-to-assessor ratio of 21.79. Text exercises displayed a slightly higher ratio of 27.13 students per assessor in the largest course, “Introduction to Software Engineering (IN0006) SS21”. The regression analysis further reveals

the general scaling challenge: a student-to-assessor ratio of 31.02 for programming and 34.48 for text exercises, excluding courses solely dependent on automated assessments. Note that all 57 courses using text exercises have at least one assessor, whereas 43 of the 107 courses using programming exercises have no assessor. These numbers substantiate the urgent need for automated feedback solutions, aligning directly with the thesis objectives.



**Figure A.3:** Student-to-Assessor Ratios in Artemis Courses. Data shows the correlation between the number of participating students and assessors, broken down by text and programming exercises per course. The regression lines have coefficients of 0.029 for text and 0.032 for programming exercises and exclude courses solely reliant on automated assessments, *i.e.* zero assessors.

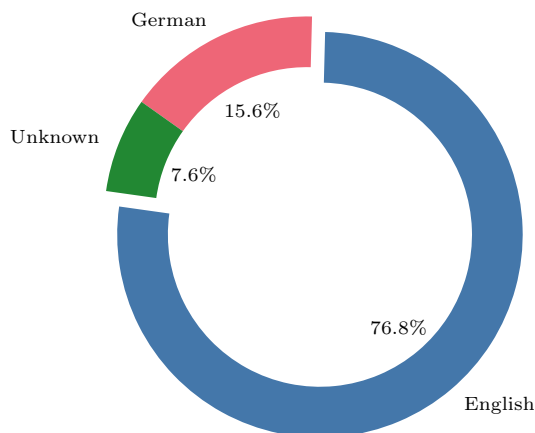
This overview underscores the growing use of Artemis for educational activities while also highlighting the scaling challenges it faces, especially in the context of assessment. These findings set the stage for a deeper dive into the characteristics of text and programming exercises, aiming to identify specific areas where automated feedback could yield significant improvements.

### A.2.2 Text Exercises

This subsection delivers an in-depth exploration of text exercises, focusing on key data-driven insights that are directly relevant to the objective of automated feedback generation. We leverage multiple visualizations to substantiate our analyses.

### Language Distribution

Figure A.4 exhibits the language distribution among text exercises. Remarkably, 76.8% of the total text submissions are in English, followed by 15.6% in German and 7.6% labeled as unknown. This high prevalence of English submissions underscores the need for an automated feedback system adept at processing the English language while also providing secondary support for German.



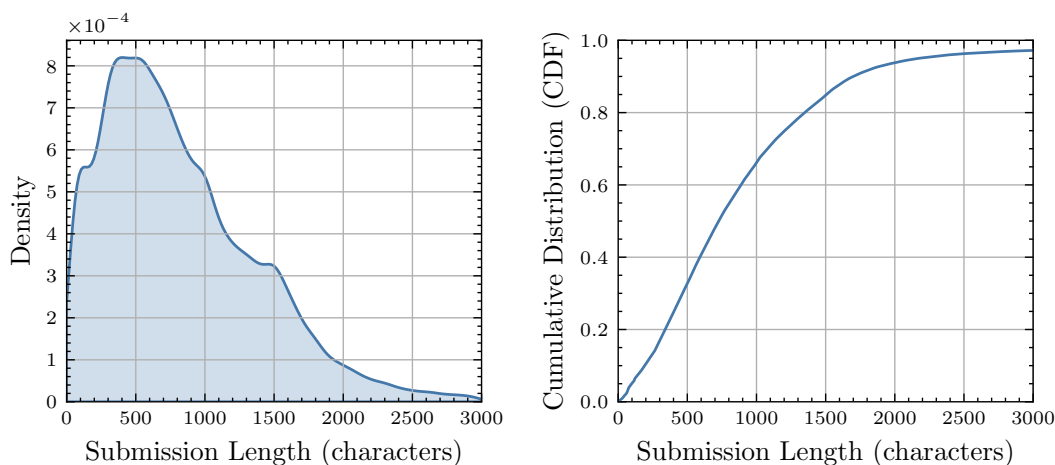
**Figure A.4:** Distribution of Languages in Text Exercises.

