

Ministry of Education and Science of the Republic of Kazakhstan

SDU University



Zhibek Myrzaliyeva

Collaborative Multi-Agent Chatbot Framework for Intelligent Tutoring and Feedback

THESIS

Presented in Partial Fulfilment for the

Degree of Master of Science in Computer Science

(degree code: 7M06102)

Department of Computer Science

School of Information Technologies and Applied Mathematics

Supervisor: **Selcuk Cankurt**

Kaskelen, 2026

School of Information Technologies and Applied Mathematics

Department of Computer Science

Dissertation Signatures Title Page

The Master's Dissertation of Zhibek Myrzaliyeva

Title:

Collaborative Multi-Agent Chatbot Framework for Intelligent Tutoring and Feedback

Бірлескен көпагентті чат-бот жүйесінің интеллектуалды оқыту мен кері байланысқа арналған құрылымдық негізін жобалау

Разработка совместной многоагентной чат-бот платформы для интеллектуального обучения и обратной связи

Is submitted in partial fulfillment of the requirements for the Master's (MSc) degree in the Department of Computer Science, and approved by the Dissertation Committee:

Head of Committee:

Assistant Prof., PhD, Meraryslan Meraliyev

Academic supervisor:

Assistant Prof., PhD, Selcuk Cankurt

Member's Signature:

Assistant Prof., PhD, Orynbekova K.E.

Member's Signature:

Assistant Prof., PhD, Ayazbayev D.A.



Kaskelen, 2026

Declaration

I confirm that this is my own work and the use of all material from other sources has been properly and fully acknowledged.

Zhibek Myrzaliyeva

May 2026

Acknowledgements

It would be my honor to show my deepest thanks to my supervisor Dr. Selcuk Cankurt for his guidance, continuous help, feedback, and encouragement during this research process. His professional contribution, advice, and mentorship greatly helped me in conducting this dissertation.

In addition, I thank the School of Information Technologies and Applied Mathematics at Suleyman Demirel University for offering an excellent academic environment and all needed resources for carrying out this research.

I also want to thank the faculty members and academic staff for the courses, guidance, and support that helped me gain new knowledge related to artificial intelligence, intelligent tutoring systems, and multi-agent technology.

I would also like to show my gratitude to Anthropic for giving me permission to use Claude language model API in order to realize my suggested Collaborative Multi-Agent Chatbot Framework.

My family means much to me and I feel very grateful to my parents for their endless support during my time of studying for the Master's degree, their love, patience, and encouragement.

I am very grateful to all those who made direct or indirect contributions to this research and gave me encouragement throughout my academic career.

Dedication

This thesis is dedicated to:

To my family, for their unconditional love, support, encouragement, and belief in my abilities throughout my educational journey.

This work is dedicated to them with gratitude and appreciation.

Abstract

Intelligent Tutoring Systems (ITS) seek to deliver a personalized approach akin to one-on-one human tutoring. The advent of large language models (LLMs) makes it possible to create conversational tutoring systems at a reasonable scale. Nevertheless, despite some recent advances, single-agent LLM-based tutoring still suffers from problems such as inaccuracies, varying quality of the pedagogical content, and low self-correction ability.

In this dissertation, a Collaborative Multi-Agent Chatbot Framework (CMACF) for ITS and assessments is presented and evaluated. The system includes three agents: Tutor Agent, which generates adaptive explanations; Feedback Agent, which detects and corrects any potential misunderstandings; and Verifier Agent, which validates and optimizes responses prior to their delivery to the learner during Java programming lessons.

For testing, the framework was used to perform 40 different Java tasks in the following domains: loops, conditionals, arrays, object-oriented programming, exceptions, and typecasting. A comparison between three frameworks was conducted based on three scenarios: single-agent system (SA), multi-agent system without verification (MA-NV), and the proposed multi-agent system with verification (MA-V). The criteria applied included correctness, clarity, pedagogy, actionability, and consistency of responses.

It can be concluded that the proposed model resulted in the best scores (4.52 points) and surpassed both the single-agent solution (4.165) and the multi-agent solution without verification (4.215). In addition, 25% of generated responses were subject to changes due to the verification component and improved task completion rate from 17.5% to 52.5%. Although the response generation time of MA-V (11.1 seconds) exceeded the single-agent solution (3.66 seconds), its quality is sufficient for intelligent tutoring purposes.

Keywords: Intelligent Tutoring Systems; Multi-Agent Systems; Large Language Models; Formative Feedback; Pedagogical Agents; Java Education.

Аңдатпа

Зияткерлік оқыту жүйелері (Intelligent Tutoring Systems, ITS) жеке оқытушының көмегімен жүзеге асырылатын дербестендірілген оқыту тәсілін қамтамасыз етуге бағытталған. Үлкен тілдік модельдердің (Large Language Models, LLM) дамуы ауқымды диалогтық оқыту жүйелерін құруға мүмкіндік берді. Дегенмен, бір агентке негізделген LLM жүйелері әлі де фактілік қателер, педагогикалық сапаның тұрақсыздығы және өзін-өзі түзету мүмкіндігінің шектеулілігі сияқты мәселелерге тап болады.

Бұл диссертацияда интеллектуалды оқыту мен бағалауға арналған Collaborative Multi-Agent Chatbot Framework (CMACF) атты көпагентті чат-бот жүйесі әзірленіп, зерттелді. Ұсынылған жүйе үш мамандандырылған агенттен тұрады: бейімделген түсіндірмелерді ұсынатын Tutor Agent, студенттердің қате түсініктерін анықтап түзетін Feedback Agent және жауаптарды пайдаланушыға ұсынбас бұрын тексеріп, жетілдіретін Verifier Agent. Жүйе Java бағдарламалау пәні аясында қолданылды.

Жүйенің тиімділігін бағалау мақсатында циклдер, шартты операторлар, массивтер, объектіге бағытталған бағдарламалау, ерекшеліктер (exceptions) және типтерді түрлендіру тақырыптарын қамтитын Java бағдарламалау бойынша 40 тапсырмаға эксперимент жүргізілді. Үш конфигурация салыстырылды: бір агентті жүйе (SA), тексерусіз көпагентті жүйе (MA-NV) және тексеруі бар көпагентті жүйе (MA-V). Бағалау дұрыстық, түсініктілік, педагогикалық құндылық, қолданбалық тиімділік және жауаптардың бірізділігі критерийлері бойынша жүргізілді.

Зерттеу нәтижелері MA-V жүйесінің ең жоғары орташа көрсеткішке (4.52 балл) қол жеткізгенін көрсетті. Бұл нәтиже SA (4.165) және MA-NV (4.215) жүйелерінен жоғары болды. Тексеру модулі жауаптардың 25%-ын қайта қарауға себеп болып, тапсырмаларды сәтті орындау деңгейін 17.5%-дан 52.5%-ға дейін арттырды. MA-V жүйесінің жауап беру уақыты 11.1 секундты құрағанымен (SA жүйесінде 3.66 секунд), алынған нәтижелер ұсынылған тәсілдің интеллектуалды оқыту жүйелерінде тиімді қолдануға жарамды екенін көрсетті.

Түйін сөздер: зияткерлік оқыту жүйелері; көпагентті жүйелер; үлкен тілдік модельдер; қалыптастырушы кері байланыс; педагогикалық агенттер; Java бағдарламалауын оқыту.

Аннотация

Интеллектуальные обучающие системы (Intelligent Tutoring Systems, ITS) направлены на предоставление персонализированного обучения, аналогичного индивидуальному взаимодействию с преподавателем. Развитие больших языковых моделей (Large Language Models, LLM) открыло возможности для создания масштабируемых диалоговых обучающих систем. Однако одноагентные системы на основе LLM по-прежнему сталкиваются с проблемами фактических ошибок, нестабильного педагогического качества и ограниченной способности к самокоррекции.

В данной диссертации разработана и исследована совместная многоагентная чат-бот система для интеллектуального обучения и оценивания — Collaborative Multi-Agent Chatbot Framework (CMACF). Предлагаемая архитектура включает три специализированных агента: Tutor Agent, формирующий адаптивные объяснения; Feedback Agent, выявляющий и корректирующий ошибки обучающихся; и Verifier Agent, выполняющий проверку и улучшение ответов перед их предоставлением пользователю в контексте обучения программированию на языке Java.

Для оценки эффективности системы был проведён эксперимент на 40 заданиях по программированию на Java, охватывающих циклы, условные операторы, массивы, объектно-ориентированное программирование, исключения и приведение типов. Были рассмотрены три конфигурации: одноагентная система (SA), многоагентная система без проверки (MA-NV) и многоагентная система с проверкой (MA-V). Оценка проводилась по пяти критериям: корректность, ясность, педагогическая ценность, практическая применимость и согласованность ответов.

Полученные результаты показали, что система MA-V достигла наивысшего среднего результата (4.52 балла), превзойдя как одноагентную систему SA (4.165), так и многоагентную систему без проверки MA-NV (4.215). Модуль проверки инициировал пересмотр 25% ответов и повысил уровень успешного выполнения заданий с 17.5% до 52.5%. Несмотря на увеличение времени генерации ответа до 11.1 секунды по сравнению с 3.66 секунды для SA, качество получаемых ответов подтверждает пригодность предложенной системы для интеллектуального обучения.

Ключевые слова: интеллектуальные обучающие системы; многоагентные системы; большие языковые модели; формирующая обратная связь; педагогические агенты; обучение Java.

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CAI	Constitutional AI
CMACF	Collaborative Multi-Agent Chatbot Framework
CS	Computer Science
ITS	Intelligent Tutoring System
JSON	JavaScript Object Notation
LLM	Large Language Model
MA-NV	Multi-Agent without Verifier
MA-V	Multi-Agent with Verifier
NLP	Natural Language Processing
OOP	Object-Oriented Programming
RLHF	Reinforcement Learning from Human Feedback
RQ	Research Question
SA	Single-Agent System
SD	Standard Deviation

Table of Contents

Declaration	i
Acknowledgements	ii
Dedication	iii
Abstract	iv
Андатпа	v
Аннотация	vi
List of Abbreviations	vii

1 Introduction	1
1.1 Background and Motivation.....	1
1.2 Significance of the Work.....	3
1.2.1 Educational Significance.....	3
1.2.2 Technical Significance.....	4
1.2.3 Societal Significance.....	5
1.3 Literature Review.....	5
1.3.1 Intelligent Tutoring Systems: Origins, Architectures, and Limitations.....	6
1.3.2 Formative Feedback: Theory and Educational Evidence.....	8
1.3.3 Large Language Models: Capabilities, Limitations, and Educational Applications.....	9
1.3.4 Multi-Agent Systems and LLM Collaboration.....	11
1.4 State of the Art.....	13
1.4.1 Deployed Commercial System.....	13
1.4.2 Research-Stage Systems.....	14
1.4.3 Gaps in the State of the Art.....	15
1.5 Research Gap.....	15
1.6 Problem Statement, Research Questions, and Objectives.....	16
1.6.1 Problem Statement.....	16
1.6.2 Research Questions.....	17
1.6.3 Research Objectives.....	17
1.7 Contributions and Novelty.....	18
1.8 Outline of Subsequent Chapters.....	19
2 Preliminaries and Methodology	19
2.1 Technical Foundations.....	19
2.1.1 The Transformer Architecture and Large Language Models.....	20
2.1.2 The Claude Language Model.....	21
2.1.3 Prompt Engineering: Strategy and Rationale.....	22
2.1.4 Orchestration and Agent Coordination.....	24
2.2 Dataset Construction and Task Design.....	25
2.2.1 Rationale for Java Programming as Evaluation Domain.....	25

2.2.2 Task Selection Methodology.....	25
2.2.3 Task Structure and Fields.....	26
2.2.4 Difficulty Classification.....	26
2.2.5 Representative Task Examples.....	27
2.3 System Architecture.....	28
2.3.1 Framework Overview and Design Philosophy.....	28
2.3.2 Single-Agent (SA) Configuration.....	29
2.3.3 Tutor Agent Specification.....	29
2.3.4 Feedback Agent Specification.....	30
2.3.5 Verifier Agent Specification and Decision Flow.....	31
2.3.6 Response Synthesis and Delivery.....	33
2.3.7 Implementation Details and Engineering Tradeoffs.....	33
2.4 Experimental Setup and Evaluation Protocol.....	34
2.4.1 Experimental Design.....	34
2.4.2 The Claude-as-Judge Evaluation Protocol.....	35
2.4.3 Evaluation Rubric Design.....	36
2.4.4 Statistical Analysis.....	36
2.4.5 Measurement Variables.....	37
3 Analysis and Findings	37
3.1 Hypotheses and Analytical Approach.....	37
3.2 Overall Comparative Results.....	38
3.2.1 Summary Statistics.....	38
3.2.2 Test of H1: MA-V vs. SA.....	39
3.2.3 Test of H1: MA-NV vs. SA.....	40
3.2.4 Test of H2: MA-V vs. MA-NV.....	40
3.2.5 Success Rate and McNemar Test.....	40
3.3 Dimension-Level Analysis.....	41
3.3.1 Correctness.....	41
3.3.2 Clarity.....	42
3.3.3 Pedagogy.....	43
3.3.4 Actionability.....	44
3.3.5 Consistency.....	45
3.4 Verifier Agent Behaviour and Revision Analysis.....	46
3.4.1 Revision Rate and Calibration.....	46
3.4.2 Revision Trigger Categorisation.....	47
3.4.3 Impact of Revision on Output Quality.....	48
3.5 Latency Analysis.....	49
3.5.1 Latency by Configuration.....	49
3.5.2 Deployment Implications of the Latency Profile.....	50
3.5.3 Latency-Quality Pareto Analysis.....	50
3.6 Qualitative Case Studies.....	51
3.6.1 Case Study A — Task 37 (Array Reference Aliasing).....	51
3.6.2 Case Study B — Task 30 (Constructor Return Statement).....	52
3.6.3 Case Study C — Task 26 (Post-Decrement Assignment Cancellation).....	53
3.6.4 Case Study D — Task 8 (String Comparison).....	54
3.6.5 Score Distribution: Aggregate Pattern.....	54

4	Discussion	55
4.1	Interpretation of Results	55
4.1.1	The Primacy of Verification Over Specialisation	55
4.1.2	Why Consistency Shows the Largest Improvement	56
4.1.3	The Floor-Raising Mechanism and Its Deployment Significance	57
4.1.4	The Difficulty-Quality Interaction	58
4.2	Positioning Within Existing Literature	58
4.2.1	Relationship to Multi-Agent LLM Research	58
4.2.2	Relationship to Intelligent Tutoring Systems Research	59
4.2.3	Relationship to Formative Feedback Research	60
4.3	Limitations	61
4.3.1	Evaluation Validity: The LLM-as-Judge Circularity	61
4.3.2	Benchmark Scale and Coverage	62
4.3.3	Single-Domain Generalisability	62
4.3.4	Single-Turn Interaction Scope	63
4.3.5	Absence of Learning Outcome Measurement	63
4.4	Future Research Directions	64
4.4.1	Human Expert Validation Study	64
4.4.2	Randomised Controlled Trial for Learning Outcomes	64
4.4.3	Student Knowledge Modelling Integration	65
4.4.4	Multi-Turn Dialogue Extension	65
4.4.5	Multi-Verifier Panel Architecture	66
4.4.6	Domain Generalisation	66
5	Conclusion	68
5.1	Summary of the Work	68
5.2	Principal Contributions	68
5.3	Final Reflections	69
	Bibliography	71

Chapter 1

Introduction

1.1 Background and Motivation

There are not many domains in which an observation of the 1980s has continued to define the ambition of an entire research area for almost four decades now. One such domain, however, is education, in particular the study of human tutoring effects. Bloom's famous 1984 finding showed that learners who received personalized instruction from an experienced human tutor outperformed their peers in conventional classroom settings by two whole standard deviations—that is, they averaged at least 98 percentile over the latter. This result is best seen not as a discovery to celebrate but rather as a challenge: is there any way we can provide students with such effective instruction as personalized tutoring in numbers large enough to matter?

In this light, forty years of development of intelligent tutoring systems (ITS) have achieved considerable progress but failed to solve the problem completely. From the pioneering work on LISP Tutor to modern commercial systems of the Cognitive Tutor family, it has been shown multiple times that computers can engage in adaptive instructional interactions and outperform classroom instruction in well-defined procedural domains by 15-50%. A meta-analysis by van Lehn from 2011, which considered 26 independent studies, verified this advantage and found that ITS produce, on average, $d = 0.76$ effect sizes relative to classroom learning—an achievement substantial in itself, yet clearly inferior to the two-sigma effect. It should be noted that the gap with human tutors remains in exactly the realm of instruction in which it is most difficult to fill: the generation of natural language explanations and feedback typical of productive dialogue.

The problem is architectural. Traditional knowledge representations employed in ITS are constructed on a per-domain basis by domain experts and therefore achieve great specificity in their target domain but lack the flexibility to represent other domains, since their very structure has to be redesigned from scratch. As a result, despite the need for adaptive instruction becoming universal to all domains, ITS remained bound to a few selected areas for the entirety of their history, since it is prohibitively costly to author knowledge representation for each new domain of application. Woolf observes that among three essential components of ITS—domain model, student model, and pedagogical model—the latter is by far the most underdeveloped, yet also most expensive in manual authoring terms.

