Master's Thesis

Academic Coursework Assessment and Feedback Generation with LLMs

Karthik Krishnan

Examiners: Ruben Nuredini, Ph.D.
Prof. Dr. Jörg Winckler

Hochschule Heilbronn
Fakultät Informatik
Software Engineering Master

March 11th, 2025

Writing period

11. 07. 2024 – 11. 03. 2025

Examiners

Ruben Nuredini, Ph.D.

Prof. Dr. Jörg Winckler

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Heilbronn, 10.03.2025

Place, Date

[illegible]

Signature

Abstract

In the realm of education, the evaluation of academic coursework is a vital but challenging task. Traditional grading methods, characterized by their time-consuming nature and subjectivity, have long been a staple of the educational process. However, the advent of advanced technologies presents an opportunity to revolutionize these methods, making them more efficient, consistent, and objective. This thesis aims to develop a prototype system for automated assessment of academic coursework, leveraging the capabilities of cutting-edge Language Models (LMs) such as OpenAI's GPT via API and locally hosted models like Llama 3.

The primary goal of this research is to design a versatile and flexible system capable of accurately assessing both coding assignments (e.g., Python scripts) and open-ended questions (e.g., essays). Traditional grading methods can often be inconsistent, influenced by individual biases and time constraints. By harnessing the power of advanced LMs, this project seeks to streamline the assessment process, ensuring that evaluations are not only rapid but also adhere to standardized criteria, thereby reducing potential biases.

The development of the system follows an iterative approach, beginning with the integration of OpenAI calls, the initial version serves as a foundational model, highlighting both the strengths and limitations of using a straightforward LM application. Subsequent iterations introduces LangChain to address identified challenges, offering enhanced efficiency and modularity. Further improvements are achieved through the integration of CrewAi, which brings additional accuracy and features. The culmination of this development process is the final version, which integrates a user-friendly Streamlit UI, alongside CrewAi, LangChain, and Neo4J for robust data handling and visualization capabilities.

Validation of the system involves rigorous testing with mock data designed to simulate real academic assignments. This will ensure the accuracy of the system's assessments and the effectiveness of the feedback provided to students. Additionally, the project addresses critical aspects of data management, including the creation of a secure data set to store the evaluation results, with a strong emphasis on data

anonymization and ethical handling.

By exploring LM-based automated assessment, this research aims to enhance the efficiency and consistency of grading processes in education. The findings are expected to make significant contributions to the advancement of automated assessment systems, offering potential improvements in how educational institutions manage grading and feedback. Furthermore, the outcomes will provide valuable insights for future developments in LM technology for educational purposes, addressing key issues of scalability, usability, and ethical considerations.

In summary, this thesis project not only investigates the application of advanced technology in education but also addresses practical challenges faced by educators in modern learning environments. By developing a prototype system for automated assessment, this research endeavors to pave the way for more efficient, objective, and scalable grading solutions.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Motivation

Automated assessment systems have become indispensable in the modern educational landscape, addressing the need for efficient and accurate evaluation of student submissions. Traditional grading systems often struggle with scalability, consistency, and personalization, particularly in large-scale educational settings. The integration of advanced Artificial Intelligence (AI) technologies provides an opportunity to revolutionize these processes by offering sophisticated grading and feedback mechanisms.

This thesis explores an innovative approach to automated assessment by employing an agentic system built using CrewAI and LangChain, combined with a graph-based storage solution (Neo4j). Unlike conventional rule-based grading systems, this agentic system dynamically evaluates responses, adapting to different answer formats and levels of complexity. By leveraging Large Language Models (LLMs) and modular agent-based architecture, the system ensures both scalability and adaptability to diverse educational contexts.

The motivation behind this project stems from the growing demand for AI-driven educational tools that enhance learning outcomes while reducing the burden on educators. This approach aims to bridge the gap between traditional grading methodologies and modern AI-driven assessment systems, offering a more accurate, unbiased, and scalable solution.

1.2 Problem Statement

Existing automated grading systems face several critical limitations that hinder their effectiveness in large-scale educational settings. One of the most significant challenges is their lack of flexibility. Many current systems rely on rigid, predefined rules that assess responses based on strict keyword matching or pattern recognition. This approach fails to account for the natural variations in student answers, where different wording, phrasing, or reasoning paths may still lead to a correct response. As a result, students who express their understanding in a non-standard manner may receive inaccurate or unfair grades, limiting the adaptability of such systems across diverse educational contexts.

Another major shortcoming of traditional grading systems is their inability to provide meaningful and constructive feedback. Most existing solutions focus solely on assigning scores without offering explanations or insights into students' mistakes. Effective learning requires not just evaluation but also personalized feedback that helps students understand their errors and improve their knowledge. Without this aspect, automated grading systems become mechanical tools that assess correctness without facilitating learning. This lack of personalization further limits their effectiveness, as students with different levels of comprehension may require tailored guidance to address their specific weaknesses.

Additionally, many current grading systems struggle with handling complex data relationships, particularly in maintaining grading integrity and traceability. In large-scale educational environments, it is crucial to store and manage information about students' past performances, grading rubrics, and evolving evaluation criteria. Without a structured way to organize and retrieve this data, inconsistencies in grading may arise, leading to fairness concerns. A well-designed system should ensure transparency, allowing educators to track how a particular grade was assigned and providing explanations if disputes arise.

Beyond these technical challenges, human grading itself presents several inherent biases that can affect the fairness of assessments. Subjective judgment plays a significant role in traditional grading, as different evaluators may interpret the same answer in varying ways depending on their personal experiences, knowledge, and expectations. This subjectivity introduces inconsistency, where one instructor may grade more leniently while another may be stricter, leading to discrepancies in student outcomes.

Furthermore, the grading process is highly susceptible to fatigue and cognitive overload, especially when educators must evaluate large volumes of submissions within limited time frames. As graders become exhausted, their ability to maintain consistent standards diminishes, resulting in unintentional variations in scoring. In some cases, instructors may subconsciously adjust their grading based on prior performance, showing leniency to students who have performed well in the past or being harsher on those who have struggled.

Implicit biases also play a significant role in human grading. Factors such as a student's handwriting, writing style, or even their identity may influence the grader's perception, consciously or subconsciously affecting their assessment. This issue is particularly relevant in subjective assessments, such as essay writing, where personal preferences may shape grading decisions. Such biases, whether intentional or not, undermine the fairness of the grading process and can negatively impact students' academic progress.

Given these challenges, there is a strong need for a grading system that addresses these deficiencies by incorporating advanced AI-driven methodologies. An ideal system should combine natural language understanding, intelligent task management, and structured data storage to ensure accurate, scalable, and unbiased assessments. By leveraging AI technologies, it becomes possible to create an assessment framework that minimizes human biases, enhances grading transparency, and delivers personalized feedback, ultimately improving the overall educational experience for both students and educators.

1.3 Research Objectives

The goal of this thesis is to design and develop a system that addresses these challenges by:

1. Automating the grading process using an agentic system that can evaluate student answers against predefined grading rubrics.
2. Providing personalized feedback to students to enhance their learning experience.
3. Utilizing a database to manage complex relationships between data entities such as questions, answers, and grading criteria.
4. Ensuring scalability, explainability, and efficiency in processing student submissions across diverse subjects and educational contexts.

While automation improves efficiency, human oversight remains crucial in grading processes. This system is designed to assist, not replace, human graders by:

- Reducing workload: Automating repetitive grading tasks, allowing educators to focus on higher-order assessments.
- Providing consistency: Ensuring uniform evaluation across all students without bias.

- Enhancing decision-making: Allowing human graders to review AI-generated feedback and make final adjustments when necessary. We have a system where the grades and Feedback can be graded manually by examiners and this grading maybe used to improve the system in the future.
- Improving feedback quality: Generating detailed, structured, and personalized feedback that educators may not have time to write manually.

By combining AI-driven automation with human oversight, this system provides a balanced and efficient grading solution, ultimately enhancing both educational fairness and student learning outcomes.

2 Related Work

2.1 Early Approaches

In the initial phases of automated grading system development, rule-based approaches were predominant. These systems relied on a set of predefined rules and heuristics to evaluate and assign grades. Typically, rule-based systems utilized keyword matching, pattern recognition, and syntactic analysis to assess the correctness and quality of student responses. For example, a rule-based system designed to grade essays might look for the presence of specific keywords, logical argument structures, and correct grammar usage.

While rule-based systems offered significant improvements in grading efficiency compared to manual processes, they had notable limitations. These systems often struggled to understand the context and deeper meaning of student responses, leading to challenges in accurately grading more nuanced, creative, and unconventional answers. Moreover, they required constant updates and maintenance to incorporate new grading criteria and educational standards.

Alongside rule-based systems, early artificial intelligence (AI) models began to emerge as a promising alternative for automated grading. These models used machine learning techniques, such as decision trees, linear regression, and simple neural networks, to mimic human judgment in grading. By training on datasets of previously graded assignments, early AI models could learn to recognize patterns and make grading decisions based on the examples provided.

One of the earliest AI applications in grading was the Intelligent Essay Assessor (IEA), developed in the 1990s, which used latent semantic analysis to evaluate the content of student essays. Similarly, projects like Project Essay Grade (PEG) explored the use of multiple regression models to predict grades based on various linguistic features.

Despite their innovative approaches, early AI models were limited by the computational power and data availability of their time. The models often required extensive training data to achieve reliable performance, and their interpretability was sometimes questionable, making it difficult for educators to trust their grading decisions fully.