### Text Length in Submissions

Analyzing the text length of submissions is crucial for tailoring the feedback process. This is especially important when using large language models, which can struggle with long textual contexts. Figure A.5 demonstrates that the majority of text submissions have a character count between 300 and 900, with a median of 726 characters. The distribution exhibits left-skewness, featuring a long tail of submissions surpassing 2000 characters. Notably, the 95th percentile caps at 2170 characters, and the 98th percentile reaches 3871 characters, suggesting a concentration of shorter submissions within the dataset.

### Feedback Type Distribution

Figure A.6 shows how different types of feedback have been used over time for text exercises on Artemis. Most of the feedback is “Referenced”, referencing a text block, followed by “Unreferenced” feedback, belonging to the whole



**Figure A.5:** Distribution of Text Submissions Length in Characters.

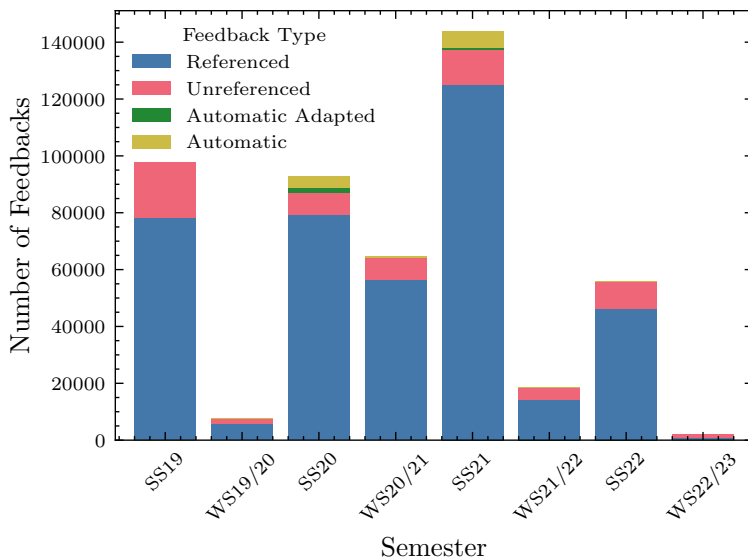
submission. An existing approach in Artemis, called CoFee [BB19], is responsible for the “Automatic” and, slightly adjusted, “Automatic Adapted” feedback. This tool was mostly used between the summer semesters of 2020 and 2021.

Although this data includes all courses, it is worth noting that specific courses using CoFee had more automatic feedback than the figure suggests. Interestingly, almost no automatic feedback was used from winter semester 2021/22 to the winter semester 2022/23. The last time automatic feedback was significantly used was in the summer semester 2021.

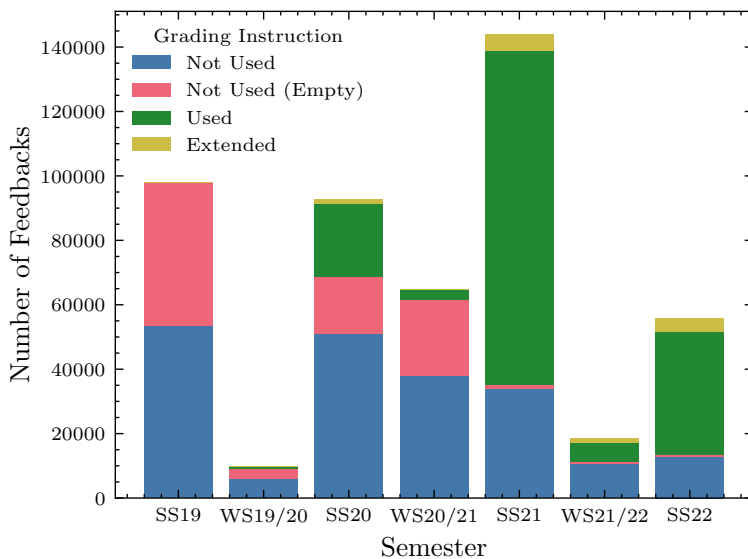
The drop in automated feedback usage during recent semesters suggests a need for further study to understand its effectiveness. This trend also highlights the importance of this thesis, which aims to develop a new automated feedback system for Artemis.

### Usage of Structured Grading Instructions

Structured grading instructions are a feature of Artemis that allows tutors to apply predefined feedback to a submission. Figure A.7 reveals a significant adoption of structured grading instructions in recent semesters. A majority of the feedback instances are directly linked to these instructions. Notably, there is a surge of empty feedback entries between the summer semester 2019 and the winter semester 2020/2021. This can be attributed to the likely deletion of associated grading instructions. Only a minor fraction of feedback, which uses structured grading instructions, is extended by additional comments. This suggests that the existing structured grading instructions largely fulfill the need for effective feedback.