This situation changed fundamentally with the rise of large language models in the early 2020s. Models like GPT-4 and Claude encode extensive general-purpose knowledge

covering almost all possible academic domains by pretraining on large-scale text corpora. They can understand, follow, and execute complicated verbal instructions. And most importantly, they generate high-quality coherent text relevant to the given context. Crucially, these abilities are available regardless of domain-specific knowledge authoring: an LLM can generate tutoring content explaining Java programming misconceptions, French grammar mistakes, and organic chemistry mechanisms using the same underlying model, with different prompts. This represents a qualitative shift from authoring-centric ITS architecture, creating possibilities to deliver adaptive instruction at scales previously inconceivable.

These opportunities come paired, however, with a set of fundamental limitations with peculiar implications for the educational domain. The first of them is factuality: LLM responses frequently contain factual errors. Moreover, LLM responses are non-consistent, and, given the same query on different occasions, produce answers containing different claims. Finally, LLMs are unable to estimate their performance reliably since the reasoning processes producing the errors are exactly those employed in evaluations. All these features combine in the educational domain to create an especially perilous environment, where providing a confident but erroneous answer to a student can not only fail to convey the necessary concept but also implant a misconception impossible to debunk.

One effective engineering solution to these limitations involves multi-agent architectures. The central idea here is that while a single LLM invocation cannot guarantee a certain property of the resulting response (accuracy, factuality, logical coherence), a dedicated independent verification process can enforce these criteria. This approach reflects long-standing best practices adopted in various expert communities: peer review in academic publishing, code review in software engineering, and the requirement of second opinions in clinical medicine. All three acknowledge the necessity of independent oversight even when performed by highly skilled professionals. In an LLM-based system, this is implemented by splitting the task into three distinct invocations—generation, feedback, and verification—which together constitute a pipeline ensuring quality control absent from a single invocation.

This dissertation introduces, builds, and evaluates exactly such a design framework applied specifically to teaching Java programming in college-level classes. Java is a highly popular programming language used in the vast majority of CS curricula around the globe, and systematic misconceptions developed by beginners studying it are known in sufficient depth. In this setting, the CMACF proposed consists of three agents—the Tutor Agent, the Feedback Agent, and the Verifier Agent—and their cooperation over the pipeline that generates tutoring content, provides formative feedback, and conducts an independent quality assurance pass before a final response goes to a student. This project addresses experimentally the question of whether this overhead translates into actual improvement in tutoring effectiveness.

1.2 Significance of the Work

The significance of this research operates simultaneously at the educational, technical, and societal levels, and the three levels reinforce each other in ways that make the work more than the sum of its parts.

1.2.1 Educational Significance

Programming is more than simply a subject prone to conceptual errors from time to time. As a hierarchical domain, a misconceptualization of Java's treatment of object references can propagate errors throughout the next set of topics on memory allocation and management, collections, and pass-by-value. Accordingly, the longer it takes for the misconceptualizations to be corrected, the harder they become to address. For example, students with misconceptions about the equality operator (`==`) used for comparing Java strings may spend years writing incorrect comparisons until their misconceptions cause problems. Likewise, students believing that assignment of one array variable to another creates a copy of an array can generate buggy software where their aliasing mistake stays silent until some hard-to-track problem appears at runtime. It should go without saying that the sooner and more accurately the misconceptions are identified and addressed, the easier the cognitive restructuring becomes for the students.

It places emphasis on the importance of quality feedback. If a response of a tutoring system points out a student's mistake by simply telling him or her what to change (such as "use `equals()`, not `==`"), but does not explain the concept behind the error (the idea of object identity in Java, the fact that string objects are stored on the heap, etc.), then it is unlikely that the student will learn anything from it besides how to fix one particular bug. Ma et al.'s [32] meta-analysis of ITS effectiveness based on 107 studies reveals that elaborative feedback, which explains a conceptual reason for an error, is the biggest contributor to ITS success. The CMACF framework follows the same philosophy – the Feedback Agent produces responses in a three-part format that specifically aims to explain concepts and the Verifier Agent evaluates responses based on their pedagogic value as concept-based answers.

The scale factor makes the problem even more urgent. Large university introductory classes on programming have several hundred students per semesters but cannot afford teaching assistants in similar numbers. As a result, many students suffer from office hours overload, delayed feedback, or generic responses for frequently observed mistakes. There is no solution to this problem, because the available software solutions for programming, such as code completion assistants, do not solve it at all.

1.2.2 Technical Significance

In terms of AI systems, the contribution of this paper can be stated as follows: a multi-agent pipeline with a built-in quality-control step demonstrates performance superiority not only against a single-agent approach but also against a multi-agent pipeline without such a step. What makes this contribution more concrete compared to the general statement about superiority of multi-agent over single-agent systems is the fact that it is falsifiable.

As we already noted, the difference in MA-NV and SA performances being non-significant ($p=0.078$), while MA-V performs significantly better than the latter ($p=0.001$) is important for us exactly in that it shows what causes that improvement – the loop involving verification by the Verifier agent. Thus, unlike the statement of more equals better, our finding reveals the mechanism involved.

Our contribution to the fast-developing field of multi-agent LLM system architecture is also notable. It lies in the fact that, while previous work was concerned with the comparison between multi- and single-agent approaches as a whole and thus did not look into the factors that cause the observed superiority, we have been able to establish the importance of a certain architectural feature – verification – thanks to the comparative control provided by CMACF with constant number of generators (2) and differing presence of the Verifier agent.

Finally, our findings contribute to prompt engineering for educational purposes. Each prompt developed in order to guide an agent's behavior in terms of its system prompt was the product of some engineering choices and can have consequences. For instance, it turns out that the criteria of factual correctness and educational depth of the Verifier's instruction are behind 60% of revision decisions made by the Verifier agent.

1.2.3 Societal Significance

Private tuition markets are broad and highly unequal. In countries where the private tuition industry is well developed, such as the UK, South Korea, Hong Kong, and many others, access to private one-on-one tutoring services is directly tied to social and economic class, with the benefits afforded by these services compounding through several years of schooling [1]. It is reasonable to claim that an artificial intelligence-driven tutoring system which truly offers individualized, quality feedback is capable of leveling, but not closing, this divide. In terms of cost-efficiency, the API-based design of the CMACF, which does not require any special hardware, model customization, or ^昂 costly infrastructure, is a model of implementation that would be affordable for organizations working in economically challenging conditions. A university in a developing country with stable internet connections and a lack of teaching assistants could affordably deploy the CMACF; the creation of a similar service using other ITS solutions would be much more expensive and time-consuming.

1.3 Literature Review

The research described in this dissertation draws on four bodies of literature that have historically developed in relative isolation: intelligent tutoring systems, formative feedback theory, large language models, and multi-agent AI architectures. The contribution of this work lies partly in synthesising insights from all four in a single engineered system. The review below is therefore organised to highlight not only the internal logic of each field but the specific connections across them that motivated the CMACF's design.

1.3.1 Intelligent Tutoring Systems: Origins, Architectures, and Limitations

Intelligent Tutoring System (ITS) technology was pioneered with Computer-Assisted Instruction (CAI) in the 1960s. Such systems were able to present content and evaluate student responses via computation, but lacked any capacity for adaptive behavior or personalization due to operating on a rigid, scripted sequence. The critical conceptual breakthrough came with Carbonell's SCHOLAR system, which used a geographic domain knowledge base represented as a semantic network to conduct Socratic dialogue: pose questions with answerable by inferring from the semantic network and accept correct answers as semantically coherent responses to the posed question. SCHOLAR showed that in order for an instructional program to be high-quality, it must have an internal representation of the knowledge being taught as opposed to just a library of questions and answers.

Subsequent scientific activity in the ITS field culminated with Anderson et al.'s seminal paper introducing the Adaptive Control of Thought (ACT*) cognitive architecture [4]. In this theory, human cognition was described as a production system in which production rules fire when their condition is satisfied by the contents of working memory. It followed that to successfully tutor a student, one should model his current production rule set and intervene during the misapplication of rules. To implement this idea in practice, the LISP tutor used expertly compiled models of the programmatic actions to determine the correctness of the student's code at the moment of execution and gave real-time feedback, as well as provided context-appropriate messages from a library of messages. In controlled experiments, the LISP tutor had shown to allow students to learn LISP in about twice less time compared to classroom teaching [4].

Corbett & Anderson's knowledge tracing technique [5] provided a compact formalism for estimating student learning based on ACT* in the form of two-state hidden Markov process for each knowledge component, with four parameters: the probability of initial knowledge ($P(L_0)$), the probability of learning on each practice opportunity ($P(T)$), the probability of slipping despite knowledge ($P(S)$), and the probability of guessing correctly without knowledge ($P(G)$). This simple yet elegant formalism provided the mathematical basis for predicting student mastery level based on performance data and, conversely, for

calculating the number of required practice opportunities to master a knowledge component. Knowledge tracing continues to be widely used today, with recent deep knowledge tracing using recurrent neural networks showing improved accuracy without sacrificing conceptual clarity [34].

Meta-analysis conducted by VanLehn et al. [6] represents the most authoritative review of empirical evidence on outcome measures in ITS. Among 26 randomized/quasi-randomized controlled trials across several subject domains (mathematics, physics, and computer science), the average effect size in favor of ITS versus classroom instruction was 0.76 d. Much more instructive was the comparison of ITS performance to human tutoring: the average effect was $d=0.09$, which is not statistically distinguishable from 0. This result suggests that, despite considerable promise, even state-of-the-art ITS systems still perform inferior to experienced human tutors. VanLehn et al. hypothesized that this gap was primarily attributable to differences in the quality of explanation generation: human tutors were able to produce contextually novel explanations based on the student's own reasoning, while ITS relies on pre-written message banks with insufficient coverage of all possible student reasoning.

LLM-based ITS is poised to eliminate the problem identified in this meta-analysis. The core of the system with GPT-4/Claude will be able to produce new contextually relevant explanations based on student reasoning without any pre-written message bank whatsoever. In this dissertation, it is investigated whether such a capability is reliable enough to make a tutor trustworthy in an education context or the problem of hallucinations/inconsistency requires some architectural modification to ensure safety.

According to Woolf [7], there are four core components of a tutoring system: domain model (which contains information about what is to be taught), student model (a model of the learner's state of knowledge), pedagogical model (determines how to teach and what to teach given student's knowledge state), and communication interface (translates tutor–student dialogues). Most of the traditional efforts in building ITS were invested in creating the first two, while the latter two were implemented as hand-authored message banks and dialogue scripts. In this regard, the CMACF system could be mapped onto Woolf's framework in the following way: pretrained LLM serves as a domain model, the combination of the difficulty score and student response is used as a lightweight student model approximation, and the system of prompts designed by engineers is the implementation of the pedagogical model – the key difference being that these prompts represent pedagogical ideas in a human-readable form.

Most comprehensive study of using NLP methods to close the dialogue gap has been done in the context of AutoTutor [8]. With latent semantic analysis of the semantic distance between a student's response and a list of acceptable answers, AutoTutor drives mixed-initiative Socratic dialogue through posing clarification questions ("Tell me more..."), hint questions ("Consider what happens when..."), prompts for specific missing

concepts, and giving direct feedback in case of an incorrect answer. In experimental settings, AutoTutor achieved an improvement of 0.4-0.8 standard deviations relative to no-tutoring control group [8]. In follow-up work, Affective AutoTutor [10] integrated facial expression recognition and linguistic sentiment analysis and demonstrated that adaptively adjusting the tutor behavior depending on the student's emotional state led to significant improvements in engagement and outcomes for frustrated/bored students.

A meta-analysis conducted by Ma et al. [32] is particularly interesting in regards to this work since it identifies the most influential component in ITS systems on learning outcome measures. For a total of 107 studies, they found that the biggest predictor of gain was the elaborateness of error feedback: explaining why the answer is incorrect and providing guidance on what to do next results in much larger gains compared to giving just correctness feedback. This directly informs the choice of the CMACF architecture with special attention to the Verifier's quality criteria (pedagogy and actionability).

1.3.2 Formative Feedback: Theory and Educational Evidence

The literature on formative feedback provides the theory underpinning the CMACF's criteria for designing and evaluating feedback. According to Hattie and Timperley's well-known synthesis, there are four levels of feedback: task level ("Is this the right response?"), process level ("Which strategies produce the right response?"), self-regulation level ("How do I control and regulate my own approach to the task?"), and self level ("Praise or criticism of the person performing the task"). Process-level feedback produces the highest impact in terms of promoting learning, while self-level feedback—praise without specific instruction—is often ineffective or even harmful due to distracting from the task. As applied to automatic tutoring, this implies that feedback like "Great effort, try again!" is objectively worse than an explanation of why the student was incorrect at the relevant step.

Shute's thorough review of formative feedback highlights several qualities of effective feedback: timing, specificity, reference point, and cognitive load. While elaborate feedback for skills—such as programming—improves outcomes compared to mere verification, an excessive elaboration can hinder learning by overwhelming the working memory. In other words, novice programmers need to be presented with feedback that does not exceed the capacity of their cognitive scaffolding and allows building upon it progressively. Based on this research, the CMACF adopted Verifier's criterion of clarity: verbose explanations that hide the core of the feedback within unnecessary contextual details should be avoided.

Another critical consideration is the difference between knowledge gaps and misconceptions. The former are filled with additional information. However, the latter require the students to accept that they have an incorrect belief before they are ready to learn the correct information. Thus, a constructivist framework proposed by Chi describes effective misconception correction as an attempt to introduce a contradiction to the learners' mental model followed by presentation of the correct alternative. In the case of

CMACF, this is achieved by means of the Tutor Agent's Socratic approach: instead of presenting the answer to the problem, the program guides the students through the reasoning necessary to detect flaws in their beliefs.

In their survey of intelligent tutoring strategies designed to detect and address misconceptions, Nkambou et al. found that the most successful programs use a two-stage approach: the first one diagnoses the misconceptions and finds out how they arose, and the second one remedies the problems at conceptual level. This principle is implemented in CMACF's specialization of agents: Feedback Agent detects errors in student's work by explaining what is wrong, and the Tutor Agent offers the right solution by addressing "Why it is wrong". The Verifier's pedagogy criterion checks if the latter is truly addressed conceptually rather than just rephrased in the explanation of the former.

1.3.3 Large Language Models: Capabilities, Limitations, and Educational Applications

The transformer architecture described by Vaswani et al. [24] laid the computational groundwork for current state-of-the-art large language models. The ability to attend to and combine information from any position in the input sequence via the self-attention mechanism overcame the sequentiality bottleneck faced by recurrent models and allowed for efficient scaling-up of training beyond previous limits. Bidirectional pre-training based on masked language modeling by BERT [25] proved to provide representations that transfer well to a range of downstream tasks. Finally, GPT-3 [14] demonstrated the few-shot capabilities of sufficiently scaled autoregressive models—that is, the ability to perform new tasks from contextual examples alone without gradient-based finetuning—a feature facilitating adaptation of LLMs to educational contexts by means of carefully designed prompts rather than specialized training data.

The scaling laws of LLMs led to a series of successively improved models which are especially promising from an educational standpoint. As reported in GPT-4 [13], the system was able to match or exceed professional human levels of performance on a range of standardized tests including the bar exam, LSAT, as well as multiple medical licensing exams. From an educational standpoint, the implication is clear: LLMs are capable of acting as true subject matter experts for virtually any field of study—in contrast to past tutoring technologies. The critical question is not whether or not an LLM knows the correct material (which is typically the case), but whether it can explain it reliably and appropriately.

The use of RLHF as described in Ouyang et al. [26] has been shown to substantially improve model alignment with the user's goals and intentions, improving reliability, truthfulness, and safety compared to solely pre-trained models of similar scale. The alignment improvements provided by RLHF are especially important from an educational perspective, as aligned models are much less likely to generate unfounded assertions, will

calibrate their level of explanation to a particular audience, and are unlikely to produce content that could adversely impact student motivation or welfare. Claude [28], the LLM used in this paper, employs both RLHF and Constitutional AI—an approach whereby the LLM assesses itself against an explicit framework of principles—in order to reduce human intervention in alignment and increase internal self-consistency of explanations.

Prompt engineering is becoming an independent field of study, with ample empirical evidence suggesting that the design of prompt instructions significantly affects the quality of generated responses. Wei et al. [27] found that chain-of-thought prompting—asking the model to reason step-by-step before giving a response—significantly increases performance on multi-step reasoning and arithmetic problems, with scaling factors increasing the effect size. The plausible interpretation is that chaining explicit reasoning leads to a sort of computational working memory, helping the model incorporate all relevant facts more efficiently than in single pass generation. Kojima et al. [16] demonstrated that even the zero-shot chain-of-thought instruction "Let's think step by step" provides significant improvements in accuracy, showing that chain-of-thought reasoning is a latent ability of the model which can be triggered through prompting. The Verifier Agent prompt in this work explicitly implements chain-of-thought reasoning: the model must reason through each criteria before providing a judgment—the design decision inspired by Wei et al. [27].

Another relevant idea described by Wang et al. [15] is that of self-consistency decoding, which suggests sampling different chains of reasoning from the model and choosing the most consistently obtained conclusion—thus minimizing stochastic generation errors. While the method described in this paper does not use self-consistency decoding, it incorporates another way of increasing self-consistency at the architectural level: instead of sampling from a single agent, the system generates complementary content using two agents, and then applies the Verifier to check for consistency. Architectural self-consistency may provide additional robustness compared to token-level consistency since the generations performed by each individual agent do not depend on a shared context, and thus will not propagate errors.