The combined efforts of rule-based systems and early AI models laid a critical foundation for the development of more advanced automated grading technologies. These early approaches demonstrated the potential benefits of leveraging technology to streamline educational processes, such as reducing grading time and providing consistent and objective evaluations.

While rule-based systems highlighted the importance of clear and well-defined grading criteria, early AI models emphasized the need for adaptability and learning from examples. The insights gained from these initial efforts have informed the design of modern, sophisticated grading systems that integrate natural language processing, deep learning, and other advanced AI techniques.

2.2 Recent Advances

Recent years have seen significant advancements in automated grading systems, driven by the integration of sophisticated artificial intelligence (AI) and machine learning (ML) techniques. These modern systems are designed to provide more accurate, reliable, and nuanced assessments of student work. One of the most notable advancements is the use of Natural Language Processing (NLP) and deep learning models. These technologies enable automated systems to better understand and evaluate the content, context, and coherence of student responses. For instance, models like BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) have been adapted for educational purposes to analyze essays and short answers more effectively

2.3 Gaps in Existing Systems

Recent years have seen significant advancements in automated grading systems, driven by the integration of sophisticated artificial intelligence (AI) and machine learning (ML) techniques. Natural Language Processing (NLP) and deep learning models are particularly noteworthy, enabling automated systems to better understand and evaluate the content, context, and coherence of student responses. For example, models like BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) have been adapted for educational purposes to analyze essays and short answers more effectively. Additionally, modern automated grading systems emphasize providing personalized feedback to students. By leveraging AI, these systems offer tailored suggestions and corrections, helping students understand their mistakes and improve their performance. This personalized approach enhances the learning experience and supports educators in managing larger classes more efficiently.

Despite these advancements, several gaps remain between current AI systems like ChatGPT and their effective use in the educational sector. One major challenge is ensuring bias and fairness, as AI systems can inadvertently perpetuate biases present in their training data, leading to unfair grading or feedback. Another significant issue is contextual understanding; while AI models have made great strides in understanding language, they often struggle with the nuanced context of educational content. Grading complex subjects like literature or history requires a deep understanding of context, which AI systems may lack.

Moreover, achieving true personalization is challenging given the diverse learning styles and needs of students. Ensuring that AI systems provide personalized feedback and support requires sophisticated models that can adapt to individual student profiles. Transparency and interpretability are also critical concerns, as many AI models are seen as "black boxes" with limited interpretability, making it difficult for users to trust their outputs. Ethical and privacy concerns further complicate the use of AI in education, particularly regarding the handling of student data. Ensuring that student data is used responsibly and securely is crucial to maintaining trust in AI systems.

Integration with existing educational infrastructures presents another challenge, as schools and institutions may lack the technical expertise or resources to effectively implement and maintain AI solutions. Accessibility is also a significant issue, with the need to ensure that AI educational tools are accessible to all students, including those with disabilities or from underprivileged backgrounds. Bridging the digital divide is essential to prevent further educational inequities.

Addressing these gaps requires ongoing research, collaboration between educators and technologists, and a commitment to ethical AI practices. By tackling these challenges, we can work towards creating more effective and equitable AI-driven educational systems.

2.4 Potential of agentic systems

To address the identified gaps in current AI educational systems, we propose a novel multi-agent approach that leverages the strengths of various AI agents working collaboratively. This approach aims to enhance the efficiency, accuracy, and fairness of automated grading and feedback systems, while also providing personalized support to students.

One of the core innovations of our multi-agent approach is the handling of query transmission to provide single-shot answers. Unlike traditional AI systems that often require multiple queries to arrive at a comprehensive response, our approach uses multiple specialized agents to process different aspects of a query simultaneously. These agents work in tandem to analyze the query context, extract relevant information, and generate a cohesive and accurate response in a single interaction. Each agent in the system is specialized in a particular domain or task, such as natural language processing, contextual understanding, bias detection, or personalized feedback generation. By dividing tasks among specialized agents, we can ensure that each aspect of the grading and feedback process is handled by the most suitable AI model. The collaboration between these agents allows for a more holistic and robust evaluation of student work, addressing the limitations of single-agent systems.Our multi-agent system significantly improves contextual understanding by integrating agents with advanced NLP capabilities and domain-specific knowledge. These agents can better grasp the nuances of student responses, especially in subjects that require deeper contextual insights, such as literature and history. This enhanced understanding leads to more accurate and fair grading.

3 Background

The Agentic System developed for this thesis integrates a variety of technologies to create a robust and efficient platform. The system relies on role-based agents that autonomously perform tasks such as grading and providing feedback, ensuring that each agent can focus on specific operations while collaborating with others to achieve complex goals. To facilitate the orchestration of these agents, LangChain provides a framework for seamlessly connecting different models and services, enabling intelligent decision-making and task management. For data management, Neo4j serves as the graph database, offering a flexible and scalable solution for storing and processing relationships between various entities in the system. The user interface is built using Streamlit, providing an interactive and intuitive front-end for users to interact with the system and upload data. This combination of tools, including CrewAI for agent management and OpenAI's models for language processing, forms the backbone of the system, ensuring its functionality, scalability, and ease of use.

3.1 Technological Stack

3.1.1 OpenAI

This project utilizes OpenAI[1] Python package to automate grading and feedback generation. The workflow involves careful model selection, structured API interactions, and response processing to ensure accurate and consistent evaluation.

Model Selection

The system initially utilized GPT-3.5-turbo during the prototyping phase due to its speed and cost-effectiveness. However, for the final implementation, GPT-4o was selected due to its improved reasoning capabilities, higher accuracy, and better contextual understanding, ensuring a more reliable grading and feedback process.

API Setup

The system interacts with OpenAI's API using an API key for authentication, which is securely managed to prevent unauthorized access. The OpenAI Python package is used to make requests efficiently, ensuring smooth communication with the model.

Prompt Engineering

Carefully designed prompts play a crucial role in ensuring unbiased and structured grading. The system dynamically integrates user responses into pre-defined templates, which guide the model to evaluate answers consistently based on predefined grading rubrics. This structured approach reduces variability and ensures objective assessment.

Message Roles

The API follows a conversational message structure, where each message is assigned a distinct role to ensure clarity and consistency throughout the process. The System Role provides overarching instructions that guide the model's behavior, establishing the framework for evaluation. The User Role contains the student's response, which is passed to the model for grading and feedback generation. The Assistant Role represents the model's output, including the assigned grade and feedback. By maintaining this structured format, the responses remain organized, consistent, and traceable throughout the entire grading process. This approach helps ensure that each step in the evaluation is clear and reproducible.

Temperature Control

The temperature parameter is set to 0.7, introducing a balance between consistency and variation in feedback. A lower temperature (e.g., 0.2) would make responses more deterministic and rigid, while a higher value (e.g., 1.0) would increase randomness. The 0.7 setting ensures feedback remains logical yet dynamic, allowing for nuanced and context-aware responses without excessive unpredictability.

This structured approach ensures that the grading and feedback system remains robust, scalable, and adaptable to different assessment types.

3.1.2 Ollama

Ollama[2] is an open-source framework that provides an efficient and customizable way to run large language models (LLMs) locally, such as Llama3.1:7b and DeepSeekR1:8b.

The decision to use Ollama stems from the need for enhanced data privacy, cost efficiency, and customization in running advanced models locally.

- Data Privacy: By running models locally on your machine, you ensure that sensitive data remains private and is not transmitted over the internet, thus preventing potential data leaks or security risks often associated with cloud-based solutions.
- Customization: Ollama enables users to have full control over their model execution. You can easily modify model configurations, interact with the models in specific ways, and experiment with new setups or fine-tuning techniques without the constraints imposed by cloud services. For CrewAI, we created a Makefile tailored for the Crew Models and integrated Ollama into the setup.
- Cost Efficiency: Cloud-based LLMs can be costly, especially when running large models at scale. Ollama provides a more economical approach by allowing users to run these models locally, significantly reducing long-term costs related to cloud resources.
- Local Execution and Flexibility: With Ollama, running models like Llama3.1:7b and DeepSeekR1:8b locally ensures quicker access to model inference, avoids network latency, and provides a flexible environment for research, experimentation, and development.

3.1.3 LangChain

LangChain[3] is a powerful Python package and ecosystem designed to simplify the process of building AI applications by providing standardized interfaces for various components, orchestration, and observability. It enables developers to easily integrate multiple models and tools, regardless of the provider, allowing for seamless interactions and complex workflows. LangChain focuses on key areas such as simplifying the integration of large language models (LLMs), providing flexible orchestration, and offering robust tools for monitoring and evaluation. These features make it an ideal choice for building AI-driven applications.

- Standardized Component Interfaces: LangChain provides standardized interfaces for critical components such as chat models and retrievers, which allowed me to easily integrate models into my application. The 'BaseChatModel' interface is particularly useful for enabling structured outputs and tool calling, which is vital for processing and interacting with grading and feedback data in a consistent way.

- Efficient Orchestration: LangChain's ability to orchestrate and manage various components through a common framework made it easier to design the complex workflows required in my thesis. This orchestration helps maintain the interaction between different models and agents, ensuring smooth task distribution, grading, and feedback processes.

In summary, LangChain's model integrations, standardized interfaces, and evaluation tools were essential in building a robust agentic System for my thesis, providing an ideal framework to manage the grading and feedback tasks with high efficiency and accuracy.