**Figure A.6:** Distribution of Feedback Types for Text Exercises Across Semesters. “Automatic” and “Automatic Adapted” refer to feedback provided by CoFee [BB19]. “Referenced” feedback is linked to a specific text passage, whereas “Unreferenced” feedback is not.

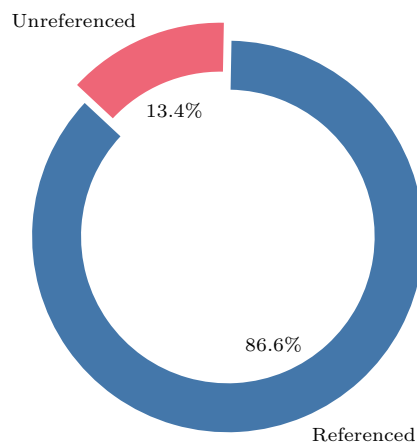


**Figure A.7:** Usage of Structured Grading Instructions in Text Exercises. “Not Used (Empty)” refers to feedback that has no content and linked grading instruction, probably due to deletion of the grading instruction. “Extended” denotes feedback that is linked to a grading instruction, but also additional comments.

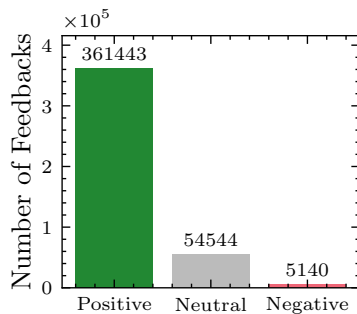
### Referenced and Unreferenced Feedback

Figure A.8 delves into the nature of the feedback given in text exercises. The overarching takeaway, illustrated in Figure A.8a, is that a substantial 86.4% of the feedback is referenced. Within this subset, Figure A.8b reveals that 85.8% of the referenced feedback is positive, 13.0% is neutral, and a mere 1.2% is negative. In contrast, unreferenced feedback, showcased in Figure A.8c, largely comprises neutral comments at 64.7%, followed by positive at 33.3%, and negative at 2.0%.

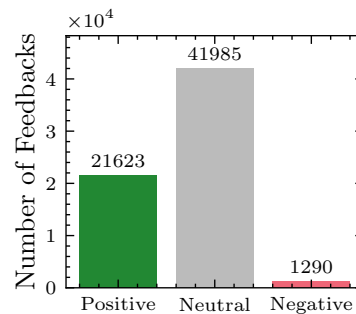
This data underscores the utility of an automated system capable of generating primarily referenced and positive feedback, mirroring existing human-assessment patterns.



(a) Distribution of Referenced and Unreferenced Feedback.



(b) Types of Referenced Feedback.



(c) Types of Unreferenced Feedback.

**Figure A.8:** Analysis of Referenced and Unreferenced Feedback in Text Exercises.



## Summary

This subsection synthesized key insights from the data-driven evaluation of text exercises in Artemis, informing the development of an automated feedback system. The highlights are as follows:

- **Language Dominance:** English submissions are predominant, necessitating a system skilled in English language processing.
- **Text Length:** Submissions are primarily shorter, with a median of 726 characters, and 95th percentile at 2170 characters.
- **Feedback Types:** A recent decline in automated feedback underscores the relevance of this research.
- **Structured Grading Instructions:** Existing structured grading instructions are widely used, effective, and largely self-sufficient for feedback.
- **Feedback Nature:** A system should mainly produce referenced and positive feedback, alternatively using neutral feedback for remarks, to align with current practices.

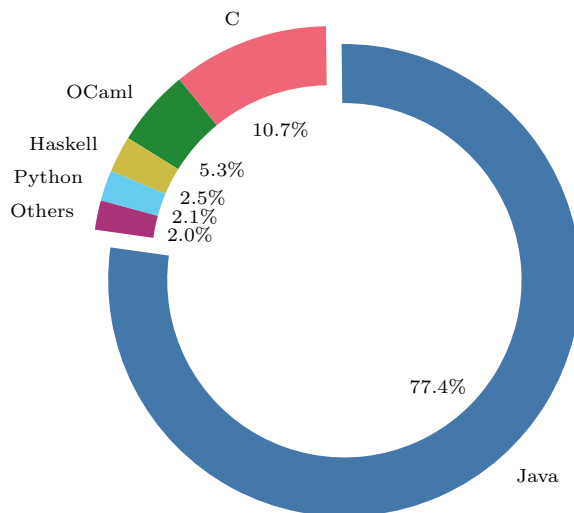
These insights collectively serve as foundational elements for designing an efficient, context-aware automated feedback system.