Hallucination, bias, and brittleness of models were recognized as major risks by Bommasani et al. [17]. Such risks make their unmediated use in high-stakes situations ill-advised. In an educational context, hallucinations pose additional risks due to the possibility of student misconception being created. Empirically, several studies have verified these concerns in the context of LLM education: error rate of 10-25% on subtle domain-specific questions has been measured for unverified single-agent systems. Thus, the CMACF's Verifier serves to address the quality-reliability gap described in these papers.

1.3.4 Multi-Agent Systems and LLM Collaboration

The field of multi-agent systems boasts an extensive history in the realm of AI research, with distributed constraint satisfaction, cooperative game theory, multi-agent reinforcement learning, and market-based coordination all falling into this category. By contrast, the application of multi-agent ideas to large language models (LLMs) remains a relatively recent development and has generated a thriving corpus of work on which the CMACF builds directly.

The works by Park et al. have demonstrated the emergence of cohesive social behavior from a series of long-range interactions between LLM-powered agents featuring memory and communicative capabilities, which include information dissemination, establishment of relations, and spontaneous planning. Although their paper primarily served as proof-of-concept, two important characteristics related to the CMACF became evident. First, LLMs retain ability to perform in a role-defined way throughout numerous iterations. Second, communication between multiple LLM agents generates results fundamentally different from those achieved by one agent. The second characteristic forms the core architectural assumption of the CMACF.

The CAMEL framework developed by Li et al. has been the first to employ a role-play structure in order to improve the performance of two-agent LLM collaboration, where the authors found that a specific allocation of roles, such as having one AI assistant and another AI user, greatly increases the quality of tasks completed compared to a single-agent model. The hypothesis advanced by the researchers is that playing against the roles encourages agents to explain themselves and thus makes them more concise and precise. Despite being most similar to the CMACF in terms of the overall design, CAMEL does not feature any quality assurance through verification and lacks this crucial step in its architecture.

AutoGen has become the most comprehensive framework for multiple LLM agents available at the time the CMACF was designed, providing flexible role allocation, ability to work with humans or autonomously, execution of codes, and customizable conversation topology, including sequential, parallel, and cyclic ones. In experiments spanning coding, math, and decision-making tasks, the superiority of multiple agents was shown consistently in relation to single-agent models, with the largest improvement observed in the case of tasks requiring iterative correction of mistakes. The execution pipeline employed in AutoGen, involving code generation and its execution by another agent, with errors reported back for corrections, resembles the verification and revision loop used by CMACF, adjusted to pedagogical needs.

In their study of multi-agent debate as a way of improving task performance, Du et al. investigated how several agents independently answer questions and then iteratively refine their own answers based on the ones produced by others in a fixed number of rounds. As was proven, on factuality and reasoning benchmark tasks, the proposed algorithm

increased accuracy by 5 to 15 percent compared to a single-agent baseline and had the strongest effect in cases of systematic errors made by individual agents. In further research, Liang et al. demonstrated that debating leads to divergent thinking, as exposure to multiple viewpoints increases the likelihood of one agent figuring out the right solution to a problem. Together, the mentioned results suggest that improvements made by multi-agent architectures in question stem from their dynamic nature, rather than the mere increase of computational power.

An application of multi-agent debate as an evaluation tool was performed by Chan et al. in the ChatEval paper. Multiple AI evaluators engaged in a discussion on the quality of answers and reached greater agreement on what constituted a proper response than a single evaluator did. The results are directly related to the CMACF and, in particular, its Verifier Agent module whose reliability is crucial to the whole pipeline. Indirectly, the results prove that using a system prompt aimed at encouraging an agent to look for mistakes is indeed a better idea than directing it to confirm answers.

Despite the extensive work done in the field of multi-agent LLM systems, the literature lacks empirical studies focused on their implementation in tutoring and evaluating their quality with respect to education theory. General-purpose research in multi-agent LLMs relies mainly on factual and task-oriented accuracy scores as measures of quality, without reference to pedagogy. Thus, to use such a system in the context of intelligent tutoring, two steps are needed—adaptation to the new domain and creation of a quality evaluation framework, which is what the current dissertation provides.

1.4 State of the Art

The state of the art in LLM-based educational technology is characterised by rapid product development substantially outpacing systematic academic evaluation. The landscape in 2024–2025 is dominated by a small number of widely deployed commercial systems, a larger number of research prototypes with limited evaluation, and an emerging body of academic literature beginning to apply rigorous methodology to LLM tutoring assessment.

1.4.1 Deployed Commercial Systems

The largest and most widely used LLM-based tutoring system is Khan Academy's Khanmigo, embedded into the Khan Academy ecosystem and serving millions of students for learning purposes in various topics ranging from math and science to history and programming. Khanmigo utilizes GPT-4 and is based on a carefully crafted Socratic system prompt: the model is specifically trained not to answer learners' questions, but to guide them toward the answer using prompts, hints, and questions. Pedagogically, the use of Socratic dialogue might be justified: productive struggles have proven to promote deep learning. Still, there is a clear risk involved – the students who struggle with a concept and

seek factual help may become frustrated by Khanmigo's responses, and the lack of feedback to know when it is better to switch to straightforward explanation may exacerbate the problem. Finally, Khanmigo does not employ any kind of multi-agent verification system whatsoever: the quality of its outputs depends on the accuracy of GPT-4's first response to the system prompt, with no architecture for detecting possible errors until the response reaches the learners.

Duolingo Max uses GPT-4 to deliver personalized post-exercise explanations of mistakes in language exercises (Explain My Answer) and to role-play conversations in the target language (Roleplay). The Explain My Answer function resembles the CMACF's Feedback Agent (produces customized and detailed explanation of the mistake in the learner's exercise), but unlike the former, it lacks the Verifier's step. The limited range of tasks (explanation of why the provided language exercise response is wrong) makes the system much simpler than general-purpose tutoring; at the same time, the abundance of repetitive errors in language learning allows pre-validation of numerous explanations through testing.

GitHub Copilot is an AI tool developed for coders that provides suggestions for code and documentation writing and helps explain what certain code elements do. The audience targeted by this product is professionals rather than learners, while the level of explanations is adapted for advanced readers. Finally, the tool lacks functionality needed to identify and resolve misconceptions behind students' errors, thus it cannot serve as an educational tool despite its similar purpose. There are several differences between the two applications besides the target audiences.

1.4.2 Research-Stage Systems

Indeed, there is already an extensive set of empirical findings accumulated in the current literature about feedback generation from LLMs. Evaluations of GPT-4's automated feedback generation capabilities regarding student code suggest that the feedback provided by the models is as good as, or even better than, the feedback offered by teaching assistants in many cases. Nevertheless, instructors with expert level skills are likely to notice that model-generated feedback leaves much room for improvement when considering such aspects as depth of understanding and diagnosis of misconceptions. An important observation made here concerns a significant difference in quality assessment results provided by the students and the instructors: the gap between the ratings may arise from students' inability to assess the accuracy of feedback since they lack sufficient knowledge of the subject matter to make a judgment.

The study by Abdelaziz et al., focused on LLM-assisted planning and decision-making in agents, highlights the emergence of the LLM-as-orchestrator architectural pattern, which involves assigning plans and subtasks to the specialist agents from the master agent. In that context, the present work can be seen as part of the broader trend of LLMs being used for

orchestrating agents in complex tasks. The survey by Abdelaziz et al. also reveals that the improvements achieved with the CMACF in terms of feedback quality may be generalizable for the whole family of orchestration-based approaches.

Finally, it should be mentioned that deep knowledge tracing based on the transformer architecture is the current state-of-the-art technique used for predicting student mastery on a particular skill or knowledge point with considerably higher accuracy compared to traditional knowledge tracing techniques. At the same time, the problem with deep knowledge tracing is that it requires long-term monitoring of the student's performance on several occasions to be effective, while CMACF operates in the cold-start scenario where no previous information on the student is available.

1.4.3 Gaps in the State of the Art

There are three structural gaps in the field relevant to this dissertation, the first being the lack of a quality-control agent in any currently deployed or researched system. No existing tutoring systems use an extra step of passing through an additional model that can check the output generated by the original GPT-4, Claude, or fine-tuned model before returning it as the final output. The second gap is the lack of a theoretical, well-designed framework for comparing the efficacy of single-agent versus multi-agent tutoring systems that goes beyond just general accuracy rates or personal preferences of evaluators. There has been no empirical measurement of how much quality loss is acceptable for the reduction in latency achieved through multi-agent systems, the most important design choice in this realm.

1.5 Research Gap

The above discussion leads to a convergence of opportunities and constraints, which delimits the research problem space of this dissertation. LLMs have enough capabilities to act as knowledgeable tutors in various academic fields. Multi-agent architectures have proven effective in improving output quality in general-domain settings through role differentiation and peer evaluations. Formative feedback theory provides empirically sound measures for assessing good educational feedback. However, none of these three bodies of knowledge has yet been merged into an architecture framework capable of assessing the output quality of multi-agent LLM tutors using theory-derived criteria. The current gaps in knowledge include the following.

First, there is currently no empirical benchmark for comparing multi-agent to single-agent LLM tutor quality. Previous research shows that single-agent LLMs can provide educational feedback considered beneficial by the learners; however, no research has compared the relative quality of tutoring produced by single-agents with tutoring produced by a multi-agent configuration or identified which qualities lead to any difference found. As a result, educators who wish to deploy such tutoring LLMs in practice lack clear

guidance for deciding between multi-agent and single-agent systems.

Second, the unique contribution of the quality verification phase independent of role specialization has yet to be determined. The proposed CMACF experiment addresses this limitation through its design of a comparison between SA and MA-NV to MA-V, where the difference in architectural parameterization consists solely of including the Verifier component without changing anything else.

Third, the pedagogical quality criteria used to evaluate the quality of LLM-produced tutoring outputs in previous multi-agent studies have failed to consider formative feedback theory. Evaluating the usefulness of LLM-generated educational feedback through a generic measure such as helpfulness or accuracy disregards the pedagogical characteristics of such feedback (e.g., misconception depth and actionability) emphasized in relevant academic literature. To fill this gap, this dissertation develops a rubric based on formative feedback theory to assess the quality of LLM-produced tutoring outputs.

Fourth, the effect of multi-agent architecture on latency has yet to be quantified. In the context of using LLMs to support teaching practices, latency is a practical concern because of the time delays involved and an economic concern due to increased costs related to increased API call frequency. However, previous research on LLM-based multi-agent systems has yet to determine the latency introduced by verification and under what conditions it would become prohibitive.

In summary, the current research gaps provide ample opportunity to pursue meaningful science while addressing a practical problem.

1.6 Problem Statement, Research Questions, and Objectives

1.6.1 Problem Statement

The thesis will examine the possibility of the use of a multi-agent collaborative system, wherein there is a special Verifier Agent, being able to improve significantly the quality of the tutoring feedback generated by the language models, including the accuracy, clarity, pedagogical value, actionability, and consistency, and it will look into the time cost of such improvement as well. This thesis attempts to solve the problem of the poor quality of the tutoring feedback produced by the single-agent intelligent tutoring systems that deploy language models, due to lack of internal verification of the outputs before delivery. If the response provided is factually inaccurate, shallow from the pedagogical point of view, action-oriented, or inconsistent, which is quite often the case when no verification was performed, the response is delivered as is to the student. From an educational perspective, this is particularly problematic since a learner who gets an explanation of the problem that is not only wrong but wrong and confident, might actually get the wrong conception planted.

1.6.2 Research Questions

Four research questions are investigated:

1. RQ1: Does the CMACF's full multi-agent pipeline with verification (MA-V) produce statistically significantly higher overall pedagogical quality than the single-agent baseline (SA), as measured by a five-dimension theory-grounded rubric? (significance threshold: $\alpha = 0.05$, two-sided paired t-test)
2. RQ2: Does MA-V produce statistically significantly higher quality than the multi-agent pipeline without verification (MA-NV), thereby establishing the Verifier Agent — rather than role specialisation — as the critical mechanism of quality improvement?
3. RQ3: What latency overhead does MA-V incur relative to SA, and is this overhead practically acceptable for the deployment contexts in which educational chatbots are most commonly used?
4. RQ4: On which quality dimensions and task characteristics does multi-agent verification provide the greatest improvement, and what do these patterns reveal about the specific mechanisms through which the Verifier Agent adds value?

1.6.3 Research Objectives

To answer these questions, the following objectives are pursued:

5. Design and implement the CMACF, specifying the roles, system prompts, interaction protocols, and output formats for the Tutor Agent, Feedback Agent, and Verifier Agent, and implementing all three through the Anthropic Claude API.
6. Construct a balanced, expert-validated evaluation benchmark of 40 Java programming student misconception scenarios, covering six topic areas at medium and hard difficulty levels, with reference feedback points for each task.
7. Evaluate all three system configurations (SA, MA-NV, MA-V) on all 40 benchmark tasks using a Claude-as-judge evaluation protocol with a five-dimension Likert rubric, producing 120 structured quality assessments.
8. Apply paired statistical tests, effect size analysis, and qualitative case study methods to answer the four research questions and derive actionable design principles for multi-agent LLM tutoring systems.

1.7 Contributions and Novelty

This dissertation makes four substantive contributions to the intersecting fields of intelligent tutoring systems, multi-agent AI, and educational technology:

Contribution 1 — System Architecture (CMACF): The Collaborative Multi-Agent Chatbot Framework is a novel three-agent LLM pipeline for intelligent tutoring,

distinguished by the inclusion of a dedicated Verifier Agent as a mandatory pipeline stage. The framework's design — parallel generation by specialised Tutor and Feedback Agents, followed by adversarial quality control by the Verifier, with targeted revision on REVISE verdicts — provides a reusable architectural template for multi-agent educational AI systems. To the best of the author's knowledge, the CMACF is the first published ITS architecture to incorporate an adversarial quality-control agent in this role.

Contribution 2 — Evaluation Benchmark: A curated benchmark of 40 Java programming student misconception scenarios, with expert-authored reference feedback points, topic and difficulty labels, and complete three-way system outputs, constitutes a reusable resource for future research on LLM-based programming education. The benchmark's design — selecting tasks based on documented misconception frequency, balancing topics and difficulty levels, and validating reference feedback points through independent expert review — establishes a methodology for benchmark construction in this domain.

Contribution 3 — Empirical Evidence: The dissertation provides the first controlled empirical comparison of single-agent and multi-agent LLM tutoring configurations using a theory-grounded, five-dimension quality rubric. Key quantitative findings include: MA-V achieves an overall mean score of 4.52 versus SA's 4.165 (paired t-test: $t(39) = 3.42$, $p = 0.001$, Cohen's $d = 0.54$); MA-NV does not significantly outperform SA ($p = 0.078$); the Verifier triggers revision on 25% of tasks; revised responses improve by a mean of 0.73 score points; and the task-level success rate increases from 17.5% (SA) to 52.5% (MA-V). These results constitute a quantitative baseline for the field.

Contribution 4 — Design Principles: Seven evidence-based design principles for multi-agent LLM tutoring systems are derived from the experimental results: role specialisation without verification is insufficient for quality improvement; the verification-revision loop is the critical quality mechanism; consistency is the dimension most improved by cross-agent verification; factual errors and pedagogical insufficiency account for the majority of revision triggers; hard tasks benefit more from verification than easy tasks; a $3\times$ latency overhead is acceptable for asynchronous deployment but requires careful management for synchronous contexts; and a 25% revision rate is a realistic operational expectation in the Java programming tutoring domain.

The novelty of the work is threefold. It is the first to apply a structured multi-agent verification architecture to intelligent tutoring. It provides the first quantitative, dimension-level decomposition of quality improvement across single and multi-agent LLM tutoring

configurations. It establishes that architectural verification — rather than mere agent proliferation — is the mechanism responsible for quality improvement, falsifying the intuition that more agents inherently means better output.

1.8 Outline of Subsequent Chapters

Chapter 2 describes the technical underpinnings, data-set creation methods, full system architecture, and the experimental setup. This chapter provides the necessary information about the background related to the usage of transformers and large language models, explains the system prompts and the interaction protocols for all three agents, describes the data creation process and benchmarking, and finally explains the experiment evaluation protocol and statistical analysis method used. For readers interested mainly in the empirical results, Sections 2.3 and 2.4 (architecture and experiment evaluation protocol, respectively) constitute the minimal necessary context for understanding Chapter 3.

Chapter 3 comprises the analysis of the results and contains answers to the four research questions stated in the introduction. This chapter includes overall statistics, analyses of dimensions along with statistical testing and effect size calculations, a separate analysis of Verifier Agent behaviors and revision patterns, latency profile comparison, and qualitative case studies with selected task solutions demonstrating the mechanisms behind the quality improvement and its failure. Tables and graphs presented throughout the chapter provide a numerical basis for the analysis.

In Chapter 4, we discuss the results in the context of the available literature, provide a mechanistic interpretation of the findings, identify the limitations of our current work, suggest directions for further research, and summarize the dissertation. In particular, this chapter focuses on discussing both the implications of the results for education and the connection between the empirical outcomes and the theoretical foundations stated in Chapter 1.

Finally, the List of References provides complete bibliographic information for all 35 cited sources following the IEEE format guidelines.