3.1.4 CrewAI

CrewAI [4] is the ideal solution for my Agentic System due to its ability to create specialized, role-based agents that collaborate efficiently to accomplish complex tasks. Each agent in CrewAI can have their own specified operations, tasks, and roles, which perfectly aligns with the needs of my system, where different agents are responsible for distinct functions like grading and providing feedback. CrewAI's flexible tools allow agents to interact seamlessly with external services and data, such as Neo4j for processing, while managing dependencies between tasks. This autonomous operation ensures that each agent makes intelligent decisions based on its role and available tools, streamlining the workflow. Additionally, its scalable and production-ready design guarantees reliability and scalability in real-world applications, making CrewAI an essential component of my thesis system.

3.1.5 Neo4j

For storing and processing complex relationships in data, Neo4j[5] was chosen as the graph database solution for this system. Unlike traditional relational databases, which store data in tables, Neo4j organizes data as nodes, relationships, and properties, offering a more intuitive way to model interconnected information. In this system, nodes represent entities, such as students or answers, and relationships define how these nodes are connected (e.g., student-to-answer relationships). Each node and relationship can also have properties that store additional information, such as student names or answer content.

Neo4j's flexible and scalable graph structure is particularly advantageous for situations where the size of data is not predetermined, such as student answers in this system. Graph databases, unlike relational databases, do not require fixed schemas, allowing for dynamic growth and adaptability over time. This flexibility makes Neo4j ideal for applications where data models evolve frequently and unpredictably. Additionally, Neo4j's powerful AI integrations enable advanced features, such as context-based search within the database, allowing the system to efficiently retrieve and analyze complex connections between data points.

Moreover, Neo4j's graph structure allows for the efficient exploration of deep hierarchies and the discovery of hidden relationships between distant data points. This capability is essential for analyzing student submissions, where connections between different answers, grades, and feedback may not be immediately obvious. Neo4j's native handling of relationships, without the need for computationally expensive JOIN operations, makes it possible to quickly traverse vast amounts of interconnected data, even when dealing with billions of nodes. This unique structure ensures high performance, especially as the dataset grows larger, providing an optimal solution for the Agentic System's evolving needs.

3.1.6 Streamlit

For the user interface (UI) of the Agentic System, Streamlit[6] was selected due to its simplicity and efficiency in quickly building interactive applications. Streamlit allows for easy creation of UIs directly in Python, which is especially beneficial for the development and testing of machine learning or AI systems like the one in this thesis. Its built-in UI components, such as buttons, sliders, and file uploaders, allow for intuitive user interaction with the system, such as uploading files for processing without requiring complex frontend development.

The primary reason for using Streamlit is its ease of building and deploying a user interface with minimal code. Streamlit's seamless integration with Python code accelerates development time and simplifies the process of connecting the user interface to the underlying system logic. The built-in methods provided by Streamlit further simplify the creation of dynamic content, ensuring a smooth interaction between users and the agentic system without the need for additional UI frameworks.

4 Methodology

The primary objective of this research is to develop an automated grading and feedback system that evaluates student answers against provided master answers and grading schemes. The system aims to produce grades and feedback that closely align with the assessments provided by examiners.

4.1 System Development Process

The development of the grading and feedback system progressed through multiple iterations, each aimed at addressing limitations identified in the previous version and enhancing overall performance and efficiency.

Version 1.0: Initial Approach with OpenAI Calls

The initial setup utilized direct OpenAI calls to handle various tasks, leveraging the powerful language processing capabilities of the model. While this approach provided a solid foundation, several issues surfaced. These included response latency, higher operational costs, and functional limitations, particularly concerning processing efficiency. The system faced challenges with token utilization and a narrow context window, which hindered its ability to handle complex or lengthy student responses effectively.

Version 2.0: Enhancing with LangChain

To overcome the limitations observed in Version 1.0, LangChain was integrated into the setup. This addition enabled a more efficient and modular framework that significantly reduced latency and optimized the use of tokens and context windows. The improved framework allowed for faster processing times and reduced resource consumption, enhancing the system's capacity to handle multiple student submissions without compromising performance. Additionally, LangChain facilitated better management and customization of language processing tasks, resolving many of the initial concerns.

Version 3.0: Integrating CrewAI with LangChain

Building upon the enhancements introduced in the previous version, CrewAI was integrated into the system. This integration introduced a structured, agentic system that significantly improved task distribution and processing efficiency. Specialized agents were assigned distinct roles, which optimized the processing pipeline and reduced redundant operations. CrewAI's task distribution capabilities allowed the system to scale efficiently while maintaining high accuracy and reliability.

Version 4.0: Final Setup with Streamlit UI, CrewAI, LangChain, and Neo4J

The final iteration of the system integrated a comprehensive setup that combined Streamlit for UI, CrewAI, LangChain, and Neo4J. Streamlit provided a user-friendly and intuitive interface, making the system accessible for users while simplifying data input and review processes. Neo4J was introduced for enhanced data handling capabilities, offering robust storage and advanced querying functionalities to support flexible data relationships and scalability.

This comprehensive setup not only streamlined the user experience but also delivered powerful data handling and processing capabilities. Neo4J's advanced features enabled efficient storage of complex data structures, while CrewAI and LangChain ensured accurate and scalable processing. Each iteration built upon the strengths of its predecessor, refining the system to enhance efficiency, accuracy, and user experience. This iterative development approach resulted in a versatile, high-performance system capable of managing a wide range of grading and feedback tasks effectively.

4.2 Evaluation Process

The evaluation process involved iterative testing and refinement. Initially, prototypes were tested using sample data to generate relevant feedback and grades. Based on these results, adjustments were made to optimize performance, culminating in the final agentic system.

Once the final system was established, evaluation proceeded with a controlled sample of 60 student answers, each accompanied by the corresponding solution. The evaluation was structured in two parts:

1. Model Comparison: Grades and feedback were generated using three models: GPT-4.o, Llama3.1:8b, and DeepSeekR1:8b and results were compared to determine the most effective model.

2. Consistency and Hallucination Testing: To ensure reliability, a subset of five randomly selected questions was iterated ten times using the most capable

model. This process aimed to verify consistency and detect any hallucinations or variability in responses.

4.3 Data Collection

Question Types: The dataset consisted of two main types of questions:

1. Descriptive Questions: These required detailed, explanatory answers that were evaluated for depth, understanding, and clarity.
2. Programming Questions: These involved writing code solutions, where evaluation focused on correctness, logic, and adherence to problem requirements.

Data collection focused on both quantitative and qualitative measures:

- Quantitative: The grades generated by the system were compared against the original grades provided by the professor to measure accuracy. The primary metric for quantitative analysis was the closeness of system-generated grades to the original grades provided by the professor.
- Qualitative: A qualitative survey was conducted, where professors reviewed 60 generated feedback samples.Professors rated each feedback sample using a Likert scale (ranging from 1 - strongly disagree to 5 - strongly agree) for each criterion.

4.4 Validation and Testing

Validation Process: The validation of the system involved a rigorous comparison of its outputs against manual grading results and expert evaluations. The primary goal was to ensure that the system's grading and feedback mechanisms aligned closely with human judgment. An iterative testing approach was employed, where initial system outputs were thoroughly reviewed and analyzed. Based on identified discrepancies or inaccuracies, refinements were made to the grading logic, prompt design, and feedback generation process. This cycle of testing and refinement continued until the system demonstrated consistent accuracy and reliability. Expert feedback played a crucial role during this phase, with professors providing insights into grading nuances and offering suggestions for enhancing feedback clarity and relevance. This iterative approach ensured that the system evolved continuously, improving its alignment with expert expectations and academic standards.

After generating the grades, to assess the effectiveness and reliability of the proposed system, three statistical metrics—Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Standard Deviation (SD)—were employed. These metrics provide a quantitative evaluation of the variances between the True Grade and the grades predicted by the system.

- MAE highlights the average magnitude of error, offering a measure of the model's overall accuracy without considering the direction of the errors.
- RMSE, by penalizing larger deviations more heavily, reveals the impact of significant outliers in the model's predictions.
- SD captures the variability of the errors, reflecting the consistency of the model's predictions relative to its mean deviation.

Controlled Testing and Consistency Checks:
To validate the system's reliability, controlled testing was conducted using a sample of 60 student answers, covering both descriptive and programming question types. These samples are graded by the system and then compared with manually graded results to evaluate accuracy and consistency. Special attention is given to consistency checks, where repeated testing was carried out to verify the system's response stability. This involves running the same prompts multiple times to ensure that outputs remained consistent and that the system minimized hallucinations—unfounded or incorrect information generated by the model. Any observed inconsistencies trigger further analysis and refinement to enhance stability. This rigorous validation process strengthens the system's reliability and ensures its capability to deliver consistent and accurate outputs across diverse question types and complexity levels.

5 System Architecture

The system architecture for our project utilizes an agentic system to efficiently handle query transmission and provide single-shot answers. This architecture includes three specialized agents created using CrewAI and LangChain, a frontend interface built with Streamlit, and a graph database implemented with Neo4j. The following sections detail the design and functionality of each component and their interactions.

5.1 High-Level Overview

The system comprises the following main components:

- Agents: Three specialized agents created using CrewAI and LangChain.
- Front-End Interface: Developed with Streamlit.
- Database: Implemented with Neo4j.