### A.2.3 Programming Exercises

This subsection offers a comprehensive overview of programming exercises, presenting critical insights that have implications for automated feedback generation in the Artemis system. Multiple visualizations support our data-driven conclusions.

#### Programming Languages Distribution

Figure A.9 reveals a striking dominance of Java, accounting for 77.41% of participations. C and OCaml follow distantly, with percentages of 10.71% and 5.27%, respectively. The dominance of Java in the dataset highlights the need for a feedback system that is proficient in both Java syntax and semantics.



**Figure A.9:** Distribution of Programming Languages in Programming Exercises.

### Feedback Type Distribution

In Artemis, the feedback types for programming exercises fall into two primary categories: manual and automatic, as depicted in Figure A.10. The subtypes for manual feedback include “Referenced” and “Unreferenced”, while automatic feedback consists of “Test Case”, *i.e.* automatic testing, and “Static Code Analysis”.

Unreferenced feedback consistently outnumbers referenced feedback, as evidenced by Figure A.10a. The only major shifts occur during the winter semesters of 2020/21 and 2021/22, where a larger portion of referenced feedback exists, making it more balanced. This data pattern suggests that the current manual assessment process leans heavily towards generalized, submission-level comments, as opposed to offering feedback on specific code locations. The persistence of this trend highlights inefficiencies and areas for potential improvement within the manual feedback system.

On the other hand, Figure A.10b demonstrates that automatic feedback dramatically surpasses manual feedback by one to two orders of magnitude in quantity. The volume of automatic feedback has been on a steady rise, which further underscores its efficiency in generating immediate, iterative assessments.

The disparity between manual and automatic feedback can largely be attributed to the human effort required for manual evaluations. While automatic feedback is generated with minimal human intervention, manual feedback demands a more labor-intensive process. Specifically, the dominance of

unreferenced feedback in the manual category points to a lack of targeted, code-specific commentary.

Given these observations, our research aims to introduce a more automatic feedback generation process that supplements the manual assessment process, thereby enhancing its efficiency and specificity. By doing so, we intend to bridge the gap between the volume and quality of manual assessments without compromising the depth of qualitative feedback.

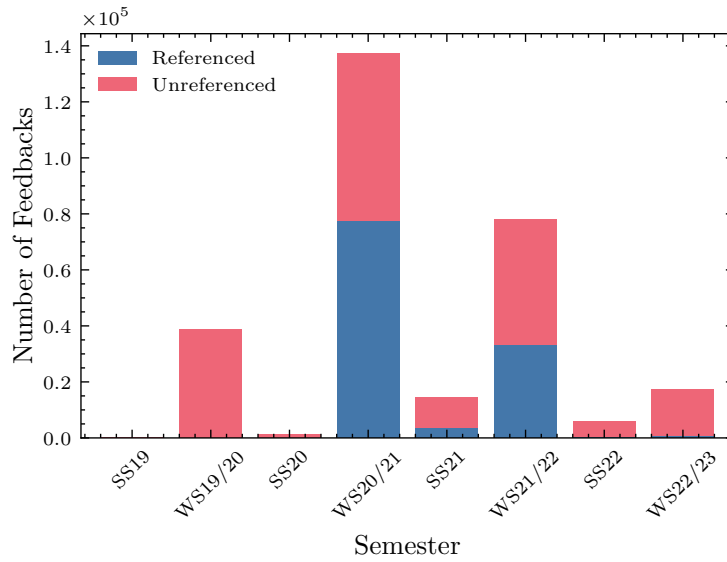
### Usage of Structured Grading Instructions

Figure A.11 reveals that structured grading instructions are barely used for manual feedback in programming exercises. A small fraction of feedback during summer semester 2021 and winter semester 2021/22 utilized grading instructions, but apart from that, the usage is negligible. This trend underscores the hypothesis that structured grading instructions have not been widely adopted in the manual feedback process of programming exercises. This finding is in stark contrast to the usage of structured grading instructions in text exercises, as seen in Appendix A.2.2.