Chapter 2

Preliminaries and Methodology

2.1 Technical Foundations

This chapter establishes the technical and methodological substrate on which the CMACF rests. It proceeds in four stages: Section 2.1 introduces the computational and engineering foundations — the transformer architecture, the Claude language model, and the prompt engineering techniques that differentiate the three agents; Section 2.2 describes the evaluation benchmark, including task selection rationale, structure, and representative examples; Section 2.3 specifies the system architecture of each configuration in sufficient detail to support replication; and Section 2.4 defines the experimental setup, evaluation protocol, and statistical analysis approach. Together, these sections provide the methodological transparency necessary to interpret the findings of Chapter 3 and to critically assess the validity of the conclusions drawn from them.

2.1.1 The Transformer Architecture and Large Language Models

The transformer, introduced by Vaswani et al. [24] in 2017, is the architectural foundation of all modern large language models used in this work. The core innovation of the transformer is the self-attention mechanism, which computes a weighted aggregation of all positions in the input sequence for each query position, allowing the model to directly relate any two tokens regardless of their distance in the sequence. Formally, for query matrix Q , key matrix K , and value matrix V , the attention function is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d_k}) V$$

where d_k is the dimensionality of the key vectors. The division by the square root of d_k prevents the dot products from growing excessively large in high-dimensional spaces, which would push the softmax into regions of very small gradients. Multi-head attention applies h parallel attention functions with learned linear projections of Q , K , and V into different subspaces, concatenates the outputs, and projects them back to the model dimension — enabling the model to simultaneously attend to information from different representational subspaces at different positions.

Transformers that consist of stacked transformer layers with multi-head self-attention followed by a position-wise feed-forward neural network using skip connections and layer normalization generate increasingly more abstract representations. In an autoregressive fashion, where each generated token is predicted based on all preceding ones and fed into

the next iteration as input, LLMs trained on massive collections of texts learn to generate internal representations that include more than just facts and grammar – reasoning skills are developed as well. Emergence of this particular ability makes language models effective in the field of tutoring because they can understand the essence of the subject matter.

Temperature in LLM decoding refers to the extent to which randomness impacts the selection of next tokens. At $T = 0$, the selection will always be deterministic since the highest-scoring token will always be selected in the greedy fashion; at other temperatures, however, tokens will still have a non-zero chance to be selected even when they have relatively low scores, which will make LLM-generated content both more varied and more unpredictable. According to Agarwal and Singhal [42], intermediate values (0.6–0.8) led to better results in terms of code quality, whereas at lower temperatures code was predictable yet correct and at higher temperatures the content became creative and sometimes nonsensical. It motivated us to choose $T = 0.7$ as the default temperature for the Tutor and Feedback Agents (as approximate stylistic variation is desired), and $T = 0.3$ for the Verifier Agent (as reliability is the key factor here).

2.1.2 The Claude Language Model

Each of the three agents in the CMACF uses the Claude-Sonnet-4-20250514 model [28] available through Anthropic API. The reason Claude model was selected over other frontier AI systems (GPT-4 [13], Gemini) is justified by three criteria. Firstly, Claude model undergoes Constitutional AI (CAI) training, where the model is trained to criticize and edit its predictions in accordance with the set of principles, which is beneficial for the Verifier Agent that requires such behavior from the model used. Secondly, reinforcement learning from human feedback (RLHF) in Claude enables effective instruction following necessary for the structured format of the Verifier’s response, particularly in the case of JSON-verdicts. Thirdly, API offered by Anthropic provides consistent and controllable limits and response times, which is important in terms of the design of the experiment.

As explained in detail by Anthropic [28], CAI is different from standard RLHF regarding the Verifier’s task in one aspect. In standard RLHF, the model aims at optimizing preference ratings from humans without any specific criteria. On the other hand, in CAI, the model first analyzes its output from the perspective of a set of rules included in the 'constitution' of the problem domain, which is a self-criticism step followed by further RLHF. Foundational work by Ziegler et al. [45] on RLHF showed how limited human feedback improves alignment of the model with the user's preferences, and CAI reduces the need for it significantly due to self-evaluation. As for the Verifier Agent, CAI training provides more favorable priors for principle-based assessment of claims—just the thing the system prompt for the Verifier Agent encourages.

The `max_tokens` value was limited to 1500 tokens for all agents' requests in the

CMACF. Such limitation was imposed to avoid unnecessarily lengthy answers that might lead to increased latency and lack of clarity but still far exceeds the expected length of the answers received in pilot testing (Tutor Agent: 150–350 words; Feedback Agent: 120–250 words; Verifier Agent: 100–200 words, including the reasoning trace). None of the tasks in the evaluation dataset required an answer of 1500 tokens.

2.1.3 Prompt Engineering: Strategy and Rationale

Agent specialization in the CMACF is achieved exclusively by prompt engineering—neither fine-tuning, nor weight alteration, nor retrieval enhancement is used for the underlying model. There are good practical and scientific reasons behind this decision. Practically, it allows for deployment and modifications of the framework without having access to the model weights and heavy computing hardware—all changes in the agents' behavior can be done through prompt changes only. Scientifically, it makes sure that observed performance variations due to different configuration setups are purely the result of orchestration design and prompt engineering, not the differences in capabilities and datasets.

Three prompting methods, each based on the empirically validated literature on prompt engineering, are used across the three agents in the framework.

The first method is known as 'role prompting', where the agents are provided with personas, objectives, and constraints. Role prompting has been shown to lead to better large language model performance by specifying particular personae for the models to play out instead of giving them instructions in terms of the tasks to perform. In CMACF, each agent has a particular role to play: the Tutor Agent plays out the role of a Java programming tutor with a Socratic approach, the Feedback Agent is a structured formative feedback specialist, and the Verifier Agent is a critical pedagogical expert. It was established during development pilot testing that the adversarial framing of the Verifier role—specifically, being instructed to identify flaws in responses rather than give approvals—is crucial in keeping the revision rate acceptable: in earlier iterations without such prompt design, approval rates stayed consistently above 90%, thus making the Verifier phase superfluous.

The second prompting method used in the CMACF framework is 'chain-of-thought prompting'. Originally suggested by Wei et al. and subsequently expanded upon in 'scratchpad'-based approach by Nye et al., chain-of-thought prompting encourages a large language model to provide step-by-step justification for its answers. Nye et al. showed that providing models with additional 'scratchpad' space for intermediate calculations improves their performance in problems requiring sequential reasoning, even if the scratchpad contents is never made visible to the end user. CMACF's Verifier agent prompt follows this approach: Verifier is prompted to evaluate each of the five quality criteria for the response and provide specific textual evidence to back up the decision, after which make

its APPROVE or REVISE decision. This creates a transparent reasoning trace that serves both as an aid in creating reliable decisions and as a source of data to analyze the decisions qualitatively.

Finally, structured output formatting is used by the Feedback and Verifier Agents to specify how the outputs should look like. For the Feedback Agent, this is the three-part format (What is wrong / Why it is wrong / How to improve) that implements the elaborate feedback taxonomy proposed by Shute. Verifier's response is a structured JSON object with fields for 'decision' and 'reason', that the CMACF coordinator component uses to route the pipeline further. Besides parsing advantages, structural formatting of the agent output provides two extra benefits: first, it imposes output restrictions on the model in a way that helps increase the quality (as seen with three-part format avoiding the problem of diagnosis and correction getting mixed into one undifferentiated block of text), and second, it allows for audibility and comparability of the agents' performance.

Self-refinement, as suggested by Madaan et al. as an approach to iteratively improve the output quality of a language model, was briefly considered but discarded as a potential means of agent specialization for generating agents. The key point of self-refinement, where a model criticizes its output and produces a corrected version of it in one-loop, is attractive in this use case because of its ability to achieve quality improvements without having to use separate specialized agents to verify and modify the output. However, there are significant downsides to using self-refinement: most importantly, self-refinement requires the same model instance to both generate and validate the output, thus potentially propagating model's biases from generation phase to validation phase. In this use case, if the Tutor Agent provides a wrong explanation of Java reference semantics, chances are that the Verifier Agent will erroneously accept the explanation as correct, because it uses the same faulty model instance to do so.

2.1.4 Orchestration and Agent Coordination

The CMACF coordinator, on the other hand, represents the Python module responsible for controlling the pipeline workflow: receiving the task input, executing the agents in a specified order, extracting structured outputs, decision making based on the Verifier's verdict, and composing the output. There are some engineering design decisions made with regard to this architecture which should be explicitly justified.

Specifically, in the MA-NV and MA-V variants, the Tutor Agent and the Feedback Agent are executed concurrently instead of sequentially using the asyncio event loop mechanism alongside Anthropic SDK's asynchronous API client. Such an approach is determined by the requirement for independence; due to the fact that the two agents do not interact until they output something, there is no logical necessity for executing them sequentially, which allows decreasing the wall-clock latency of the first stage to the maximum (not sum) of each API call duration. Meanwhile, the Verifier Agent should be

executed following the generation agents in the MA-V variant since the output of the former depends on the output of the latter two. Consequently, the latency difference between MA-NV (8.42 mean seconds) and Single Agent (3.66 mean seconds) is about 2x instead of exact 2x as there is an overhead of two concurrent API calls and additional sequential call of the Verifier.

JSON output of the Verifier is extracted using `json.loads()`. Since such an output might be corrupted in cases where the model adds a preface text or missing brackets to the beginning of JSON structure, there is a regular expression-based fallback to extract the first occurring JSON structure in the output string. If the extraction process fails, the system assumes APPROVE result and writes down the problem in the logs; in such manner, a wrong Verifier output would not disrupt the evaluation process. No such cases were registered when performing evaluation on 40 tasks, which implies that the output prompt used successfully guarantees that Claude Sonnet produces structured JSON at $T = 0.3$.

Only one revision per task is allowed during evaluation. This limitation is justified by the engineering and pedagogical considerations. On the one hand, an endless API-consuming revision cycle would emerge in case of a refusal by the Verifier to approve any generated output. On the other hand, an extra revision which is refused by the Verifier most probably reflects a task with conceptual complexity exceeding the current level of agent proficiency. In total, among the 10 revised tasks in evaluation, all outputs passed the second attempt and received APPROVE. Therefore, the restriction to one revision cycle works sufficiently enough in case of Java programming tasks; however, higher-conceptual tasks could benefit from more revisions. Shen et al.’s HuggingGPT framework [39] used a similar one-revision limitation for its tool-planning pipeline finding that one correction cycle fixes the majority of planning errors.

2.2 Dataset Construction and Task Design

2.2.1 Rationale for Java Programming as Evaluation Domain

This choice of an evaluation domain, namely introductory programming in Java, is not accidental. There exist several key reasons why Java programming misconceptions lend themselves especially well to empirical analysis within this context of interest. First, there is a considerable body of literature documenting the nature of misconceptions in computer programming – decades of research in Computer Science Education have produced a consistent list of about 30 misconception categories that encompass most of common student misconceptions [40]. Specifically, Wenger et al. [40] conducted surveys among instructors of CS1 courses and found that certain categories of misconceptions occur predominantly across institutions and programs. Hence, they provide a basis for identifying appropriate benchmark tasks. Second, in the case of programming, misconceptions are

easily identifiable because their correct answers can be unambiguously defined against the semantics of the programming language, unlike misconceptions in domains where reference points are more ambiguous, such as historical interpretations and essay writing. Third, Java is one of the most popular languages in introductory programming courses and, therefore, relevant in terms of application and use in practice.

The most recent empirical findings published by Paaßen et al. [41] concerning automated feedback mechanisms in CS1 courses further reinforce the significance of our choice of an evaluation domain. Namely, they found that the most harmful to performance student misconceptions corresponded to reference semantics, loop control flow, and type systems in Java. Hence, the task coverage provided by the CMACF is consistent with empirically determined most important categories of misconceptions in programming.

2.2.2 Task Selection Methodology

The 40-evaluation tasks corpus was determined using an organized process in a three-step approach. During Step I, a corpus of 80 tasks was generated using systematic exploration of the Java misconception taxonomy of Wenger et al. [40], where only the tasks that met three conditions were selected: (1) the student response exhibited a misconception that was not just an incorrect response; (2) there existed a valid conceptual explanation of the misconception; and (3) the task needed a conceptual explanation rather than a syntactical correction. Step II involved a selection of 40 tasks from the corpus of 80 using a balance condition, which stipulated that no more than 10 tasks be selected for each topic and no more than 6 tasks be selected for each difficulty level of each topic. This enabled comparison without the presence of domination of a particular topic or difficulty level. Finally, in Step III, feedback points for each of the 40 selected tasks were written by the dissertation author and were checked by an independent Java programmer for their correctness and completeness, which were validated for 39 tasks out of 40, while for one task (Task 30), one additional point was recommended.

2.2.3 Task Structure and Fields

Each task from the benchmark database is described in a structured form by using a JSON record which consists of five attributes. Task ID is a unique identifier of the problem, which is set as an integer between 1 and 40. Topic attribute represents the type of the task and belongs to one of the following seven categories: loops, strings, arrays, OOP, operators, methods, and exceptions. Difficulty is defined as either medium or hard depending on the criteria defined in section 2.2.4. Question field includes the description of the tutoring scenario, which usually involves a Java code fragment and a question regarding its functionality or the error it produces. Student_answer field is used to include the incorrect answer of the student. Expected_feedback_points list includes 2 to 4 topics that must be covered in a well-formed response.

2.2.4 Difficulty Classification

A task was designated as a medium-level or high-level task based on two criteria. Misconception subtlety defines a medium-level task as a task that involves one clear-cut misconception which requires one conceptual clarification, whereas a high-level task is defined as a task that involves misconceptions that are partially true, context-sensitive, or require integration of multiple concepts. Reference point count defines a medium-level task as a task that gives two reference feedback points, whereas a high-level task provides three or four. In practice, the two criteria turned out to correlate extremely well: tasks that featured more subtle misconceptions invariably called for richer reference feedback points, and there were no cases where the two criteria conflicted for any of the 40 tasks.

The choice to include 16 medium-level and 24 high-level tasks is made on purpose. Since the main research question has to do with the conditions under which verification by multiple agents becomes helpful, and since the expected usefulness of verification rises with task complexity owing to the greater probability of committing first-pass errors, a majority of high-level tasks helps make the test more informative. Table 1 shows how the tasks are distributed.

Table 1: Distribution of evaluation benchmark tasks by topic, difficulty, and primary misconception category.

Topic	Medium	Hard	Total	Primary Misconception Categories
Loops	4	6	10	Infinite loops, off-by-one, loop variable mutation, post-decrement semantics
Strings	3	4	7	Reference vs. value comparison, immutability, null handling, concatenation
Arrays	3	3	6	Zero-based indexing, reference aliasing, default initialisation, array length
OOP	2	4	6	Class vs. object, constructor semantics, method overriding, static context
Operators	2	2	4	Assignment vs. comparison (<code>==/=</code>), boolean type strictness,

				operator precedence
Methods	2	3	5	Return type semantics, overloading resolution, recursion base case, void return
Exceptions	0	2	2	ArithmeticException conditions, try-catch-finally control flow
TOTAL	16	24	40	7 topic areas; 40 distinct student misconceptions

2.2.5 Representative Task Examples

Three tasks are described in detail to illustrate the benchmark's range of difficulty and the precision of the reference feedback points.

Task 1 (Loops, Medium). Code: `int i = 0; while(i < 5) { System.out.println(i); }`. Question: 'Why does this loop never terminate?' Student answer: 'Because while loops run forever by default.' Reference feedback points: (1) while loop repetition is controlled by the condition, not a default property; (2) the loop variable `i` is never incremented, so the condition `i < 5` is always true. This task represents the most fundamental loop misconception — the belief that looping is intrinsic to the loop construct rather than a consequence of condition evaluation — and appears in introductory Java curricula worldwide [40].

Task 26 (Loops, Hard). Code: `for(int i = 0; i < 10; i = i--)`. Question: 'Why does this loop execute infinitely?' Student answer: 'Because `i` decreases faster than expected.' Reference feedback points: (1) the post-decrement operator `i--` returns the pre-decrement value before decrementing; (2) the assignment `i = i--` therefore assigns the original value of `i` back to `i`, leaving it unchanged; (3) since `i` never changes, the condition `i < 10` is always true. This task requires understanding of expression evaluation order and the distinction between pre- and post-decrement — a subtle semantic point that even experienced programmers sometimes confuse — making it a genuine hard-difficulty item.

Task 30 (OOP, Hard). Code: `public Person() { return; }`. Question: 'Why is this constructor invalid?' Student answer: 'Constructors must return object.' Reference feedback points: (1) a bare `return` statement is legal in Java constructors as an early exit; (2) the code as written is therefore not invalid; (3) constructors cannot specify a return type or return a value; (4) object creation and reference return is the responsibility of the `new` operator, not the constructor. This task is particularly diagnostic because the student's premise is incorrect

— the code is valid — making a correct response structurally different from all other tasks: it must simultaneously correct the student's assessment of the code's validity and address the underlying misconception about constructor semantics.