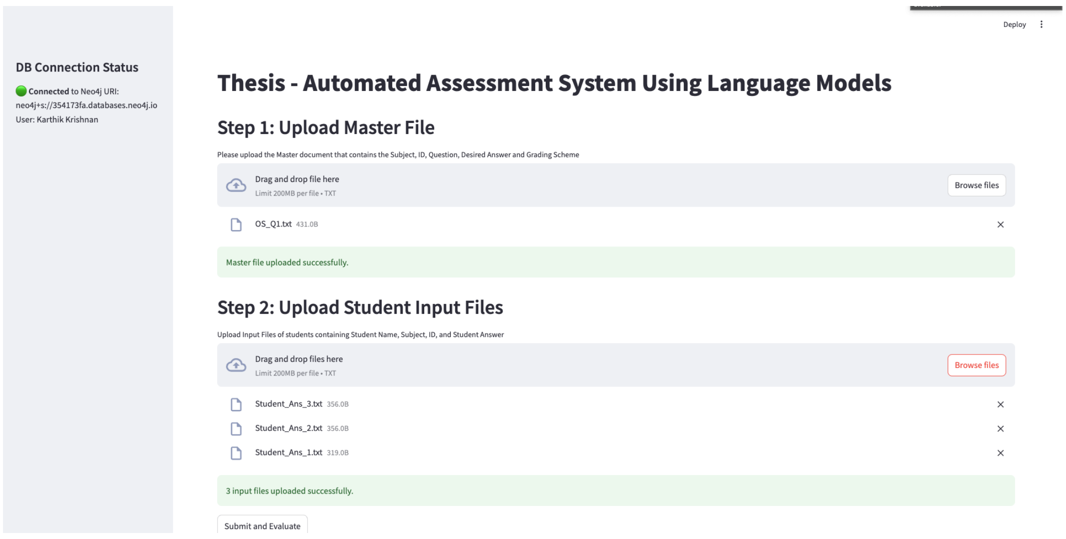
5.2 Detailed Component Descriptions

5.2.1 Input-Output Interfaces

The system is designed to handle data interactions seamlessly through a user-friendly interface built with Streamlit. This interface allows users to input data, which is subsequently processed and displayed back in the same environment. Streamlit's capabilities enable the creation of a responsive and interactive web application, ensuring efficient management of user inputs and outputs.

The interface is organized into two main sections: the left panel and the main panel. Each section serves specific purposes to streamline the data handling process.

The Left Panel focuses on system-level interactions. It includes the connection status indicator, which provides real-time updates on the system's link to the database, and user authentication features, ensuring secure access to the database for authorized users.


DB Connection Status

● **Connected** to Neo4j URI: neo4j+s://354173fa.databases.neo4j.io
User: Karthik Krishnan

Thesis - Automated Assessment System Using Language Models

Step 1: Upload Master File

Please upload the Master document that contains the Subject, ID, Question, Desired Answer and Grading Scheme

OS_Q1.txt 431.0B

Master file uploaded successfully.

Step 2: Upload Student Input Files

Upload Input Files of students containing Student Name, Subject, ID, and Student Answer

Student_Ans_3.txt 356.0B
Student_Ans_2.txt 356.0B
Student_Ans_1.txt 319.0B

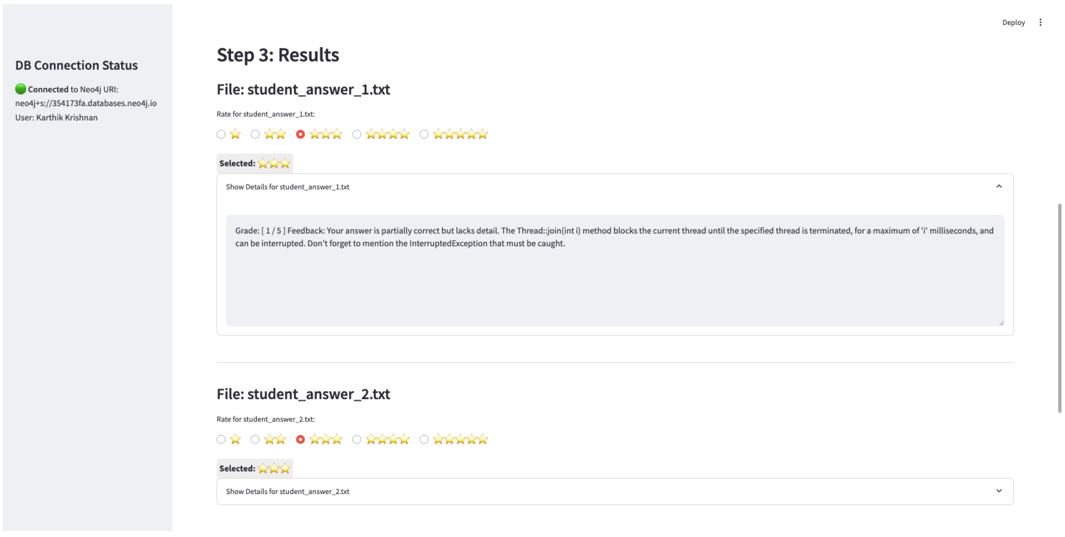
3 input files uploaded successfully.

Figure 1: Frontend UI - Upload Section

The Main Panel is dedicated to data uploads and comprises two distinct sections. The first section handles the upload of the Master Document, which serves as the reference framework for the system. This document outlines essential elements such as Subject, ID, Question, Desired Answer, and Grading Scheme. A visual representation of the schema for the Master Document is provided in Figure 1.

The second section facilitates the upload of student answers through a multiple-upload field. Each uploaded file is expected to conform to a predefined schema, including details of Student ID, Question, and Student Answer. This structured input ensures consistency and accuracy in processing.

The workflow is centered around leveraging the Master Document as a precedent. After uploading, the system processes the data by using the Master Document to evaluate student answers. The grading is performed according to the scheme defined within the Master Document, and the results are then collated and presented back to the user through the Streamlit interface shown in Figure 2. This transparent cycle allows users to interact effectively with the system, ensuring usability and reliability. There is also a Feedback towards the Graded answers which is necessary to ensure that the evaluations produced can also be graded for future inference. Overall, this interactive design minimizes user effort while maintaining a robust and efficient data processing mechanism.



DB Connection Status

● **Connected** to Neo4j URI: neo4j+s://354173fa.databases.neo4j.io
User: Karthik Krishnan

Step 3: Results

File: student_answer_1.txt

Rate for student_answer_1.txt:

○ ⭐ ○ ⭐⭐ [✓ ⭐⭐⭐] ○ ⭐⭐⭐⭐ ○ ⭐⭐⭐⭐⭐

Selected: ⭐⭐⭐

Show Details for student_answer_1.txt

Grade: [1 / 5] Feedback: Your answer is partially correct but lacks detail. The Thread::join(int i) method blocks the current thread until the specified thread is terminated, for a maximum of 'i' milliseconds, and can be interrupted. Don't forget to mention the InterruptedException that must be caught.

File: student_answer_2.txt

Rate for student_answer_2.txt:

○ ⭐ ○ ⭐⭐ [✓ ⭐⭐⭐] ○ ⭐⭐⭐⭐ ○ ⭐⭐⭐⭐⭐

Selected: ⭐⭐⭐

Show Details for student_answer_2.txt

Figure 2: Frontend UI - Results Section

5.2.2 Processing Units

The system architecture leverages a modular and agentic approach, where each processing unit (agent) specializes in distinct tasks to ensure a streamlined and efficient workflow. These agents collaborate using advanced AI frameworks such as LangChain and CrewAI, facilitating seamless integration and task delegation.

Integration of LangChain and CrewAI

LangChain:

LangChain serves as a foundational framework for integrating large language models (LLMs) into the system, enabling the agents to perform sophisticated natural language processing (NLP) tasks. This capability ensures seamless handling of text-heavy workflows, crucial for both grading and feedback generation.

- Natural Language Understanding: LangChain enables agents to accurately interpret and process student submissions, identifying key concepts, discrepancies, and alignment with the expected answers.

- Contextual Information Retrieval: The framework facilitates the extraction of relevant content from grading rubrics, model solutions, and other reference materials. This ensures that the agents have the necessary context to make accurate evaluations and generate meaningful outputs.

- Content Generation and Structuring: LangChain helps in formatting feedback in a clear, concise, and student-friendly manner. It ensures that the feedback is both pedagogical and personalized, adapting its tone and depth to the needs of each student.
- Scalability in NLP Tasks: By leveraging pre-trained LLMs, LangChain allows for a flexible and scalable solution, accommodating different subject domains and varying complexity levels in student responses.

CrewAI:
CrewAI acts as the central communication and coordination layer, ensuring smooth collaboration between the agents and optimizing the overall system workflow. Its focus lies in enhancing the efficiency and transparency of task execution.

- Efficient Task Distribution: CrewAI assigns tasks dynamically to the agents based on their roles and current workloads. This ensures optimal resource utilization and prevents bottlenecks during peak loads.
- Agent-to-Agent Communication: The platform facilitates seamless communication between the agents, allowing them to exchange necessary data without redundancies. For example, the Grading Expert Agent can share its evaluation results directly with the Feedback Expert Agent, speeding up the feedback generation process.
- Dynamic Adaptability: CrewAI supports the scalability of the system by dynamically adjusting to variations in workload. As the number of student submissions increases, it ensures that agents are efficiently allocated without compromising performance.
- Transparency and Monitoring: CrewAI provides a clear view of task progress and agent interactions, allowing the Manager Agent to make informed decisions and intervene when necessary.