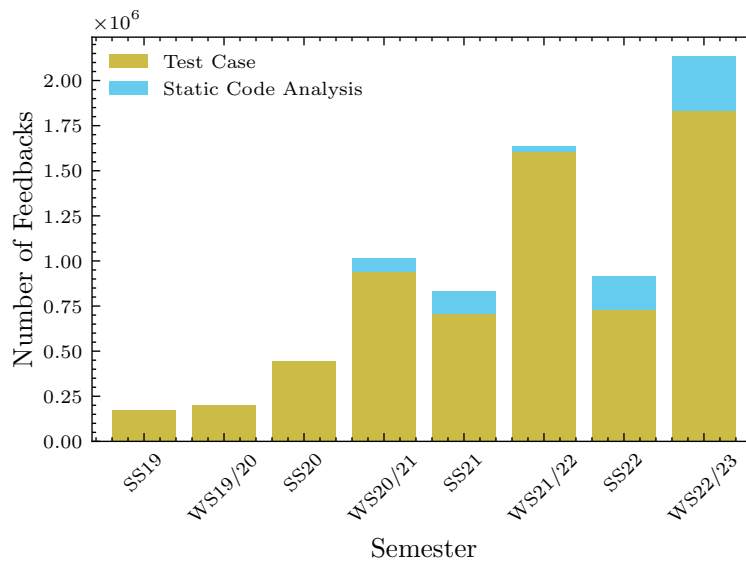
### Referenced and Unreferenced Feedback

A critical examination of the referenced and unreferenced feedback in programming exercises is found in Figure A.12. Notably, Figure A.12a reveals that an overwhelming 88.5% of feedback is unreferenced. In contrast, the ratio between referenced and unreferenced feedback is reversed for text exercises, as seen in Appendix A.2.2. Within the unreferenced category, Figure A.12c breaks down the types as 55.1% positive, 44.8% neutral, and a marginal 0.02% negative. Conversely, Figure A.12b indicates that referenced feedback is predominantly neutral (84.7%), followed by negative (8.5%) and positive (6.8%) remarks.

The high prevalence of positive, unreferenced feedback may result from tutors predominantly using this category for awarding credits. Conversely, the tilt towards neutral comments in referenced feedback suggests that tutors use code annotations for making clarifications or providing non-evaluative guidance. These observations align with the hypothesis that tutors allocate credits via unreferenced feedback while offering explanatory or cautionary remarks as referenced, neutral feedback in the code. This behavior warrants further investigation as it provides critical context for designing an automated feedback generation system intended to assist in semi-automatic assessment processes.

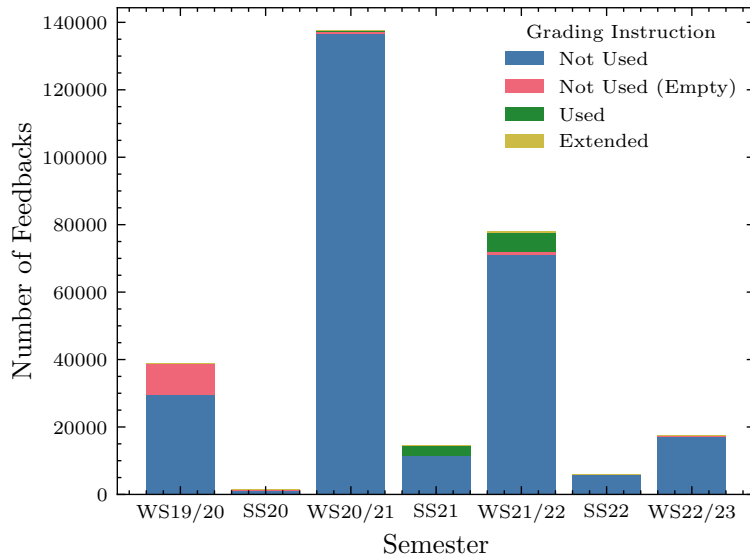


(a) Manual Feedback Types.



(b) Automatic Feedback Types.