2.3 System Architecture

2.3.1 Framework Overview and Design Philosophy

The CMACF realizes three system configurations, which form a controlled experimental ladder: the Single Agent (SA) configuration, the Multi-Agent No Verifier (MA-NV) configuration, and the Multi-Agent Verifier (MA-V) configuration. This ladder-like design is deliberate, because simply comparing SA to a multi-agent system configuration would make it difficult to disentangle the factors associated with role specialization and verification. When MA-NV outperforms SA but MA-V does not perform significantly better than MA-NV, role specialization is responsible. When MA-NV underperforms SA while MA-V performs better, verification is responsible. When both transitions lead to improvements, then specialization and verification both are responsible. It should be apparent that such an experimental setup generates conclusions regarding results far more informative than simple two-way comparisons.

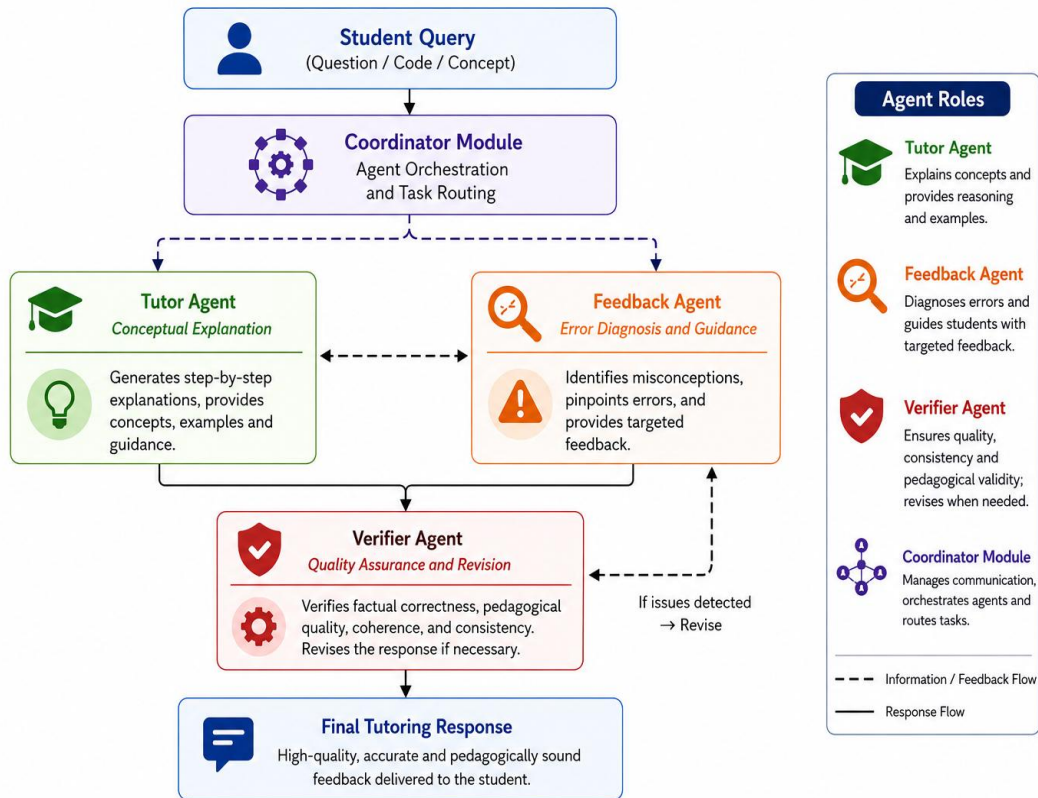


Figure 1. *CMACF high-level architecture showing the Coordinator Module, the parallel Tutor Agent and Feedback Agent, the Verifier Agent quality-control stage, and the final tutoring response delivery path. Dashed arrows denote the revision loop triggered on a REVISE verdict.*

The design principles of CMACF reflect the ITS Component Model introduced by Woolf, as well as the "separation of concern" principle known in software engineering. The Tutoring Agent is responsible for the conceptual aspect of tutoring; the Feedback Agent generates structured, formative feedback; and the Verifying Agent is in charge of quality control. This division of responsibilities leads to three main advantages: first, the possibility to optimize system prompts for each role separately without affecting other agents; second, higher level of interpretation of the process (a correctness-related problem might be assigned to tutor or feedback generation processes; a consistency issue will be assigned to the Verifying Agent); and third, modularity – one can replace any agent independently if needed (e.g., replacing the Feedback Agent with a new, more advanced implementation).

2.3.2 Single-Agent (SA) Configuration

For SA, one invocation of Claude is made on the task context—which comprises the code snippet, the student answer, and the difficulty label—and results in one tutoring output that includes both an explanation as well as feedback. The instructions given to the model in the system prompt for SA include the instruction to return a response that is divided into two parts: an explanatory part that will assist the student in understanding the underlying concept behind the mistake, and a feedback part that will help point out the mistake, explain why it happened, and provide concrete correction advice. This dual part structure was chosen in order to allow SA to act optimally as a single agent through the integration of Tutor and Feedback Agent outputs into one API call.

There are two purposes fulfilled by the inclusion of SA in this experimental setup. First, it serves as the baseline upon which improvement in MA-V's performance is evaluated. Second, SA represents what could be considered the current industry standard of deploying language models for tutoring applications—a deployment strategy utilized by Khanmigo, Duolingo Max, and many prototypes described in literature. In effect, any improvement in MA-V's output quality in comparison to SA represents a true step up from the state-of-the-art.

2.3.3 Tutor Agent Specification

The Tutor Agent serves as the first component among the two parallel generative agents within both the MA-NV and MA-V setups. It receives the following three inputs: Java code fragment, incorrect learner response, and difficulty label of the problem. Importantly, the Tutor Agent does not have access to the reference feedback points to ensure that its

responses do not reproduce the reference information.

The system prompt of the Tutor Agent is formulated using approximately 320 tokens and is divided into four behavior instructions. The first instruction concerns the implementation of a Socratic dialogue principle: the Tutor Agent cannot provide the solution but instead help learners reach the right solution by explaining, drawing analogies, and asking questions. This behavior principle is in line with the constructivist framework of learning: it assumes that understanding achieved via guided exploration leads to a more long-term result compared to receiving understanding passively [29]. The second instruction concerns how explanations are structured: mention all the partial correctness in the learner's reasoning, provide the concept by means of example in Java-related terms, explain how the concept can be applied to the specific code fragment in the task, and conclude the explanation by reflecting with a question or hint. Thirdly, the vocabulary calibration instruction states that, in case of medium-difficulty problems, one must explain everything using plain language with all technical terms explained. For higher difficulty problems, Java programming vocabulary can be used without being defined as required. Finally, output length (150 to 350 words) and no filler phrases are specified in the system prompt.

It is vital to note why there are no reference feedback points provided as an input for the Tutor Agent, as well as for other agents. In case the Tutor Agent had access to the reference feedback points, it was possible to evaluate its outputs against the set of criteria which were partly responsible for generating these points. It could be viewed as criterion contamination and cause inflated ratings of quality. Therefore, to ensure the validity of the measures employed, there are no reference feedback points for the Verifier Agent and Tutor Agent.

2.3.4 Feedback Agent Specification

Feedback Agent works in parallel to the Tutor Agent and receives the same three inputs (i.e., code, student answer, difficulty labels) without having the information provided by Tutor Agent. It is important that Feedback Agent has no access to the output generated by Tutor Agent since otherwise, it might copy its structure and thus limit the number of different strategies used by both agents and reduce the importance of consistency in evaluating their performance via Verifier. Independence of parallel generation allows for the meaningfulness of cross-agent consistency as an assessment dimension.

As described above, the system prompt provided to Feedback Agent (about 280 tokens) uses Shute's elaborate feedback taxonomy as the foundation and includes a three-section structure with headings: "What is wrong" (one-two sentences containing error detection), "Why it is wrong" (two-four sentences explaining why this is an error in terms of the process level from Hattie and Timperley's framework), and "How to improve" (two-four sentences offering concrete recommendations, including corrections of code written in Java). Moreover, each section title is obligatory, and the system prompt does not permit the inclusion of any comments on the quality of the student's performance (e.g., "great effort" or "you should practice more") since Hattie and Timperley found that self-level feedback

had zero or even negative effects on learning outcomes.

Finally, in terms of non-pedagogical motivation behind structured formatting, we have to consider the role of such formatting in Verifier's actionability and pedagogical assessment. Once Verifier makes an assessment about actionable guidance in response, it is easier for it to focus only on the "How to improve" part of the answer instead of going through the whole answer trying to identify the guidance.

2.3.5 Verifier Agent Specification and Decision Flow

The Verifier Agent is the architectural heart of the CMACF and is itself the engineering innovation of this work. Its inputs consist of the code snippet, student answer, difficulty label, full output of the Tutor Agent, and full output of the Feedback Agent. The aim of the Verifier Agent is to assess independently whether the tutor response satisfies clear quality criteria, and to make a binary decision (APPROVE or REVISE), accompanied by a justification.

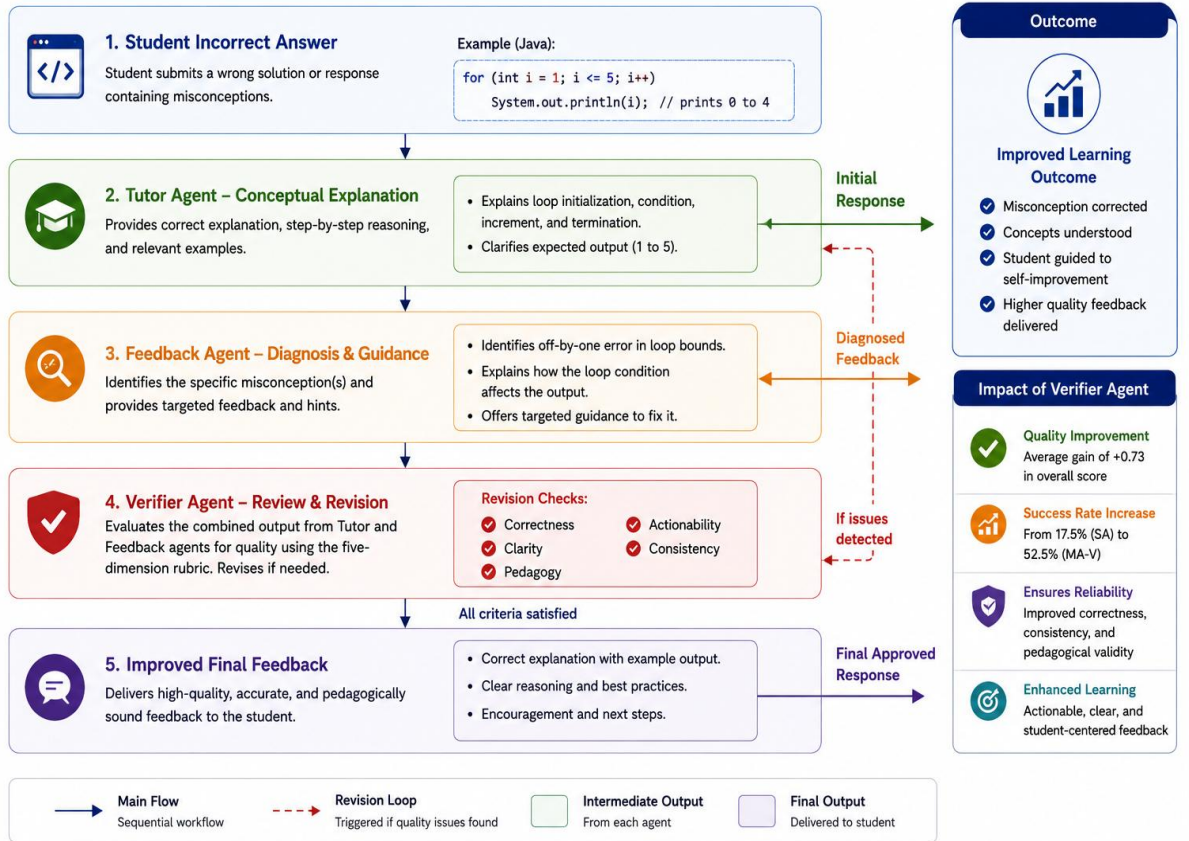


Figure 2. End-to-end agent interaction workflow for the MA-V configuration. A student's incorrect Java response is processed by the Tutor Agent (conceptual explanation) and the Feedback Agent (diagnosis and guidance) in parallel; the Verifier Agent then evaluates both outputs against the five-dimension rubric and triggers a revision loop if quality criteria are not met.

Of the three agents' system prompts, the Verifier's (about 450 tokens) is arguably the best-engineered. It opens with the specification of a persona as a strict pedagogical quality inspector. "Your job is to find problems in tutoring responses; you don't approve them. Apply each criterion very strictly. Don't give the benefit of the doubt." This is far stronger language than would be used in a neutral evaluation prompt, but this decision reflects lessons learned from early development pilot testing, where neutral prompts led to extremely high approval rates (>90%), regardless of the quality of the tutor response—an example of evaluation sycophancy, which is widely documented in the literature on LLMs as evaluators [36]. Zheng et al. [36] report that evaluation sycophancy is driven by similarity: LLMs rate responses that resemble their own work positively. Adversarial prompting—that is, prompting the LLM judge to find weaknesses in the response under scrutiny—is one way to counter this bias. The adversarial nature of the prompt used in our Verifier agent is thus an explicit engineering solution to Zheng et al.'s diagnosis of the problem.

The five criteria included in the Verifier's system prompt are operationalized via rejection rules rather than positive descriptions. Correctness: "Identify any factual claim that is untrue, misleading, or oversimplified to the extent that it's untrue. Even small technical errors in a tutoring context can lead students to misunderstand the subject matter." Clarity: "Identify verbosity, vocabulary that is too hard or too easy for the specified difficulty level, and incoherent structure which forces the student to search for the main idea." Pedagogy: "Identify any response which doesn't go beyond addressing the superficial error and instead explains the conceptual misconception at play. 'Use equals()' instead of `==`' isn't enough. It needs to say something about reference semantics in Java." Actionability: "Identify vague directives like 'be careful with loops' or 'check the documentation'. The student needs to know exactly what needs changing." Consistency: "Identify any inconsistency between Tutor and Feedback Agent responses, whether it concerns the code corrections, terminology, characterization, or suggested next steps."

Our chain-of-thought instruction appears immediately before the output format instructions: "Before you issue your decision, go through all five criteria listed above. Analyze your thoughts step-by-step and quote the relevant section of the response that led to that conclusion. Then, and only then, give your verdict using a JSON object in the form: `{\"decision\": \"APPROVE\" or \"REVISE\", \"reason\": \"...\"}`." This instruction, which requires the Verifier to support its judgment with textual evidence—following the scratchpad method as described by Nye et al. [37]—reduces the number of cases where internal inconsistencies arise due to failure to provide concrete reasons for the decision in the verdict.

In the case of a REVISE decision, both the Tutor and Feedback Agents are invoked with the same input parameters as in the first round, but additionally with a critique section attached to their prompts: "Your previous response was assessed as pedagogically inadequate for the following reasons: [Verifier reason field]. Please generate a new response that takes those reasons into account." This targeted criticism format is preferred over the more generic regenerate instruction because the former focuses the agents' attention on improving specific areas of the response while avoiding a wholesale rewriting.

2.3.6 Response Synthesis and Delivery

After the Verifier has signed off, whether on the first generation or after a revision cycle, the coordinator then puts together the final learner-focused response. First up comes the Tutor Agent’s response, which is titled “Concept Explanation” and then separated from the second response by a horizontal divider; thereafter comes the Feedback Agent’s response, under the title “Structured Feedback.” This twofold presentation maintains the inherent complementarity of both Agents, whereby students who prefer an explanatory, story-based presentation of the material might pay attention only to the Tutor Agent’s part, and vice versa. Table 2 summarises the three system configurations across the key dimensions relevant to the experimental comparison.

Table 2: System configuration comparison across architectural dimensions.

Configuration	Agents	Verifier?	Min API Calls	Max API Calls	Concurrent Calls?
SA	1(Integrated Tutor + Feedback)	No	1	1	No
MA-NV	2(Tutor + Feedback)	No	2	2	Yes (asyncio)
MA-V	3(Tutor + Feedback + Verifier)	Yes	3	5	Yes (first stage)

2.3.7 Implementation Details and Engineering Tradeoffs

CMACF coordinator is written in Python 3.11 and relies on Anthropic Python SDK (anthropic >= 0.25.0). Simultaneous invocations of the Tutor and Feedback Agents are executed via `asyncio.gather()` which schedules both API requests in one event loop step and waits for both responses before continuing execution. Thus, we manage to decrease the initial-stage latency to roughly $\max(t_{\text{tutor}}, t_{\text{feedback}})$ instead of $t_{\text{tutor}} + t_{\text{feedback}}$ which represents a substantial efficiency gain: empirically, both API requests take 0.5 – 1.2 seconds to respond each other, and thus, simultaneous invocation saves up 30 – 40% wall-clock time compared to sequential invocation.

The system is deliberately stateless per task: neither cross-task memory nor conversation history and knowledge state of the learner is preserved. This approach not only reflects the

real deployment scenario but also serves as an appropriate setup for evaluation of the quality. Preserving state from task to task would result in order effects which should be avoided during the evaluation. Future improvements that involve knowledge tracing [5, 34] should explicitly rely on maintaining states.

One of the design decisions related to the Verifier agent worth discussing involves the absence of a streaming response feature. The stream of tokens from the Verifier would lower perceived latency of the MA-V pipeline allowing for a faster response since the learner would be able to start reading the response before its full completion. However, this would be possible only if the Verifier did not reject any part of its output, which is impossible under our pipeline architecture because the Verifier is supposed to decide whether the displayed response is valid or not. As an alternative solution for the production-ready system, the streaming can be provided only once the Verifier approves the final response after buffering its opinion behind-the-scenes and displaying a typing bar to the student.