Agent Roles and Responsibilities

Expert Tutor Agent

The Manager/Captain Agent acts as the central orchestrator of the grading and feedback system. This agent ensures the coordinated execution of all grading-related tasks by effectively managing and delegating responsibilities to specialized agents.

It synthesizes the results from these agents to provide a comprehensive and unified output for each student..

Responsibilities:

- Acts as the central coordinator, ensuring the smooth execution of grading and feedback tasks.
- Delegates responsibilities to specialized agents (Grading Expert Agent and Feedback Expert Agent).
- Collates and integrates the results from other agents, providing a unified output for each student submission.

Grading Expert Agent

The Grading Expert Agent is dedicated to the thorough and precise evaluation of student submissions. This agent meticulously applies a predefined grading scheme to assess student responses, ensuring fairness, accuracy, and consistency in the grading process. It is responsible for interpreting the grading criteria and assigning appropriate grades based on the quality and correctness of each submission.

Responsibilities:

- Applies a predefined grading scheme to assess student responses.
- Ensures accuracy and consistency in grading by following established criteria.

Feedback Expert Agent

The Feedback Expert Agent specializes in delivering detailed, personalized, and constructive feedback to students. By closely analyzing the discrepancies between student answers and ideal solutions, this agent generates insightful feedback tailored to each student's unique needs. The aim is to help students understand their mistakes, provide guidance for improvement, and enhance their overall learning experience.

Responsibilities:

- Analyzes discrepancies between student answers and the ideal solutions.
- Generates insightful feedback that is tailored to the individual student's response, aimed at improving their understanding and performance.

5.2.3 Storage Solution

The Neo4j Database serves as the primary storage solution for the system, offering robust graph database capabilities tailored to managing complex relationships within

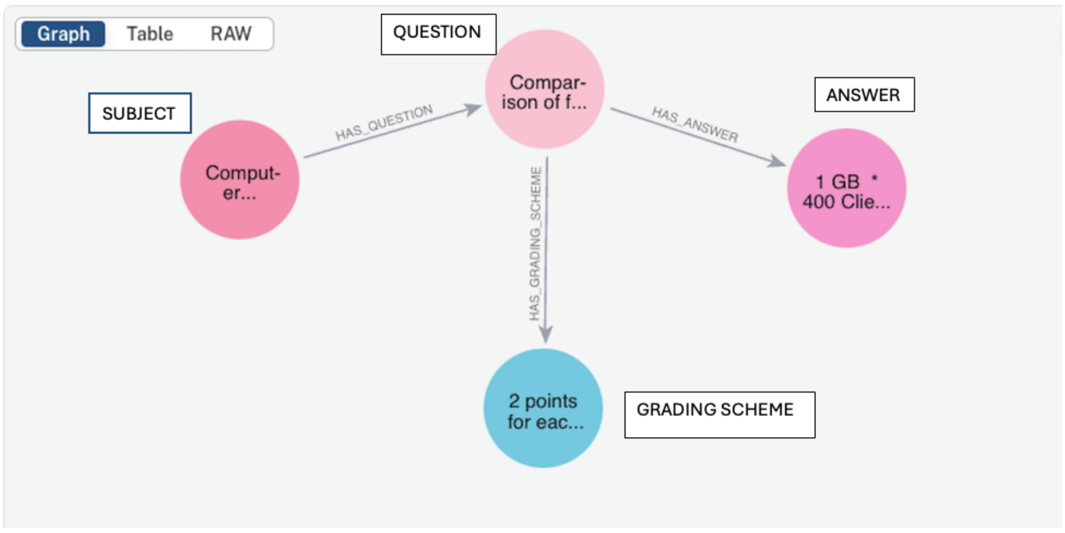


Figure 3: Sample Node

data. As a core component of the system architecture, Neo4j ensures efficient data handling, structured storage, and rapid retrieval.

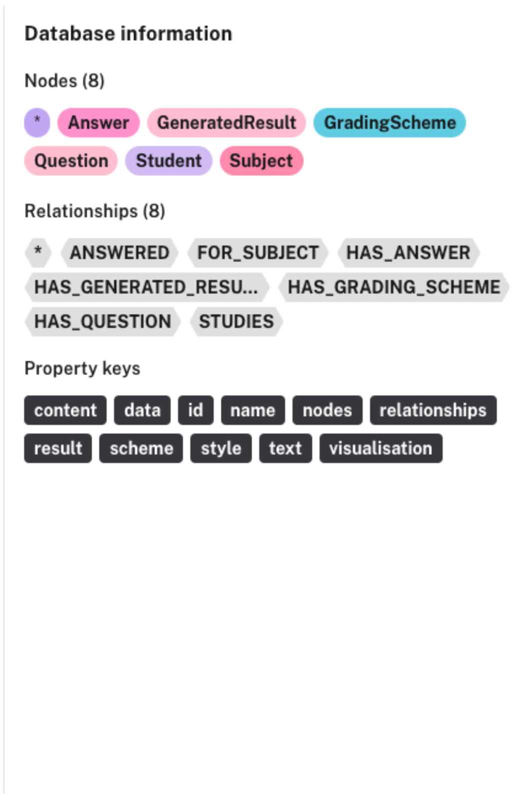
Data is stored in a structured format based on a predefined schema. Each Master entry in the system is converted into a graph node adhering to this schema, ensuring the integrity of the Master file and preventing unauthorized modifications. The system enforces this structure statically, creating a reliable framework for consistent data representation. Nodes are systematically categorized and labeled, facilitating seamless querying and traversal across the dataset.

For all nodes created, the GraphDB enables clear mapping by storing task details as nodes linked to corresponding agents. This ensures traceability of relationships between the Answer and corresponding ID and provides an intuitive visualization of agent responsibilities. When processing student inputs, the system evaluates each answer individually, producing distinct nodes for each response. These nodes are enriched with metadata, including feedback, timestamps, and grading parameters, creating a comprehensive record for every processed input.

Upon completion of processing, student response nodes are merged with their respective Question IDs using Cypher queries, Neo4j's powerful declarative query language. This integration establishes meaningful relationships between data points, ensuring cohesion and logical mapping within the graph structure.

This is a sample data Node for the graph database that is formed:

The primary nodes in this schema include entities such as **Answer**, **Gener-**



Database information

Nodes (8)

* Answer GeneratedResult GradingScheme
Question Student Subject

Relationships (8)

* ANSWERED FOR_SUBJECT HAS_ANSWER
HAS_GENERATED_RESU... HAS_GRADING_SCHEME
HAS_QUESTION STUDIES

Property keys

content data id name nodes relationships
result scheme style text visualisation

Figure 4: Database Information

atedResult, **GradingScheme**, **Question**, **Student**, and **Subject**. Each node represents a critical component within the database, contributing to its overall functionality.

The relationships between these nodes are meticulously defined, featuring types like **ANSWERED**, **FOR_SUBJECT**, **HAS_ANSWER**, **HAS_GENERATED_RESULT**, **HAS_GRADING_SCHEME**, **HAS_QUESTION**, and **STUDIES**. These relationships establish logical connections that facilitate effective data interaction and retrieval.

Moreover, the property keys associated with both nodes and relationships—such as **content**, **data**, **id**, **name**, **nodes**, **relationships**, **result**, **scheme**, **style**, **text**, and **visualization**—provide a structured and detailed representation of the dataset. This comprehensive schema is crucial for understanding the organization and dynamics of the database, thus enabling efficient data management and analysis within the context of this thesis.

5.3 Data Flow

The flowchart, Figure 5, illustrates the backend control flow of a sophisticated system designed to integrate a Large Language Model (LLM) with a Neo4j graph database, aiming to efficiently manage and process student data and their academic assessments.

Initialization Phase:

- Initialize the LLM and Environment Variables: The system initiates by loading the Large Language Model and setting up essential environment variables to ensure seamless operation.
- Establish Connection with Neo4j: A secure connection is established with the Neo4j graph database instance to enable data storage and retrieval.
- Start Streamlit UI: The user interface is launched using Streamlit, providing an interactive platform for user interaction.

Data Parsing Phase:

- Parse Master Data: The system processes and organizes master data, which includes static information such as subjects, desired answers, and grading schema.
- Parse Student Data: Student data is parsed to extract relevant attributes such as Student ID, submitted answers, and other pertinent information.

LLM Execution Phase:

- Execute LLM Flow: The Large Language Model is employed to understand and process the parsed data. This is accomplished using a crew chain, a sequence of tasks that are carried out to process the input data.
- Manager LLM Delegates Tasks: The manager LLM assigns specific tasks to various agents within the system. These tasks are essential for processing the data and generating desired outputs.

Grading and Feedback Phase:

- Grading Agent: A designated agent within the system evaluates the students' submitted answers against the grading schema, generating grades for each student.