**Figure A.10:** Distribution of Feedback Types for Programming Exercises Across Semesters.



**Figure A.11:** Usage of Structured Grading Instructions in Programming Exercises for Manual Feedback.

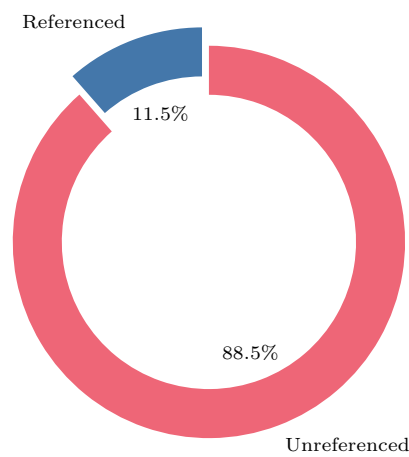
## Summary

This subsection distills the essential findings from our data-driven analysis of programming exercises within the Artemis system, shaping the design of an enhanced automated feedback mechanism. The pivotal observations are summarized as follows:

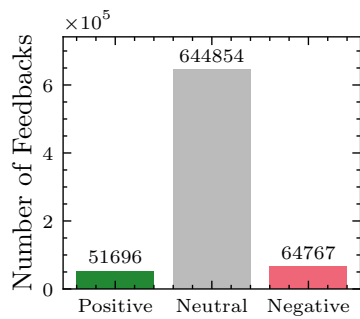
- **Language Dominance:** Java dominates with a 77.41% participation rate, necessitating Java expertise in the feedback system, followed by C and OCaml.
- **Feedback Types:** Automatic feedback outnumbers manual, signaling efficiency but also the need for refinement of the manual feedback process.
- **Structured Grading Instructions:** The low usage of structured grading instructions suggests they're not central to the current assessment process.
- **Feedback Nature:** Most manual feedback is unreferenced (88.5%) and positive, while referenced feedback tends to be neutral and clarifying.

These observations serve as foundational guidelines for developing an automatic feedback system. This system aims to enhance the volume and

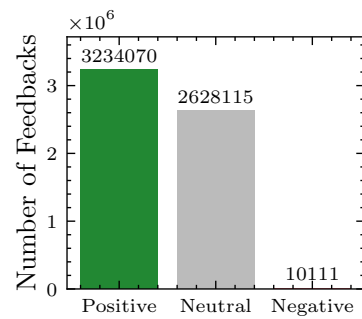
quality of manual assessments within the Artemis platform.



(a) Distribution of Referenced and Unreferenced Feedback.



(b) Types of Referenced Feedback.



(c) Types of Unreferenced Feedback.

**Figure A.12:** Analysis of Referenced and Unreferenced Feedback in Programming Exercises.

# List of Figures

2.1	Basic Workflow using Large Language Models . . . . .	9
4.1	Text Exercise Assessment Features in Artemis . . . . .	14
4.2	Programming Exercise Assessment Features in Artemis . . . . .	15
4.3	UML Use Case Diagram: Learning Management Context . . . . .	24
4.4	UML Use Case Diagram: Research and Development Context . . . . .	26
4.5	UML Class Diagram: Analysis Object Model . . . . .	27
4.6	UML Activity Diagram: Learning Management Context Assessment Workflow . . . . .	28
4.7	UML Activity Diagram: Research and Development Context Experiment Conduction Workflow . . . . .	29
4.8	Proposed Inline Feedback Suggestion UI . . . . .	30
4.9	Proposed Unreferenced Feedback Suggestion UI . . . . .	31
5.1	UML Component Diagram: Top-Level Subsystem Decomposition . . . . .	35
5.2	UML Component Diagram: Subsystem Decomposition of a LLM Module . . . . .	36
5.3	UML Component Diagram: Subsystem Decomposition of the R&D Playground . . . . .	37
5.4	UML Component Diagram: Subsystem Decomposition of the Feedback Suggestions Within Artemis . . . . .	38
5.5	UML Deployment Diagram: Hardware-Software Mapping . . . . .	39
6.1	UML Class Diagram: Module Interface . . . . .	41
6.2	UML Activity Diagram: Text LLM Module . . . . .	43
6.3	Prompt for Generating Feedback Suggestions for Text Exercises . . . . .	44
6.4	Prompt for Generating Evaluations for Text Exercises . . . . .	45
6.5	UML Activity Diagram: Programming LLM Module . . . . .	47
6.6	Prompt for Splitting the Grading Instructions by File for Programming Exercises . . . . .	48