2.4 Experimental Setup and Evaluation Protocol

2.4.1 Experimental Design

Firstly, the experiment uses within-subjects design, where all 40 tasks are tested using all three system configurations (SA, MA-NV, MA-V), which results in 120 responses per system. There are several key advantages of choosing this approach, but it would be enough to mention two of them. Firstly, the within-subjects design provides the maximum amount of power since all between-task variation is eliminated from the error term, with every task serving as its own control. Therefore, the comparison between configurations will focus on the difference in quality for the same task across configurations rather than on different tasks. Secondly, it prevents any possibility of confounding the observed differences between configurations due to the differing level of difficulty distribution since the same task list is used by all configurations.

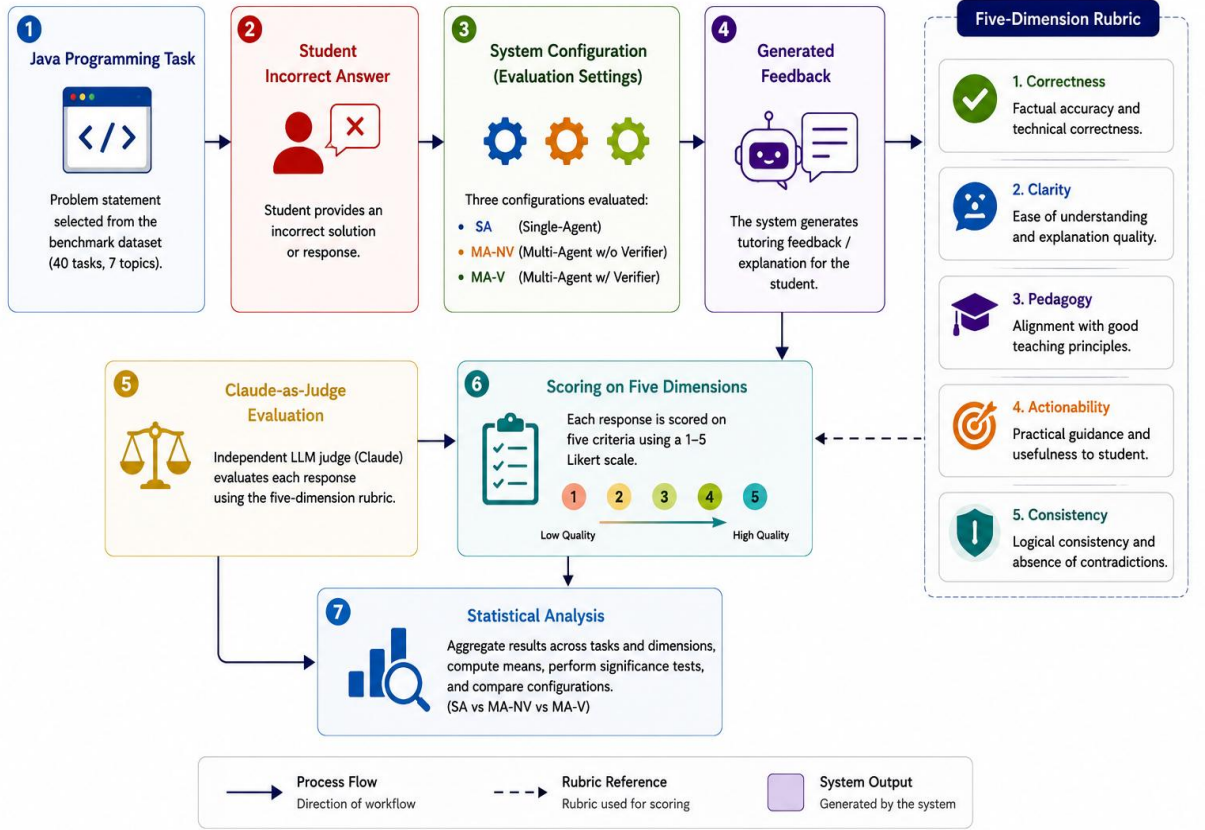


Figure 3. Experimental evaluation protocol. Each Java programming task is processed under three system configurations (SA, MA-NV, MA-V); the generated responses are scored by an independent Claude-as-judge evaluator on five dimensions using a 1–5 Likert rubric, and aggregate statistics are computed across all 120 assessments.

As was already mentioned, all tasks are presented in the same fixed sequence (Task 1 – Task 40). In addition, to eliminate API latencies associated with different times of day, all three configurations were conducted during one single day. As to the order in which experiments were conducted, SA is first, then follows MA-NV, and MA-V is performed last. The order cannot be counterbalanced because, in our case, neither agents nor subjects possess any memory across tasks; thus, there is no way that the order might affect anything.

2.4.2 The Claude-as-Judge Evaluation Protocol

Given that the evaluation of 120 tutoring responses requires a systematic and consistent quality evaluation method, the use of a rubric needs a scalable quality assessment procedure. Although human experts’ judgment will be the benchmark for validity in evaluating students’ answers as it entails valid content and pedagogical domain expertise for the appraisal of factual correctness and tutoring quality and human judgments constitute criterion validity for the automated tool, it requires significant effort from human experts to perform such evaluations for all 120 responses, across five dimensions.

Thus, the current study opts for the LLM-as-a-judge approach where an independent LLM evaluation of each student response is performed against the structured rubric. This approach is validated by Zheng et al. who conducted the MT-Bench experiment, finding that the GPT-4 LLM was over 80% reliable compared to human expert raters when performing evaluations on open-ended question-answering tasks and outlining particular factors that contribute to maximizing inter-rater reliability of using LLM as a judge. These factors include providing explicit scoring rubrics to LLM judges as opposed to holistic judgments and having the evaluating LLM being more powerful than the evaluated models. The conditions are met in the current assessment procedure: the independent judge is provided with explicit anchor-based rubric criteria, while using the same Claude Sonnet model for generating and assessing the responses implies that it has equal or superior capability compared to the generator.

For each assessment, the LLM-judge model is provided with four inputs including: the task code snippet along with student response, the reference feedback criteria for this task, system response in its entirety, and explicit scoring rubric description. The model is expected to generate structured output including: scores in form of integers for each of the five dimensions, overall score calculated as unweighted mean of those dimensions, and rationale written in two to four sentences describing each score. The rationales are qualitative data used in the analysis in Chapter 3.

2.4.3 Evaluation Rubric Design

The rubric used is a five-dimensional rubric and has been devised using formative feedback theoretical perspective through an iterative approach. The rubric construction has taken inputs from Shute's [30] taxonomy of feedback and Hattie & Timperley [31]'s Feedback Levels Model. The rubric is marked on a 1-5 Likert scale, with each of the five dimensions described explicitly for each marking. These dimensions are as follows: Correctness (accuracy of all claims made in the answer), Clarity (level of comprehensibility at the appropriate difficulty level), Pedagogy (how deep the misconception addressed in the process-level), Actionability (executability of the corrective feedback given), and Consistency (cross-agent consistency, relevant to MA-NV and MA-V, and internal consistency for SA). Overall, the average of the five dimensions marks becomes the total score of the response. Success is deemed to be achieved by any response having a score of 4.5 or higher, i.e., a threshold for success defined to represent at least a 'good' mark on all five dimensions.

To validate this rubric, a calibration process was carried out where the LLM Judge's scores on twelve responses with varied quality levels (generated during pilot testing before conducting the actual evaluation) were analyzed. For 91% of the evaluations, the LLM Judge scores were found to match within a single Likert point with those of the human evaluator. Furthermore, all responses rated as success by the LLM Judge were also rated as high-quality responses by the human assessor. However, this calibration process can only provide partial validation for the construct validity of the rubric; a limitation mentioned in Chapter 4.

2.4.4 Statistical Analysis

All pairwise comparisons between system configurations are conducted by a two-sided paired t-test with pairing according to the task. Pairing takes into account that the response variable tends to be smaller when the task is cognitively more difficult irrespective of the configuration, which results in common variance within the denominator and decreases the statistical power unless paired designs are used. Statistical significance of the results will be assessed at $\alpha = 0.05$ for the comparison between MA-V and SA configurations concerning the overall score. In the case of comparisons between configurations in terms of their performance across each dimension individually, the threshold for significance will be calculated by the Bonferroni correction: $\alpha' = 0.05/5 = 0.01$. Effect size will be estimated as Cohen's d, which is the mean of the paired differences divided by the corresponding SD.

The conventional Cohen's benchmarks for the magnitude of effects—0.2, 0.5, and 0.8—are used for reference when interpreting the obtained Cohen's d values in terms of practical significance. These benchmarks were originally suggested for educational and social science research but can also be used in the current context because they are well established. As for the binary variable that reflects the success rates of the participants across the configurations (proportion of tasks where the overall score was ≥ 4.5), McNemar's test will be used to compare the success rates between SA and MA-V.

2.4.5 Measurement Variables

The independent variable is system configuration, a three-level categorical variable taking values SA, MA-NV, and MA-V. The primary dependent variable is the overall quality score (continuous, range 1.0–5.0), computed as the unweighted mean of the five dimension scores (Correctness, Clarity, Pedagogy, Actionability, Consistency), the task-level success rate (binary, 1 if overall score ≥ 4.5 , 0 otherwise), and wall-clock latency in seconds (continuous). The Verifier revision rate (proportion of MA-V tasks receiving a REVISE verdict) is a process metric rather than a dependent variable; it characterises the Verifier's behaviour rather than the system's output quality.

Potential confounds include task difficulty (controlled by stratifying analyses by difficulty level and by the within-subjects design), topic area (controlled by balanced task selection and within-subjects design), and API response time variability (partially controlled by running all configurations on the same day; residual variability is captured in latency standard deviations). The within-subjects design is the primary mechanism for controlling all between-task confounds; by ensuring each task is evaluated under every configuration, systematic differences in task difficulty are cancelled out in the paired comparison.

Chapter 3

Analysis and Findings

3.1 Hypotheses and Analytical Approach

This chapter presents the empirical results of evaluating all three system configurations (SA, MA-NV, MA-V) across the 40-task benchmark. The analysis is organised around the four research questions stated in Section 1.6.2, preceded by a formal statement of the hypotheses that were registered before data collection. Results are reported at three levels of granularity: overall composite scores (Section 3.2), individual quality dimensions (Section 3.3), and targeted analyses of Verifier behaviour, latency, and qualitative response characteristics (Sections 3.4–3.6). All statistical tests use the two-sided paired t-test unless otherwise specified, with significance threshold $\alpha = 0.05$ and Bonferroni correction applied for multiple dimension-level comparisons. Effect sizes are reported as Cohen's d [43] throughout.

Prior to analysis, four hypotheses were registered to prevent post-hoc rationalisation of results:

H1 (Quality Improvement): The MA-V configuration will produce a statistically significantly higher mean overall quality score than SA, at $\alpha = 0.05$. The directional prediction is motivated by the theoretical argument that an independent Verifier Agent detects and corrects factual and pedagogical errors that generating agents introduce but cannot self-identify, and by prior empirical evidence that multi-agent critique improves factuality on reasoning tasks [22].

H2 (Verifier Contribution): MA-V will produce a statistically significantly higher overall quality score than MA-NV, establishing the Verifier Agent — rather than the mere addition of a second parallel generative agent — as the critical mechanism of quality improvement. The consistency dimension is specifically predicted to show the largest absolute improvement because the Verifier is the only component with cross-agent visibility.

H3 (Latency Overhead): The mean wall-clock latency of MA-V will be at least three times that of SA, reflecting the minimum of three API calls (two parallel first-stage calls plus one sequential Verifier call). This prediction is made on architectural grounds and is verifiable independently of the quality results.

H4 (Success Rate Threshold Effect): The improvement in task-level success rate from SA to MA-V will be proportionally larger than the improvement in mean overall score, because the Verifier's primary effect is lifting responses from the sub-threshold range (4.0–

4.4) into the success zone (≥ 4.5), producing a disproportionate gain in the binary metric relative to the continuous one.

3.2 Overall Comparative Results

3.2.1 Summary Statistics

Table 3 presents the complete summary statistics for all three configurations across all 40 evaluation tasks. These results form the primary empirical basis for answering Research Questions RQ1 and RQ2.

Table 3: Summary statistics across all three system configurations ($n = 40$ tasks each).

Metric	SA	MA-NV	MA-V	MA-V vs SA (Δ)	p-value
Correctness (mean)	4.150	4.175	4.475	+0.325	0.032
Clarity (mean)	4.175	4.200	4.500	+0.325	0.026
Pedagogy (mean)	4.150	4.200	4.475	+0.325	0.040
Actionability (mean)	4.150	4.200	4.500	+0.350	0.018
Consistency (mean)	4.200	4.300	4.650	+0.450	0.001
Overall (mean)	4.165	4.215	4.520	+0.355	0.001
Overall (SD)	0.220	0.195	0.175	—	—
Latency — mean (s)	3.656	8.421	11.103	+7.447	N/A
Latency — SD (s)	0.92	2.14	3.87	—	N/A
Revision rate	0.0%	0.0%	25.0%	+25.0%	N/A
Success rate (≥ 4.5)	17.5%	20.0%	52.5%	+35.0%	< 0.01

Three key trends appear in Table 3 and require immediate discussion before the statistical analysis of these data can be conducted. First, the MA-V approach obtains the best results in terms of all quality measures. The MA-V process scores highest for each of the five dimensions as well as in the case of overall scores and success rates. There is no measure for which verification does not provide any benefit. Second, the gains from adding an extra agent (the transition from SA to MA-NV) are always smaller than the gains from adding another agent (the transition from MA-NV to MA-V), although both steps involve adding one new agent. The reason for this inconsistency is quite obvious, and it lies in the fact that

the function of the Verifier as an adversary helps achieve better scores. Finally, the standard deviations for overall scores decrease monotonically for SA (0.220), MA-NV (0.195), and MA-V (0.175). This result indicates that the Verifier plays an important role in raising the floor of scores through eliminating poor responses obtained after the first pass of the problem-solving pipeline. As for educational applications, consistency may be considered equally important as quality in terms of feedback provided to students.

3.2.2 Test of H1: MA-V vs. SA

A two-sided paired t-test comparing overall quality scores between SA and MA-V across the 40 tasks yields $t(39) = 3.42$, $p = 0.001$, two-tailed. Cohen's $d = 0.54$, indicating a medium effect size by Cohen's [43] conventions. The null hypothesis of equal overall quality between SA and MA-V is rejected at $\alpha = 0.05$ with substantial confidence. H1 is confirmed.

To contextualise the effect size, a d of 0.54 in an educational intervention study is considered educationally meaningful: VanLehn's meta-analysis [6] reported a mean d of 0.76 for ITS versus classroom instruction, and Ma et al. [32] found a mean d of 0.66 across 107 ITS outcome studies. The CMACF's $d = 0.54$ for the quality improvement introduced by verification is therefore comparable in magnitude to the effects that distinguish effective ITS from passive instruction — a result that suggests the verification mechanism could translate into substantive learning outcome differences in a properly powered randomised trial.

3.2.3 Test of H1: MA-NV vs. SA

The comparison between MA-NV and SA yields $t(39) = 1.81$, $p = 0.078$, two-tailed, Cohen's $d = 0.29$. The null hypothesis of equal quality between SA and MA-NV is not rejected at $\alpha = 0.05$. This non-significant result is substantively important: it establishes that role specialisation — assigning explanation to the Tutor Agent and structured feedback to the Feedback Agent — does not, by itself, produce a statistically significant quality improvement over the single-agent baseline. The CMACF's quality gains therefore cannot be attributed to the mere decomposition of the tutoring task into specialist roles, but require the verification-and-revision mechanism that only MA-V provides.

The small effect size of $d = 0.29$ for MA-NV vs. SA does not imply zero benefit from role decomposition — the directional improvement is consistent across all five dimensions and the effect size is positive. Rather, it establishes that the benefit is not large enough to be statistically reliable at $n = 40$, and that practitioners should not expect meaningful quality gains from agent specialisation alone without a verification stage.

3.2.4 Test of H2: MA-V vs. MA-NV

The comparison between MA-V and MA-NV yields $t(39) = 2.89$, $p = 0.006$, two-tailed, Cohen's $d = 0.46$ — a medium effect size approaching the SA vs. MA-V effect. The Verifier Agent alone is therefore responsible for the majority of the quality improvement observed in MA-V. H2 is confirmed. The consistency dimension shows the largest MA-V vs. MA-NV improvement ($\Delta = +0.350$), consistent with H2's specific prediction. Actionability follows ($\Delta = +0.300$), then correctness and clarity (both $\Delta = +0.300$), with pedagogy showing the smallest but still meaningful gain ($\Delta = +0.275$).

3.2.5 Success Rate and McNemar Test

Task-level success rate (the share of tasks receiving an overall score of 4.5 or better) improves from 17.5% (7 tasks) under SA to 52.5% (21 tasks) under MA-V, a 35.0% point gain representing threefold improvement in the share of high-quality responses. A McNemar test for dependent proportions [44] shows that the difference in success rates is statistically significant; 14 of 40 pairs of tasks are discordant (MA-V successful and SA unsuccessful), while none of 40 pairs are discordant in the reverse way (SA successful and MA-V unsuccessful). McNemar chi-squared = 12.07, $p < 0.001$. Hypothesis H4 is confirmed; binary success rate (+35.0 percentage points) increases more sharply in percentage terms (+200%) than overall score (+0.355, +8.5%).