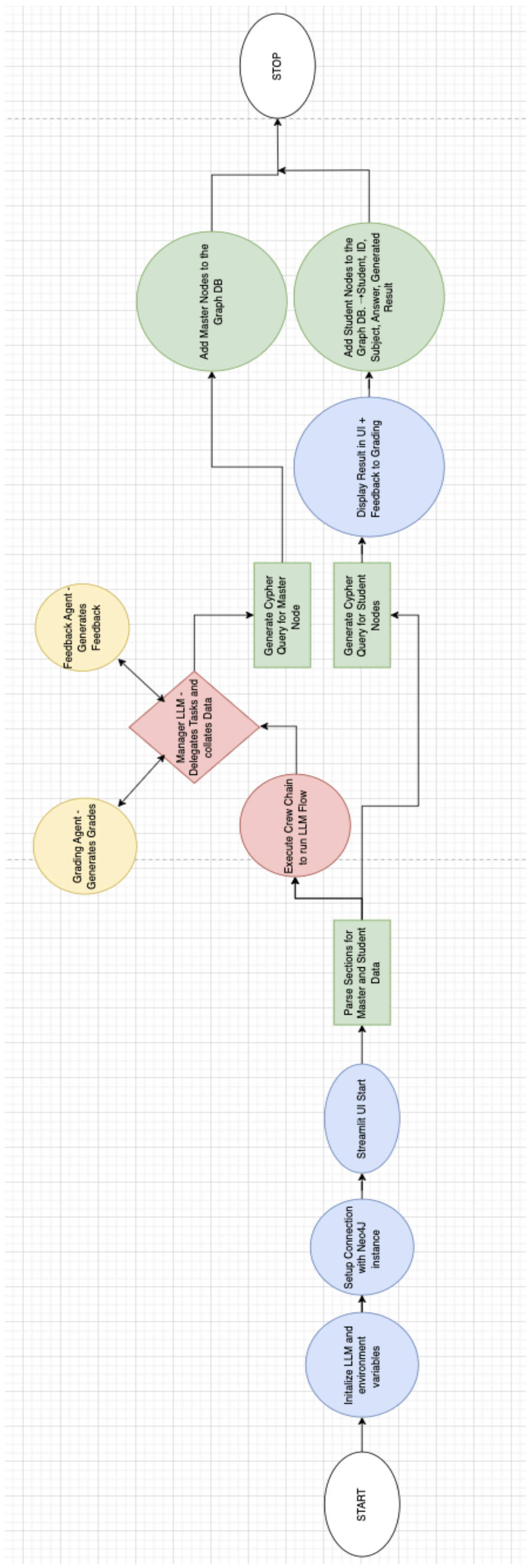

START
Initalize LLM and environment variables
Setup Connection with Neo4J instance
Streamlit UI Start
Parse Sections for Master and Student Data
Execute Crew Chain to run LLM Flow
Manager LLM - Delegates Tasks and collates Data
Grading Agent - Generates Grades
Feedback Agent - Generates Feedback
Generate Cypher Query for Master Node
Generate Cypher Query for Student Nodes
Display Result in UI + Feedback to Grading
Add Master Nodes to the Graph DB
Add Student Nodes to the Graph DB. →Student, ID, Subject, Answer, Generated Result
STOP

Figure 5: Backend Data Flow

- Feedback Agent: Another agent is responsible for providing detailed feedback on the students' performance, highlighting strengths and areas for improvement.

Graph Database Integration Phase:

- Generate Cypher Query for Student Nodes: The system generates a Cypher query to add student nodes to the Neo4j graph database. Each student node contains attributes such as Student ID, Subject, Answer, and Generated Result.
- Generate Cypher Query for Master Nodes: Similarly, the system generates a Cypher query to add master nodes to the graph database. These nodes include attributes like Subject, Desired Answer, and Grading Schema, representing the static master data.

Completion Phase:

- Data Storage and Retrieval: The processed data, including student grades and feedback, is stored in the Neo4j graph database, making it accessible for future queries and analysis.
- System Termination: The system ensures that all processes are completed successfully before terminating operations, ready to be initiated again for subsequent data processing tasks.

5.4 Technical Specifications

The table below outlines the technical specifications of the key software packages and their respective versions utilized in this project. Each package was selected with careful consideration of its compatibility, stability, and ability to support critical functionalities required for the system. These include tasks such as AI model development, efficient data processing, web scraping, environmental configuration, and database integration. The listed versions ensure reliable performance, adherence to modern standards, and seamless interaction among components, making them ideal for handling complex workflows in AI-driven applications. Additionally, the inclusion of community and experimental tools reflects the project's emphasis on leveraging cutting-edge innovations to enhance flexibility and functionality.

python	>=3.10.0,<3.12
crewai	0.11.0
crewai$_{t}ools$	*
python-dotenv	1.0.1
requests	2.31.0
beautifulsoup4	4.12.3
langchain-openai	0.0.5
langchain$_{o}penai$	*
langchain$_{c}ore$	*
langchain	*
langchain$_{c}ommunity$	*
faiss-cpu	1.7.1
streamlit	1.38.0
langchain-groq	*
langchain$_{g}roq$	*
langchain$_{e}xperimental$	*
neo4j	*

Table 1: Technical Specifications

6 Results

6.1 Evaluation Metrics

The performance of the grading and feedback system is assessed through a set of evaluation metrics that provide a comprehensive view of its effectiveness in grading accuracy, feedback quality, and processing efficiency. These metrics are designed to measure how well the system adheres to the expected standards and provide meaningful insight into its areas of strength and improvement.

6.1.1 Grading Accuracy

The accuracy of the grade is the cornerstone of the system's evaluation capabilities. It determines how closely the grades assigned by the system match the grades given by human evaluators, ensuring that the system can perform reliable assessments of student work. Grading accuracy is calculated by comparing the grades assigned by the system with the grades given by a set of expert human graders. In Version 1.0, grading accuracy was limited, especially in cases involving complex or borderline answers. However, with the introduction of more sophisticated algorithms in Version 2.0, such as improved natural language processing (NLP) via LangChain and more refined heuristics for grading, the accuracy rate improved substantially. After receiving satisfactory results in the prototype the final sample set was considered for checking grading accuracy across all three models, GPT-4.o, Llama3.1:8b and DeepSeekR1:8b

Descriptive Questions:

- GPT-4 showed moderate variance, with fluctuations ranging from -4 to 7. Despite some negative deviations, it maintained relatively consistent grading overall.
- Ollama3.1:8b exhibited higher variability, with scores ranging from -7 to 5, indicating occasional extreme deviations in grading.
- DeepSeekr1:8b displayed a broader variance range, from -5 to 8, highlighting both significant overestimations and underestimations in its grading.

Figure 5 represents the score variations between the descriptive questions and presents the score variations between 'True Grade' rated by the examiner against the system produced values for grade.

Programming Questions:

- GPT-4 displayed limited variance, with deviations mostly ranging between -6 to 11, indicating relatively consistent grading accuracy.
- Ollama3.1:8b showed the highest variance, with deviations from -4 to 18, suggesting inconsistency and frequent grading fluctuations.
- DeepSeekr1:8b had mixed performance, with variances from -3 to 13, showing occasional significant deviations.

Figure 6 represents the score variations between the programming questions and presents the score variations between 'True Grade' rated by the examiner against the system produced values for grade.

Please refer to Table 4 and 5 in the appendix for complete grade values.