## LIST OF FIGURES

---

6.7	Prompt for Splitting the Problem Statement by File for Programming Exercises . . . . .	49
6.8	Prompt for Generating Feedback Suggestions for Programming Exercises . . . . .	50
6.9	Module Configuration UI of the R&D Playground for the Text LLM Module . . . . .	53
6.10	Module Requests UI of the R&D Playground for Feedback Suggestions from Athena . . . . .	54
6.11	Define Experiment UI of the R&D Playground . . . . .	55
6.12	Configure Modules UI of the R&D Playground . . . . .	55
6.13	Conduct Experiment Exercise Details and Tutor Feedback UI of the R&D Playground . . . . .	56
6.14	Conduct Experiment Module Comparison UI of the R&D Playground for a Text Module . . . . .	56
6.15	Conduct Experiment Module Comparison UI of the R&D Playground for a Programming Module . . . . .	57
A.1	Simplified Data Model of the Artemis Application Server . . . . .	80
A.2	Student Participation Trends in Text and Programming Exercises Across Semesters . . . . .	81
A.3	Student-to-Assessor Ratios in Artemis Courses . . . . .	82
A.4	Distribution of Languages in Text Exercises . . . . .	83
A.5	Distribution of Text Submissions Length in Characters. . . . .	84
A.6	Distribution of Feedback Types for Text Exercises Across Semesters . . . . .	85
A.7	Usage of Structured Grading Instructions in Text Exercises . . . . .	85
A.8	Analysis of Referenced and Unreferenced Feedback in Text Exercises . . . . .	86
A.9	Distribution of Programming Languages in Programming Exercises . . . . .	88
A.10	Distribution of Feedback Types for Programming Exercises Across Semesters . . . . .	90
A.11	Usage of Structured Grading Instructions in Programming Exercises for Manual Feedback . . . . .	91
A.12	Analysis of Referenced and Unreferenced Feedback in Programming Exercises . . . . .	92



# List of Tables

7.1	Empty Feedback Rates in Artemis Courses for Text Exercises	59
7.2	Text Exercises Selected for Evaluation . . . . .	60
7.3	Potential Programming Exercises Selected for Evaluation . . .	61
7.4	Comparative Analysis of Five Different Prompts in Experiment T1 . . . . .	65
7.5	Comparative Analysis of LLMs in Experiment T2 . . . . .	66
7.6	Comparative Analysis of Selected Text Exercises in Experiment T4 . . . . .	67
7.7	Comparative Analysis of GPT-3.5-Turbo and GPT-4 in Experiment P1 . . . . .	67
8.1	Status of Functional Requirements for Text Exercises within the Learning Management Context . . . . .	72
8.2	Status of Functional Requirements for Programming Exercises within the Learning Management Context . . . . .	73
8.3	Status of Functional Requirements for the Research and Development Context . . . . .	73

# Bibliography

- [Ala05] Kirsti M Ala-Mutka. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education*, 15(2):83–102, June 2005.
- [BB] Jan Philip Bernius and Bernd Bruegge. Toward the Automatic Assessment of Text Exercises.
- [BB19] Jan Philip Bernius and Bernd Bruegge. Toward the automatic assessment of text exercises. In *2nd Workshop on Innovative Software Engineering Education (ISEE)*, pages 19–22, Stuttgart, Germany, 2019.
- [BD10] Bernd Bruegge and Allen H. Dutoit. *Object-Oriented Software Engineering: Using UML, Patterns, and Java*. Prentice Hall, Boston, 3rd ed edition, 2010.
- [Ber22] Jan Philip Bernius. *Automatic Assessment of Textual Exercises*. PhD thesis, Technische Universität München, 2022.
- [BJV13] Sumit Basu, Chuck Jacobs, and Lucy Vanderwende. Powergrading: A Clustering Approach to Amplify Human Effort for Short Answer Grading. *Transactions of the Association for Computational Linguistics*, 1:391–402, October 2013.
- [BKB22] Jan Philip Bernius, Stephan Krusche, and Bernd Bruegge. Machine learning based feedback on textual student answers in large courses. *Computers and Education: Artificial Intelligence*, 3:100081, January 2022.
- [BMR<sup>+</sup>20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, et al. Language Models are Few-Shot Learners, July 2020.