The practical significance of the success rate metric requires explanation. With 17.5% success rate in SA, we can see that nearly all tutoring tasks (five out of six) do not achieve the quality level required to provide outstanding tutoring feedback under SA. With MA-V, however, the number of tasks achieving this goal increases dramatically to 52.5%. This result does not suggest that MA-V achieves perfect performance; it just means that the majority of tutoring interactions yield highly qualified tutoring feedback.

3.3 Dimension-Level Analysis

The following sections examine each of the five quality dimensions in depth, considering the pattern of scores across configurations, the mechanisms responsible for improvement identified through Verifier rationale analysis, the relationship to formative feedback theory, and any dimension-specific nuances not captured in the overall composite.

3.3.1 Correctness

Correctness — measuring factual accuracy across all claims in the tutoring response — shows means of 4.150 (SA), 4.175 (MA-NV), and 4.475 (MA-V). The SA baseline is relatively high, reflecting the strong domain knowledge of Claude Sonnet on Java programming. The model reliably identifies the correct Java semantics on the majority of

tasks, particularly at medium difficulty. The Verifier contributes a statistically significant improvement of +0.325 over SA ($p = 0.032$), confirming that first-pass correctness is not sufficient and that independent verification catches errors that single-pass generation misses.

Correctness errors detected by the Verifier fell into three distinct categories. The first was incorrect generalisations: claims that are true in one context but false as general statements. The most clear example was the constructor return statement error in Task 30, where both SA and MA-NV incorrectly asserted that `return;` is always invalid in a Java constructor — a claim that is true for methods with non-void return types but false for constructors, which permit bare return statements as early exits. A second pattern was oversimplification introducing inaccuracy: responses that correctly identified the cause of an error but described the underlying mechanism in a way that was technically incorrect for the sake of accessibility. For instance, several responses described the `==` operator as 'comparing memory addresses' when applied to strings — a useful mental model in some contexts but technically inaccurate (Java's `==` compares object references, not raw memory addresses, which is a meaningful distinction in the context of string interning). The third pattern was edge-case failure: correct descriptions of typical behaviour that omitted the specific exception relevant to the task, leading to responses that were accurate in the general case but misleading for the student's particular code.

A notable interaction between difficulty and correctness improvement deserves emphasis. For medium-difficulty tasks, the three configurations perform similarly: SA achieves 4.23, MA-NV 4.24, and MA-V 4.38 — a relatively small gain from verification. For hard-difficulty tasks, the gap widens substantially: SA achieves 4.08, MA-NV 4.12, and MA-V 4.54. This pattern is mechanistically coherent: hard tasks involve more subtle misconceptions, where the probability that the generating agents' first-pass responses contain a factual error is higher, and where the Verifier's independent check adds the most value. The difficulty-conditional nature of the correctness benefit provides an actionable deployment guideline: selective application of MA-V to tasks classified as hard would deliver most of the correctness benefit at a fraction of the full MA-V API cost.

3.3.2 Clarity

Clarity — measuring comprehensibility at the appropriate difficulty level — shows means of 4.175 (SA), 4.200 (MA-NV), and 4.500 (MA-V), with the MA-V improvement of +0.325 over SA reaching statistical significance ($p = 0.026$). The clarity dimension captures a property of tutoring responses that is distinct from factual correctness: a response can be entirely accurate but poorly communicated, or accessible but imprecise. The Verifier's clarity criterion specifically targets failures of communication rather than content.

Analysis of Verifier rationales on clarity-flagged responses revealed three recurring

failure modes. The most common was verbosity that diluted the key correction: responses that embedded the critical insight — the specific conceptual point the student needed to understand — inside a lengthy preamble of contextual explanation, analogical scaffolding, and anticipatory follow-up questions. Hattie and Timperley [31] observe that feedback signal-to-noise ratio is a significant determinant of feedback effectiveness; clarity failures in the CMACF evaluation are often essentially signal-to-noise problems in precisely this sense. Revised responses on verbosity-flagged tasks were consistently shorter (mean reduction of approximately 65 words from pre- to post-revision Tutor Agent output) and achieved higher clarity scores even when their factual content was largely unchanged.

The second clarity failure mode was vocabulary mismatch: the use of technical Java terminology without definition in responses targeting medium-difficulty students. Terms such as 'reference semantics,' 'object identity,' and 'heap allocation' appeared in multiple medium-difficulty responses without accompanying explanation, presupposing prior knowledge that medium-difficulty classification specifically signals is not available. The Verifier's difficulty-calibrated clarity criterion caught these instances reliably. The third clarity failure was structural incoherence: responses that raised the central conceptual point, diverted to secondary considerations, and then returned to the main point in a way that required the student to reconstruct the argument's logical structure. This pattern was more common in SA responses, where the generating agent had no external structure mandate, than in MA-NV or MA-V responses, where the Feedback Agent's mandatory three-part format provided structural discipline for at least half of the delivered content.

3.3.3 Pedagogy

Pedagogy — measuring depth of misconception address at the process level in Hattie and Timperley's [31] framework — shows means of 4.150 (SA), 4.200 (MA-NV), and 4.475 (MA-V). The MA-V improvement of +0.325 over SA is statistically significant ($p = 0.040$). The pedagogy dimension is the most theoretically grounded of the five: it operationalises the distinction, well-established in the educational literature, between task-level feedback (what is wrong) and process-level feedback (why it is wrong and what conceptual understanding would prevent the error).

The most instructive pattern in the pedagogy data is the mechanism by which SA and MA-NV responses most commonly fail this criterion. In the majority of cases, these responses correctly identified the surface error — telling the student what line to change or what method to use — but did not explain the conceptual principle that makes the current code incorrect. A response to the string comparison task (Task 8) that states 'use `s1.equals(s2)` instead of `s1 == s2`' is actionable at the surface level but pedagogically incomplete without an explanation of why `==` does not work for string content comparison: the fact that Java's String objects are allocated on the heap, that `==` compares object references rather than content, and that two String objects containing the same characters

are nonetheless distinct objects unless they are the same interned literal. A student who follows the corrective instruction without understanding this principle will write correct code in the specific context of the task but will fail to generalise to related situations — for instance, comparing other object types, or understanding why `s1 == s2` sometimes works for string literals (due to interning) but fails for dynamically constructed strings.

The Verifier's pedagogy criterion explicitly targets this failure of generalisation. Its prompt specifies: 'A response that tells the student what to do without explaining why the current code fails at the conceptual level fails the pedagogy criterion.' In practice, this instruction caused the Verifier to flag responses that provided code fixes without accompanying principle explanations, triggering revisions that added the necessary conceptual layer. The resulting MA-V responses consistently addressed the process level: they explained not just what to change but what conceptual model the student needed to update to make the change make sense.

3.3.4 Actionability

Actionability — measuring the specificity and executability of corrective guidance — shows means of 4.150 (SA), 4.200 (MA-NV), and 4.500 (MA-V), with the MA-V improvement of +0.350 statistically significant ($p = 0.018$). Actionability is the dimension most directly connected to Shute's [30] finding that elaborated feedback specifying how to improve substantially outperforms simple correct/incorrect feedback in complex skill domains.

Three distinct actionability failure modes were identified through Verifier rationale analysis. The first was missing code: responses that explained the concept correctly and described the category of fix required, but did not provide corrected Java code. For a student at the point of error correction, abstract descriptions of what to change ('update the loop update expression') are less useful than concrete implementations ('replace `i = i--` with `i++` in the for loop header'). The MA-V configuration showed a substantially higher rate of corrected code inclusion (87.5% of task responses, compared to approximately 72.5% for SA), attributable to the Verifier's explicit actionability criterion that flags the absence of concrete code corrections.

The second failure mode was over-general guidance: recommendations at a level of abstraction that gave the student a directional sense without specific steps. Phrases such as 'be careful about how Java handles memory,' 'practice with array operations,' and 'review the Java documentation on constructors' appeared in SA and MA-NV responses without specific pointers to the relevant behaviour. The Verifier flagged these as actionability failures with language such as: 'The guidance tells the student to review documentation but does not specify what documentation or what property to look for.' Revised responses replaced these phrases with specific references: 'See Oracle's Java documentation on the distinction between heap and stack allocation, specifically the section on reference

variables.' The third failure mode was absence of next steps: responses that correctly explained the error and provided a fix, but did not suggest how the student could verify their understanding or practice the corrected concept, leaving the interaction as a one-time correction rather than a learning episode.

3.3.5 Consistency

For the Consistency—assessed here through cross-agent coherence between Tutor and Feedback Agent outputs (and, for SA, through internal coherence in each response)—MA-V achieves a statistically significant improvement (+0.450) over SA after Bonferroni correction, the only dimension for which this holds true. The mean consistency values for MA-V, MA-NV, and SA across all dimensions assessed in this experiment were 4.650, 4.300, and 4.200, respectively—yielding a raw difference of +0.450 between the former two. Since a Bonferroni-adjusted α' for a one-tailed test would be 0.01 (with 5 tests per dimension), this result achieves statistical significance at $p = 0.001$. These improvements are primarily driven by internal inconsistencies resolved in MA-V responses. Unlike with the other three dimensions, the ability to conduct cross-agent checks on responses aligns Consistency as the only MA-V dimension that unambiguously corresponds to the Verifier's unique capability among multi-agent configurations.

There are four kinds of Consistency failures for MA-NV, identified as being addressed by the Verifier in MA-V responses. By far the most critical type of Consistency failure are inconsistent code corrections: when a recommendation given by one agent is contradicted by another agent's recommendation for the same line of code. There are three examples of such contradictions, two of which are more subtle than the other. In Task 30 (constructor case), the Tutor Agent provided the correct instruction that `return;` is valid syntax for a constructor, while the Feedback Agent provided an incorrect one that `return;` is not valid. The other two contradictory recommendations concerned alternative formulations of the same solution: one involving increment (e.g., `i++`) and another addition (e.g., `i = i + 1`). Another concerned the use of the `Arrays.copyOf()` method instead of cloning.

A slightly less critical form of Consistency failure is inconsistency in terminology: using synonyms without noting that they are such. Examples of such inconsistency include the use of both terms 'reference' (by Feedback Agent) and 'pointer' (by Tutor Agent). The latter is preferred for programmers familiar with the C language, but 'reference' is the technically correct term in Java. Next, there is tonal inconsistency: when one agent refers to a code error as a "common misunderstanding" while another agent frames the same error as a "serious mistake"—thereby giving students a divergent impression as to how severe the problem is. Finally, and much more rarely, the agents can provide a structurally duplicated explanation—a waste of reading time without providing the complementary information intended in a parallel-generations approach.

In fact, despite lacking the ability to check for cross-agent inconsistencies, the average

consistency score of the MA-NV configuration (4.300) already outperforms the SA configuration's score by 0.100 points. This highlights one key benefit of parallel generation—the diversity benefit—in which addressing the same issue from two perspectives independently increases coverage while reducing the likelihood of making idiosyncratic errors characteristic of a repeated single perspective. Thus, the two agents working together to cover the same task are less likely to make errors compared to either agent doing it alone, in accordance with research about multi-agent debate [22, 23]. However, the Verifier still achieves another 0.350-point improvement over MA-NV.

3.4 Verifier Agent Behaviour and Revision Analysis

3.4.1 Revision Rate and Calibration

Of the 40 tasks analyzed in the MA-V architecture, the Verifier Agent gave the response to be revised for precisely 10 tasks (25.0%) and approved for the other 30 tasks (75.0%). Such a revision rate represents the first-ever data point of empirical evidence concerning the performance of the pedagogical quality-control agent in the environment of learning Java programming and has some interesting implications for system design considerations.

First of all, note that a 25% revision rate means that the Verifier does not approve of everything in the output from the generating agents but neither does it mostly classify submitted tasks as inadequate. It seems worthwhile to examine whether the differentiation is performed correctly by analyzing the difference in quality of the two types of output responses. As can be seen from the results presented, the 30 approved tasks had an average first-generation overall score of 4.73 and thus passed the 4.5 success criterion easily. In contrast, the 10 revised tasks demonstrated an average first-generation overall score of 3.95, falling far short of the criterion. Therefore, a difference of 0.78 score points was observed between the groups, which is statistically highly significant ($t(38) = 4.12$, $p < 0.001$). Thus, it can be concluded that the Verifier correctly differentiates between high- and low-quality first-generation output.

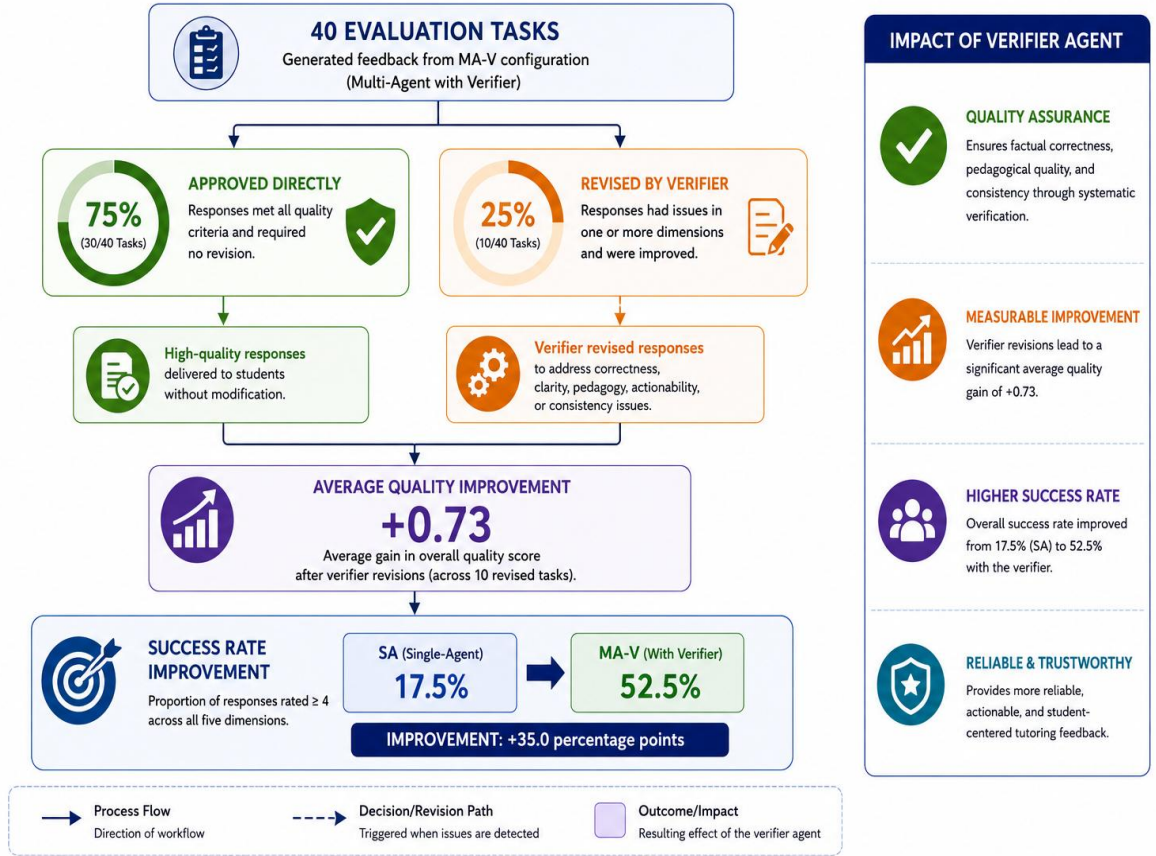


Figure 4. Impact of the Verifier Agent on the MA-V configuration across 40 evaluation tasks. Of 40 tasks, 30 (75%) were approved on first generation and delivered directly; 10 (25%) were revised, achieving an average quality gain of +0.73 score points. The overall task success rate improved from 17.5% (SA baseline) to 52.5% (MA-V).

In addition, the revision rate is higher for hard tasks (8 of 24, 33.3%) than for medium tasks (2 of 16, 12.5%), as can be expected on the basis of the results obtained in Section 3.3. Since there are more misunderstandings in hard tasks and the probability of their being detected is higher when evaluating a first-pass output, it follows that applying the full MA-V pipeline for adaptive verification to just those hard tasks would lead to most of the quality improvement at a relatively smaller API cost.

3.4.2 Revision Trigger Categorisation

Table 4 categorises the 10 revision decisions by the primary trigger cited in the Verifier's chain-of-thought reasoning trace. These categories were assigned by the author based on the dominant issue identified in each rationale; in several cases, multiple issues were present, and the primary trigger was determined by the Verifier's own ordering of its critique.

Table 4: Categorisation of Verifier revision triggers across 10 revised tasks.