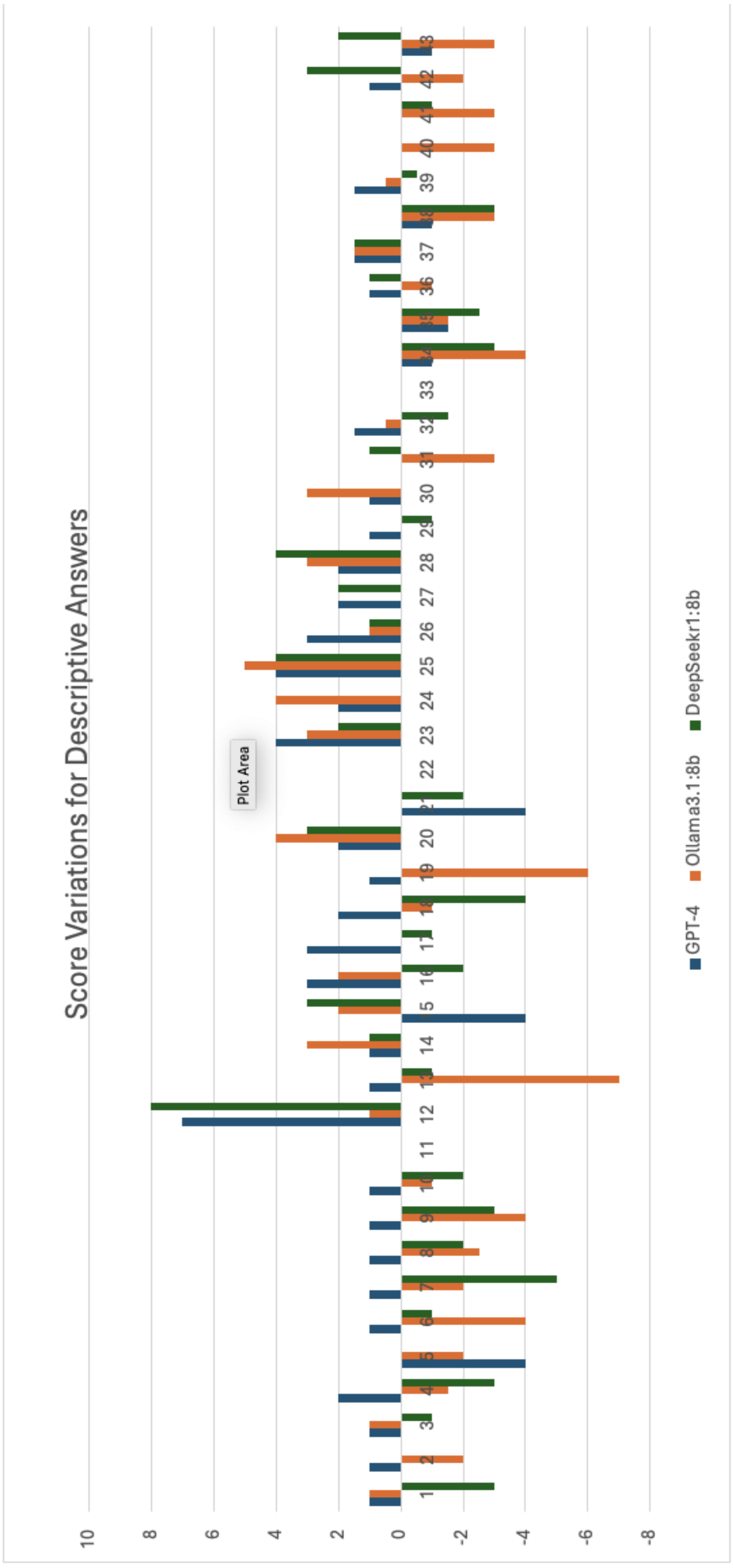


Figure 6: Score variations for Descriptive Questions

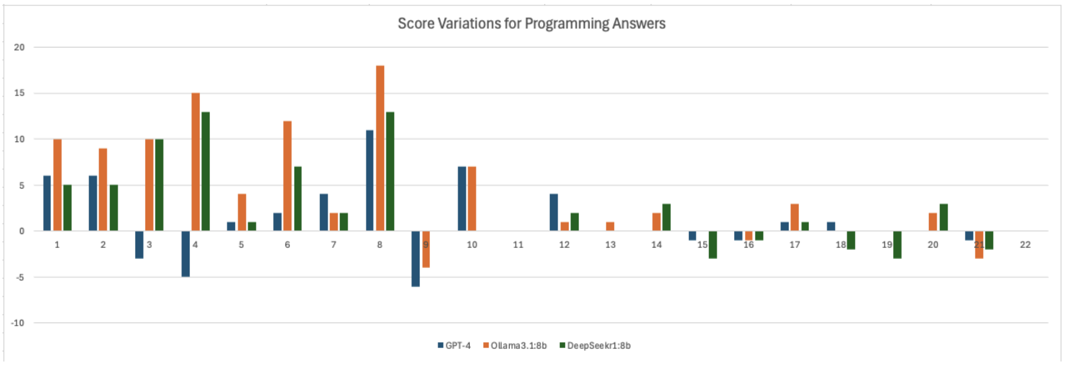


Figure 7: Score variations for Programming Questions

6.1.2 Feedback Quality

A qualitative survey was conducted involving feedback from two assessors, Prof. Jorg Winckler and Ruben Nuredini PhD, who evaluated 60 generated feedback samples. Each sample was rated using a Likert scale ranging from 1 (strongly disagree) to 5 (strongly agree) based on predefined criteria.

Considering the combined feedback from both experts, the general perception of feedback quality was moderately high. The majority of ratings ranged from 3 to 5, indicating that the feedback was generally well-received. However, occasional low scores (1-2) suggest that certain feedback samples did not meet expectations.

The feedback assessments reflected a degree of consistency, with higher ratings being common. However, variability in scores highlighted inconsistencies in feedback quality, particularly in the generated feedback did not align with expert expectations. Most evaluations demonstrated moderate consistency, with ratings often falling within the mid to high range.

Overall, while the generated feedback was generally considered satisfactory, the presence of variability suggests a need for refinement in the feedback generation process to ensure consistent quality across assessments.

6.1.3 Controlled Testing and Consistency Checks

To assess the system's reliability, we conducted controlled testing using a sample of 60 student answers, which included both descriptive and programming question types.

Consistency of Results:

The testing was repeated multiple times to examine the system's stability and to minimize potential hallucinations (i.e., incorrect or unfounded information generated by the system). Variations across the runs were recorded to assess consistency, and the results were found to be generally stable, with some fluctuations observed.

The following table summarizes the variance for each run:

Variance Run1	Variance Run2	Variance Run3	Variance Run4	Variance Run5	Variance Run6
0.5	0.5	0.5	-1.5	-0.5	0.5
-1.33	0.67	-1.33	0.67	0.67	0.67
-0.17	-2.17	-0.17	0.83	1.83	-0.17
-0.5	0.5	0.5	0.5	0.5	-1.5
-0.5	1.5	-0.5	1.5	-1.5	-0.5
-0.33	0.67	-0.33	0.67	-0.33	-0.33
0.0	0.0	0.0	0.0	0.0	0.0
-0.83	1.17	-0.83	0.17	0.17	0.17
-0.83	0.17	0.17	0.17	0.17	0.17
-0.5	-0.5	-0.5	0.5	0.5	0.5

Table 2: Variance for Multiple Runs for the same sample

While there were variations in the results across the different runs, the fluctuations were relatively small, indicating a good level of consistency. The observed variations are expected in a system that processes diverse inputs and operates in a dynamic environment. Despite these small differences, the system produced consistent outputs across all test runs, demonstrating the reliability of the model in generating accurate grades and feedback. Please refer to Table 6 in the appendix for complete grade values.

Analysis of Variations:

The variations across the test runs ranged from minor positive to negative values. Notably, the system maintained consistency in most of the answers, with the variance mostly being within the range of -2 to 2. The consistency checks, performed by running the same prompts multiple times, ensured that the outputs remained stable, and any major inconsistencies or significant deviations were minimal.

The variations that were noted were primarily attributed to the complexity of the questions and slight differences in the model's responses. These differences were further analyzed and did not significantly affect the overall grading accuracy. For example, in runs with lower variance, the system's responses remained almost identical, suggesting high stability in those cases.

The results of the controlled testing indicate that the system is reliable and stable in its performance. Although minor fluctuations in variance were observed, they were

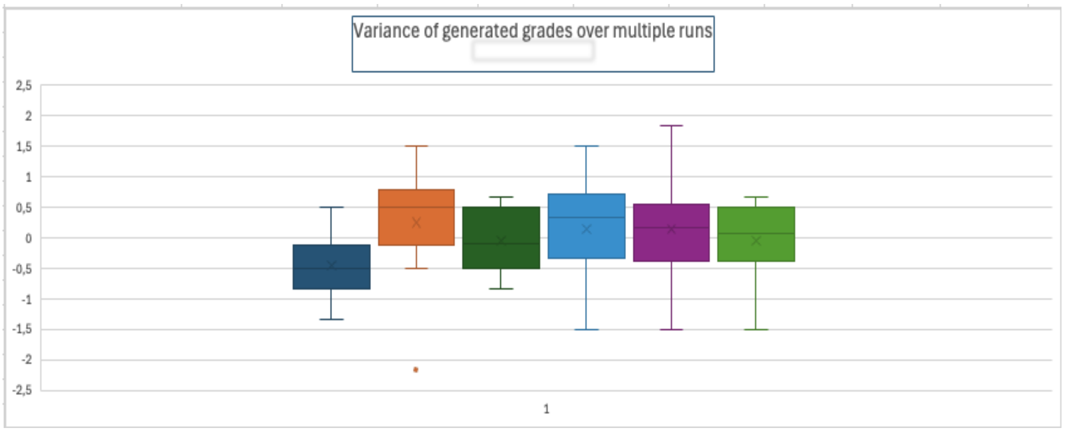


Figure 8: Variations of system generated grades

within acceptable limits and did not detract from the overall accuracy and consistency of the system's outputs. This supports the conclusion that the system is capable of providing consistent, accurate, and reliable grading and feedback for both descriptive and programming question types, even in a diverse testing environment.

6.1.4 Metrics

All test samples were run on the three models—GPT-4.o, Ollama3.1:8b, and DeepSeekr1:8b—to evaluate their performance against the True Grade. The results, including the calculated Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Standard Deviation (SD) for the variances, are summarized in Table 3. These metrics provide a comprehensive understanding of each model's accuracy, ability to handle outliers, and consistency in grading. From the variances shown in Figures 5 and 6, we can derive the metrics as follows:

Descriptive Questions			
Metric	**GPT-4**	**Ollama3.1:8b**	**DeepSeeker1:8b**
MAE	1.653	2.236	2.111
RMSE	2.261	3.034	3.122
SD	2.192	2.782	2.835
Programming Questions			
Metric	**GPT-4**	**Ollama3.1:8b**	**DeepSeeker1:8b**
MAE	2.773	5.045	3.455
RMSE	3.558	6.031	4.139
SD	3.047	5.218	4.057

Table 3: Comparison of Metrics Across Models for Different Types of Questions

The comparison table highlights the performance of three models—GPT-4.o, Ollama3.1:8b, and DeepSeekr1:8b—on both descriptive and programming questions. The evaluation is based on three metrics: MAE (Mean Absolute Error), RMSE (Root Mean Square Error), and SD (Standard Deviation).

For descriptive questions, GPT-4.o clearly leads in performance with the lowest scores across all metrics. It demonstrates greater accuracy and consistency, indicated by its MAE of 1.653, RMSE of 2.261, and SD of 2.192. In contrast, Ollama3.1:8b performs the weakest in this category, showing the highest levels of error and inconsistency. DeepSeekr1:8b ranks in the middle but still falls short of matching GPT-4's reliability.