- [CH10] A. Collins and R. Halverson. The second educational revolution: Rethinking education in the age of technology: The second educational revolution. *Journal of Computer Assisted Learning*, 26(1):18–27, January 2010.
- [CTJ+21] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating Large Language Models Trained on Code, July 2021.
- [DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019.
- [DZY17] Fei Dong, Yue Zhang, and Jie Yang. Attention-based Recurrent Convolutional Neural Network for Automatic Essay Scoring. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 153–162, Vancouver, Canada, August 2017. Association for Computational Linguistics.
- [EP08] Stephen H. Edwards and Manuel A. Perez-Quinones. Web-CAT: Automatically grading programming assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE '08*, page 328, New York, NY, USA, June 2008. Association for Computing Machinery.
- [HT07] John Hattie and Helen Timperley. The Power of Feedback. *Review of Educational Research*, 77(1):81–112, March 2007.
- [JM23] Daniel Jurafsky and James H. Martin. Speech and Language Processing. Third Draft edition. Draft of January 7, 2023. <https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf>, January 2023.
- [KR16] Rezaul Kabir and Ismat Rahman. The Value and Effectiveness of Feedback in Improving Students’ Learning and Professionalizing Teaching in Higher Education. *Journal of Education and Practice*, 2016.
- [Kru21] Stephan (Dr ) Krusche. *Interactive Learning - A Scalable and Adaptive Learning Approach for Large Courses*. 2021.

## BIBLIOGRAPHY

---

- [Kru22] Stephan Krusche. *Semi-Automatic Assessment of Modeling Exercises Using Supervised Machine Learning*. January 2022.
- [KS18] Stephan Krusche and Andreas Seitz. ArTEMiS: An Automatic Assessment Management System for Interactive Learning. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 284–289, Baltimore Maryland USA, February 2018. ACM.
- [KSK<sup>+</sup>23] Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, Stephan Krusche, et al. ChatGPT for Good? On Opportunities and Challenges of Large Language Models for Education, January 2023.
- [Mü22] Technische Universität München. TUM in zahlen 2021. Technical report, Technische Universität München, München, 2022.
- [NMD06] David J. Nicol and Debra Macfarlane-Dick. Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. *Studies in Higher Education*, 31(2):199–218, April 2006.
- [Ope23] OpenAI. GPT-4 Technical Report, March 2023.
- [OWJ<sup>+</sup>22] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, et al. Training language models to follow instructions with human feedback, March 2022.
- [RJO19] Pedro Uria Rodriguez, Amir Jafari, and Christopher M. Ormerod. Language models and Automated Essay Scoring, September 2019.
- [RNSS] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training.
- [RSR<sup>+</sup>20] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

- [RWC<sup>+</sup>] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners.
- [Sch23] Paul Schwind. Generalizing machine-learning based assessments. Master’s thesis, Technical University of Munich, 2023.
- [SKGA17] Arjun Singh, Sergey Karayev, Kevin Gutowski, and Pieter Abbeel. Gradescope: A Fast, Flexible, and Fair System for Scalable Assessment of Handwritten Work. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*, pages 81–88, Cambridge Massachusetts USA, April 2017. ACM.
- [TCL21] Darren Turnbull, Ritesh Chugh, and Jo Luck. Learning management systems: A review of the research methodology literature in Australia and China. *International Journal of Research & Method in Education*, 44(2):164–178, March 2021.
- [TDFH<sup>+</sup>22] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. LaMDA: Language Models for Dialog Applications, February 2022.
- [Tea22] OpenAI Team. ChatGPT: Optimizing language models for dialogue. <https://openai.com/blog/chatgpt/>, Nov 2022. Accessed: 2023-02-05.
- [TMS] Hugo Touvron, Louis Martin, and Kevin Stone. Llama 2: Open Foundation and Fine-Tuned Chat Models.
- [VSP<sup>+</sup>23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, August 2023.
- [WHZ<sup>+</sup>15] Dongqing Wang, Hou Han, Zehui Zhan, Jun Xu, Quanbo Liu, and Guangjie Ren. A problem solving oriented intelligent tutoring system to improve students’ acquisition of basic computer skills. *Computers & Education*, 81:102–112, February 2015.
- [WSF<sup>+</sup>23] BigScience Workshop, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, et al. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model, June 2023.

## *BIBLIOGRAPHY*

---

- [ZCS<sup>+</sup>23] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, et al. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena, July 2023.