The trend continues in programming questions, where GPT-4 once again outperforms its counterparts. With a significantly lower MAE (2.773), RMSE (3.558), and SD (3.047), it demonstrates better handling of programming-related tasks. Ollama3.1:8b struggles most in this domain, with high levels of error (MAE of 5.045) and variability, making it the least suitable for such questions. DeepSeekr1:8b fares moderately, with scores better than Ollama3.1:8b but notably worse than GPT-4.

For descriptive answers, the performance of GPT-4.0 is reasonably strong, showcasing adequate accuracy and reliability. However, when it comes to programming tasks, the system exhibits limitations that suggest the need for improved metrics and enhanced precision to make it more practical and reliable for real-world application scenarios.

7 Conclusion

This thesis presents the development and evaluation of an agentic system designed to automate the grading and feedback process for student assessments. The system leverages advanced large language models (LLMs), including OpenAI's GPT-3.5, Llama 2, Llama3.1, DeepSeekR1 and later iterations like GPT-4.o, to efficiently analyze and grade student responses, while also providing personalized feedback. Through an iterative development process, the system evolved from its initial prototype to a more robust, scalable, and efficient solution capable of handling large volumes of data.

In Version 1.0, the system's ability to handle large context data was limited by suboptimal token utilization and narrow context windows, which hindered the processing of complex student submissions. However, by Version 2.0, improvements such as the integration of LangChain allowed for better management of context windows and optimization of token utilization, leading to faster processing times and more accurate feedback. The introduction of CrewAI in Version 3.0 further enhanced system efficiency by distributing tasks across specialized agents, allowing the system to scale more effectively while maintaining performance even under higher workloads.

The addition of a user-friendly interface (UI) and a database in Version 3.0 improved accessibility and operational efficiency. The UI enabled instructors and administrators to easily interact with the system, while the database ensured efficient storage and retrieval of student data and feedback, supporting scalability and faster query performance. Together, these advancements made the system more user-centric and capable of handling an increasing volume of submissions.

The system developed in this thesis has significant potential for transforming educational automation by providing efficient, scalable, and personalized grading and feedback solutions. By leveraging advanced AI models and agent-based systems, the project can significantly reduce the administrative burden on educators, allowing them to focus more on student engagement and teaching quality. The system's ability to evaluate a wide range of student submissions—ranging from simple responses to complex, open-ended questions—ensures that assessments are more consistent and objective. Additionally, the personalized feedback generated by the 'Feedback Expert Agent' can guide students in improving their understanding of the subject matter, providing actionable insights that are tailored to their individual performance. This approach fosters a more interactive learning environment, where students receive timely and constructive feedback that encourages continuous improvement. The integration of such systems can be particularly beneficial in large-scale educational settings, such as online courses and large university classes, where manual grading becomes a time-consuming and error-prone task.

The results indicate a notable difference in performance across the models tested. While GPT-4.o demonstrates reasonable accuracy for descriptive tasks, its performance in programming tasks reveals room for improvement. Enhanced precision and refinement are required to make the system more suitable for practical applications in programming. Conversely, Ollama3.1:8b and DeepSeekr1:8b lag behind significantly in both areas, making them less ideal for reliable use. These findings suggest that while GPT-4.o shows promise, further advancements are essential for its wider applicability, particularly in programming scenarios.

Looking ahead, there are several directions for scaling and refining the system. One key area of development is improving the system's ability to handle a wider range of subjects and diverse assessment types, such as coding assignments or interactive problem-solving tasks. Refining the agent-based framework and incorporating more specialized agents for these different task types will enhance the flexibility and accuracy of the system. Additionally, integrating a REST API would allow for easier integration with Learning Management Systems (LMS) and other educational platforms, enabling seamless interaction between the grading and feedback system and existing educational tools. By continuing to scale the system's capabilities and broadening its integration options, this project can play a key role in the evolution of automated educational systems, ensuring they remain relevant and effective in addressing the needs of both educators and learners.

In conclusion, this thesis demonstrates the potential of integrating generative AI with educational tools to create scalable, efficient, and personalized grading and feedback systems. The agentic framework, combined with advanced LLMs and task orchestration mechanisms like CrewAI, paves the way for future developments in educational AI systems. The lessons learned and the improvements made during the development process serve as a foundation for further research and refinement, ensuring that such systems can meet the growing demands of modern education.

8 Acknowledgments

I am deeply grateful to everyone who contributed to the successful completion of this project.

Firstly, my profound thanks to my supervisors, Ruben Nuredini, Ph.D. and Prof. Dr. Jörg Winckler. Your invaluable guidance, encouragement, and insightful feedback have been instrumental in shaping this work's direction and ensuring its quality.

I also extend my gratitude to Hochschule Heilbronn for providing the resources and support necessary to conduct this research.

A heartfelt thank you to the LangChain and CrewAI communities, as well as the Discord group, for their collaborative support, technical discussions, and inspiration throughout the development of this project. Your contributions have been immensely helpful in overcoming technical challenges and enhancing the project's overall design.

Lastly, I am deeply thankful to my family and friends for their unwavering support and encouragement. Your belief in me has been a constant source of strength and motivation throughout this journey.

Thank you to everyone who played a role in this journey.

Bibliography

[1] OpenAI Documentation, "Openai platform documentation: Overview." https://platform.openai.com/docs/overview, mar 2025. Retrieved on March 9, 2025, from https://platform.openai.com/docs/overview.

[2] Ollama Documentation, "Ollama: Readme quickstart guide." https://github.com/ollama/ollama/blob/main/README.md#quickstart, mar 2025. Retrieved on March 9, 2025, from https://github.com/ollama/ollama/blob/main/README.md#quickstart.

[3] LangChain Documentation, "Langchain documentation." https://python.langchain.com/docs, mar 2025. Retrieved on March 9, 2025, from https://python.langchain.com/docs.

[4] CrewAI Documentation, "Crewai: Ai teams for complex tasks." https://docs.crewai.com/, mar 2025. Retrieved on March 9, 2025, from https://docs.crewai.com/.

[5] Neo4j Documentation, "Getting started with graph databases." https://neo4j.com/docs/getting-started/graph-database/, mar 2025. Retrieved on March 9, 2025, from https://neo4j.com/docs/getting-started/graph-database/.

[6] Streamlit Documentation, "Streamlit documentation." https://docs.streamlit.io/, mar 2025. Retrieved on March 9, 2025, from https://docs.streamlit.io/.

.1 Appendices

Question	**Student Answer**	**True Grade**	**GPT-4.0**	**Llama3.1**	**DeepSeek**
QN1	SA1	5	6	6	2
	SA2	2	3	0	2
	SA3	3	4	4	2
	SA4	4	6	2.5	1
	SA5	4	0	2	4
	SA6	4	5	0	3
	SA7	5	6	3	0
	SA8	3	4	0.5	1
	SA9	4	5	0	1
	SA10	5	6	4	3
QN2	SA1	0	7	1	8
	SA2	7	8	0	6
	SA3	5	6	8	6
	SA4	4	0	6	7
	SA5	4	7	6	2
	SA6	5	8	5	4
	SA7	4	6	3	0
	SA8	6	7	0	6
	SA9	2	4	6	5
	SA10	8	4	8	6
QN3	SA1	0	4	3	2
	SA2	2	4	6	2
	SA3	0	4	5	4
	SA4	0	3	1	1
	SA5	2	4	2	4
	SA6	0	2	3	4
	SA7	3	4	3	2
	SA8	0	1	3	0
	SA9	3	3	0	4
	SA10	1.5	3	2	0

Table 4: Grading Results for Student Answers - 1

Question	Student Answer	True Grade	GPT-4.0	Llama3.1	DeepSeek
QN4	SA1	4	3	0	1
	SA2	3.5	2	2	1
	SA3	1	2	0	2
	SA4	0.5	2	2	2
	SA5	4	3	1	1
	SA6	0.5	2	1	0
	SA7	4	4	1	4
	SA8	3	3	0	2
	SA9	3	4	1	6
	SA10	3	2	0	5
QN5	SA1	10	16	20	15
	SA2	13	19	22	18
	SA3	10	7	20	20
	SA4	7	2	22	20
	SA5	11	12	15	12
	SA6	5	7	17	12
	SA7	18	22	20	20
	SA8	4	15	22	17
	SA9	18	12	14	18
	SA10	15	22	22	15
QN6	SA1	1	5	2	3
	SA2	1	1	2	1
	SA3	0	0	2	3
	SA4	5	4	5	2
	SA5	3	2	2	2
	SA6	2	3	5	3
	SA7	2	3	2	0
	SA8	4	4	4	1
	SA9	0	0	2	3
	SA10	5	4	2	3

Table 5: Grading Results for Student Answers - 2

Question	**Student Answer**	**True Grade**	**RUN 1**	**RUN 2**	**RUN 3**	**RUN 4**	**RUN 5**	**RUN 6**
QN2	SA2	7	8	8	8	6	7	8
	SA3	5	6	6	4	4	6	8
	SA7	4	6	8	4	5	6	8
	SA8	6	7	6	7	4	8	6
	SA9	2	6	6	8	6	7	7
Q6	SA2	1	4	5	4	4	5	4
	SA3	0	3	4	4	5	2	5
	SA5	0	3	5	3	5	4	5
	SA6	1	5	6	4	3	3	3
	SA9	0	2	2	2	3	3	3

Table 6: Grading across multiple runs