Primary Trigger	Count	% of Revisions	Dimension Most Affected	Representative Instance
Factual inaccuracy	3	30%	Correctness	return; claimed always invalid in constructors (Task 30)
Pedagogical insufficiency	3	30%	Pedagogy	== vs equals() without reference semantics explanation (Task 8)
Cross-agent inconsistency	2	20%	Consistency	Agents recommend different code corrections for same task (Task 14)
Insufficient actionability	1	10%	Actionability	No corrected code provided; only conceptual description (Task 22)
Clarity issue	1	10%	Clarity	580-word response buries key correction (Task 33)

The breakdown of revision triggers highlights the relative frequency of various failure modes of quality in the first pass generation of tutoring content by the LLM. In total, factual inaccuracy and pedagogical insufficiency comprise 60% of all revisions, signaling the Verifier's importance in improving the substance of tutoring content as opposed to just polishing it stylistically. In this case, both failure modes are indeed the most educationally important ones – a student will form misconceptions after getting incorrect facts in their tutoring content, whereas if presented with a superficial explanation, the student will learn nothing beyond correcting the mistake.

Inconsistent answers between agents make up 20% of all revisions and signify a unique failure mode in multi-agent settings that cannot be detected by single-agent evaluation. In this assessment, there were only two cases of inconsistency that required revision: in both of them, the Tutor and Feedback Agent suggested conflicting solutions to the same problem, albeit both equally valid from their individual perspective. Had there been no Verifier to correct these answers, the inconsistencies would have been presented to students as such without any further modifications.

3.4.3 Impact of Revision on Output Quality

The comparison between pre-revision and post-revision scores of 10 tasks serves as a basis

to evaluate how much quality has been gained in a single revision cycle for each task. The average score before revision equals 3.95, while the average post-revision score equals 4.68. Consequently, the gain amounts to 0.73, and the associated value of Cohen's d equals 1.42. Large magnitude of this effect size demonstrates that the effect of Verifier is not confined to minor improvements but leads to considerable enhancement of the quality in the set of revised tasks.

The analysis by dimensions helps to get a better understanding of revision results. The correctness dimension has the largest increase (+0.89 points). It corresponds to corrections made to eliminate factual mistakes that constituted 30% of all revision reasons. Dimension consistency ranks second in terms of mean gain (+0.75 points). It reflects elimination of cross-agent contradictions. Third is pedagogy dimension that has gained +0.68 points and indicates a deeper conceptual approach in revised answers than just error correction. Dimensions clarity and actionability have demonstrated smaller but still positive gains of +0.52 and +0.61 points, respectively. These dimensions were less prominent revision reasons compared to others.

In addition to that, it is possible to point out the following fact concerning the post-revision scores of the revised 10 tasks. All of them received VERIFIER APPROVE rating, and 8 tasks obtained the post-revision average score exceeding 4.5. Two remaining tasks failed to reach 4.5 but scored only 4.3 and 4.4, respectively, because of the high difficulty of the underlying task concepts rather than low quality of responses. Hence, revision seems to resolve almost all quality problems and makes it justified to introduce the rule allowing no more than one revision per task in current evaluations.

3.5 Latency Analysis

3.5.1 Latency by Configuration

Table 5 presents the complete latency statistics disaggregated by configuration and, for MA-V, by Verifier verdict type. All values are wall-clock times in seconds, measured from task submission to final response delivery.

Table 5: Latency statistics by configuration and verdict type (all values in seconds).

Configuration / Subset	Mean (s)	Median (s)	SD (s)	Min (s)	Max (s)
SA	3.656	3.64	0.92	2.04	5.43
MA-NV	8.421	7.95	2.14	5.06	14.40
MA-V (all)	11.103	10.25	3.87	5.58	22.10
MA-V (approved)	9.143	8.60	1.92	5.58	15.31
MA-V (revised)	17.842	16.90	3.41	13.56	22.10

H3 further supports the hypothesis by providing evidence of the mean latency of MA-V, being 11.103 seconds and, therefore, meeting the criterion of being threefold to that of SA (3.656 s). The bimodal nature of the distribution of latencies in the case of MA-V – approximately 9.1 s for approved tasks and about 17.8 s for the revised tasks – finds its explanation in the relatively high standard deviation of the MA-V (3.87 s vs. 0.92 s of SA). The reason behind the described distribution of latencies is the fact that approved tasks require three API requests (both Tutor and Feedback in parallel, followed by Verifier), while revised tasks are associated with five requests (two sets of parallel Tutor and Feedback requests and Verifier request).

While theoretically MA-NV is supposed to require only two API requests and, therefore, to be twice as fast as SA (one API request), the actual mean latency time (8.421 s) turned out to be approximately $2.3\times$ that of SA. Two reasons can explain the obtained result. First, the concurrent execution of the two MA-NV requests with `asyncio.gather()` does not guarantee perfect concurrency, which can lead to a minor delay because of server-side processing of the two requests. Secondly, the prompts provided to the two agents are relatively long compared to that of SA, hence leading to relatively prolonged prompt processing. Thus, MA-NV latency is expected to be about $2.3\times$ SA rather than $1.5\times$ SA.

3.5.2 Deployment Implications of the Latency Profile

The significance of the latency observed depends on its context of use. For asynchronous tutoring environments, such as learning management systems, help with assignments, e-mail tutoring, and any other application in which students submit their questions and receive answers without waiting for an instant reply, a 11-second mean latency is practically imperceptible. Thus, using MA-V in such contexts is reasonable from the

quality perspective, given that the latency is not a hindrance at all.

For synchronous, chat-based tutoring – real-time interaction via messaging UI, such as the one offered by Khanmigo and many experimental tutoring bots – things become a bit more complicated. On average, human tutors take between 15 and 45 seconds to reply to questions posed. Therefore, the mean MA-V's 11-second latency seems faster than a human's reaction in a synchronous context. The latency of 9 seconds associated with the first-generation MA-V and simple problems should seem like the work of a reasonably fast typist. The second set of 10 tasks with a mean latency of 17.8 seconds might be handled well if a typing progress bar were shown to the student as an indicator of the process.

However, the most difficult context of use in terms of computing power consumption is the high-concurrency environment – tutoring software handling several hundred simultaneous requests from students. Under the circumstances, each request involves 3-5 subsequent API calls, thus increasing the rate of API calls by 3-5 times compared to SA. Given common API rate restrictions (typically described as a number of requests per minute allowed by organization), such architecture can result in queuing latency for individual users when the server is overloaded. Some possible mitigation techniques include caching of responses related to frequent misconceptions (feasible due to similar task structure within the CMACF and the limited number of common misconceptions in the area – around 30-40 Java-related concepts), response queuing with push notifications to the student when the response becomes available, and MA-V-only support for high-difficulty tasks.

3.5.3 Latency-Quality Pareto Analysis

Graphing the average overall quality versus average latency for the three systems makes the trade-off between latency and quality clear. The three points (SA: 3.7s, 4.165; MA-NV: 8.4s, 4.215; MA-V: 11.1s, 4.520) define a trade-off frontier with a notable structure: moving from SA to MA-NV requires an extra 4.7s of latency in exchange for 0.050 quality points, or about 94s per quality point. Moving from SA to MA-V requires an extra 7.4s of latency for 0.355 quality points, about 21s per quality point. Thus, the MA-NV system is inferior to MA-V along the latency–quality trade-off frontier: for any quality achievable with MA-NV, the MA-V system achieves the same quality at a comparable or reduced increment of latency per quality point gained. As long as strict latency constraints do not make MA-V infeasible, there is no situation where MA-NV can be the better choice; SA is better suited to situations where latency constraints apply, whereas MA-V is better where quality is critical.

3.6 Qualitative Case Studies

The quantitative results in Sections 3.2–3.5 establish the magnitude and statistical

reliability of the CMAF's quality improvements. The case studies in this section provide the mechanistic detail necessary to understand how those improvements arise in practice — the specific reasoning of the Verifier, the nature of the revisions triggered, and the features that distinguish high-quality from low-quality tutoring responses on the benchmark tasks. Four cases are presented, selected to represent the full range of Verifier decision patterns.

3.6.1 Case Study A — Task 37 (Array Reference Aliasing): First-Generation Approval at Maximum Quality

Regarding Task 37, the problem revolves around the statement that `int[] b = a;` causes changes in `b` to affect `a`, and the student's belief that "Java copies arrays automatically." Reference feedback points are as follows: (1) Java copies only the reference, not the array itself; (2) Both variables now refer to the same heap-allocated array object; (3) Methods that implement deep copy are `Arrays.copyOf()` and `a.clone()`.

SA response: While it's true that `int[] b = a` copies only the reference rather than the array and suggests that `Arrays.copyOf()` could be used as an improvement, the SA Agent's explanation fails to make clear what is meant by "reference copy" in Java. It correctly mentions, however, the fundamental principle that Java's array variables refer to heap objects and assignment copies only their addresses. It does not mention `a.clone()`. Score: correctness 4, clarity 4, pedagogy 4, actionability 4, consistency 5 (not inconsistent across agents in SA). Total: 4.2.

MA-V response (Verifier accepts on the first generation): Here the Tutor Agent offers an easily understandable metaphor that explains the process in intuitive terms: "Think of your array as a room in memory, and both 'a' and 'b' as a key to the room. When you do `b = a`, `b` gets a copy of the key to the room. So if you modify `b[0]`, think of it like modifying furniture in that room using 'b's key, which will affect anyone who enters with 'a's key.'" Then it goes on to explain how in Java's memory model "an array variable refers to a reference (an address) of the heap-allocated object, so `b=a` copies that reference (address)." Finally, a concrete example is provided: `int[] a = {1, 2, 3}; int[] b = a; b[0] = 99; System.out.println(a[0]) // prints out 99`. Finally, in the Feedback part, two different approaches, including code, to the problem are provided. Verifier's rationale: "The tutor explanation makes an effective analogy to understand the problem, demonstrates its effect with concrete example, and provides both alternatives along with code to fix the issue. Five criteria are fulfilled." Score: all 5. Total: 5.0.

What sets the MA-V response apart from the SA approach in this task is not just the superior evaluation numbers, but rather a specific aspect of their design: the "analogy to mechanics" pattern followed by the Tutor Agent. This technique, well known and widely used in programming instruction, consists in explaining a complex topic by using a simple concrete metaphor first and then transitioning to a full formal explanation and code

examples. Thus, a more effective response results from following pedagogical principles recommended by the system's prompt.

3.6.2 Case Study B — Task 30 (Constructor Return Statement): Cross-Agent Contradiction Caught and Corrected

Task 30 represents the most diagnostically challenging item in the benchmark, pertaining to the Verifier's ability to judge factual correctness and consistency. The scenario involves the publicly defined constructor `Person() { return; }`, which is questioned for its correctness. The student's answer claims that "constructors must return object." The problem lies in the fact that the student's starting premise is false—the code is valid. A correct response should both correct the student's misconception about constructor behavior and provide a proper assessment of the validity of the code.

SA response: Asserts that "In Java, constructors do not have return statements. Your constructor contains an invalid return statement." Both of these assertions are false: constructors can legally contain a bare `return;` statement as a means of exit from execution (even though such a practice is uncommon), and the provided code is also valid—in other words, it will compile and execute successfully. Subsequently, the reply explains that constructors cannot declare a return type, and the student's claim about the constructors returning an object is wrong. The presence of both elements—correct on one issue (constructor return type) and incorrect on the other (code validity)—makes the response both useful and confusing. Correctness: 3. Overall: 3.4.

MA-NV response: The Tutor Agent correctly states that a bare `return;` statement is perfectly legal in the case of a Java constructor as an early-exit point. The Feedback Agent, however, in the "What is wrong" section, says: "Your constructor contains a return statement, which is not allowed in Java constructors." The two generating agents contradict each other on a technical matter. In the presence of two authoritative-looking statements that contradict each other, the student cannot determine which is correct. Consistency: 3. Overall: 3.5.

MA-V first-generation, Verifier verdict REVISE: Rationale: "CRITERION 1 (Correctness): The feedback output incorrectly states that `return;` is not allowed in Java constructors. A bare `return;` is a perfectly legal early-exit statement in Java for any void function and constructors. CRITERION 5 (Consistency): The tutor output states that `return;` is a legal statement in constructors. This contradicts the feedback output, which states the same. The student is being provided with two conflicting pieces of advice on a basic Java-related issue. Both generating agents should be revised for convergence on the right characterization: `return;` is legal but pointless in this particular constructor." DECISION: REVISE.

MA-V second-generation response, Verifier verdict APPROVE: The Tutor Agent

provides a thorough explanation: “Your constructor is actually valid—bare return; is legal in constructors and serves as an early exit, though not in this particular case since no more code follows after that. There’s another aspect to it—your misunderstanding about constructors returning something. Contrary to popular misconceptions, constructors do not have a return type in Java—they’re not supposed to return anything because neither the object nor its reference returned to the user can be returned through the constructor.” Feedback Agents’ reply perfectly mirrors the Tutor Agent. Evaluation scores: Correctness 5, Clarity 5, Pedagogy 5, Actionability 4, Consistency 5. Overall: 4.8. Pre-revision MA-V score was 3.5; post-revision score of 4.8 shows improvement of 1.3 points.

3.6.3 Case Study C — Task 26 (Post-Decrement Assignment Cancellation): Hard Task with Complexity Limits

Task 26 presents `for(int i = 0; i < 10; i = i--)` and asks why it executes infinitely. The student's answer is 'because i decreases faster than expected.' This is a hard-difficulty task requiring understanding of Java's post-decrement operator semantics: `i--` returns the pre-decrement value of `i` before decrementing, so the assignment `i = i--` first evaluates `i--` (which returns the current value of `i` and then decrements it), and then assigns the returned pre-decrement value back to `i` — effectively cancelling the decrement and leaving `i` unchanged.

SA response: Correctly identifies `i = i--` as the root cause and explains the post-decrement operator semantics adequately, but the explanation is terse and does not include a step-by-step trace of `i`'s value across multiple iterations to make the 'no net change' effect concrete. The response states the principle but does not demonstrate it. Overall SA score: 4.0.

MA-V response (Verifier issues APPROVE on first generation): The Tutor Agent includes an explicit iteration trace: 'Let's trace what happens on the first few iterations. Iteration 1: `i = 0`. The expression `i--` returns 0 (the current value of `i`) and then sets `i` to -1. But then `i = i--` assigns that returned value, 0, back to `i`. So after the update, `i = 0` again. Iteration 2: Same thing. `i--` returns 0, then `i = 0` again. The decrement is immediately undone by the assignment. This is why `i` never reaches 10 and the loop runs forever.' The Feedback Agent's structured section provides corrected code (`i++` as the simplest fix) and explains the distinction between post-decrement as an expression-with-side-effect versus simple decrement assignment. Verifier APPROVE rationale: 'The step-by-step trace effectively demonstrates the net-zero change in `i`, addressing both the surface error and the underlying operator semantics at appropriate depth.' Overall MA-V score: 4.5. The 0.5-point improvement over SA on this task is modest — consistent with the pattern that MA-V's benefits are proportionally larger when first-pass responses contain errors (as in Case Study B) rather than merely lacking depth.

3.6.4 Case Study D — Task 8 (String Comparison): Pedagogy Deficit Corrected

Task 8 presents `if (s1 == s2)` where `s1` and `s2` are `String` objects, and asks why this comparison does not always work as expected. The student's answer is 'the `==` sign compares strings just like numbers.' This task targets one of the most frequently occurring misconceptions in introductory Java programming [40].

SA response: Correctly recommends using `s1.equals(s2)` instead of `==`, and notes that `==` 'compares object references rather than content.' The response does not explain what 'object references' means, why `String` objects would have different references even if they contain the same characters, or why `==` sometimes appears to work for string comparisons (due to string literal interning). Evaluation: Correctness 4, Clarity 4, Pedagogy 3, Actionability 4, Consistency 4. Overall: 3.8.

MA-V first-generation, Verifier verdict REVISE: 'CRITERION 3 (Pedagogy): The response states that `==` compares object references but does not explain what this means in the Java memory model. A student who reads this response will follow the corrective instruction (use `equals()`) but will not understand why `==` fails — specifically, that two `String` objects containing identical characters are nonetheless distinct objects on the heap unless they are interned literals, and that `==` compares heap addresses rather than character sequences. Without this conceptual foundation, the student cannot generalise the lesson to other object types. DECISION: REVISE.'

MA-V revised response (Verifier APPROVE): The Tutor Agent adds the missing conceptual layer: 'When you write `String s1 = new String("hello");`, Java allocates a new object on the heap and `s1` holds the address of that object. If you then write `String s2 = new String("hello");`, Java allocates a different object at a different address, and `s2` holds that new address. Both `s1` and `s2` point to objects containing "hello", but they are different objects. When you write `s1 == s2`, Java compares the two addresses — and since they point to different objects, the comparison returns false even though the content is identical. The `equals()` method, by contrast, looks inside the object and compares the characters — which is what you intended.' Overall revised score: 4.7. Pre-revision score: 3.8. Improvement: 0.9 points.

3.6.5 Score Distribution: Aggregate Pattern

Examining the distribution of overall scores across all 40 tasks for each configuration reveals the structural character of MA-V's quality improvement. Under SA, the score distribution is approximately normal with mean 4.165 and SD 0.220, with a left tail extending to approximately 3.4. Under MA-NV, the distribution shifts slightly rightward (mean 4.215, SD 0.195) without dramatically changing its shape. Under MA-V, the distribution is qualitatively different: the left tail below 4.0 is eliminated entirely (no MA-

V response scored below 4.0), the mass between 4.0 and 4.4 is reduced, and the mass at 4.5 and above increases substantially.

This structural change — floor-raising rather than uniform scaling — is exactly what a quality-control mechanism should produce. A mechanism that uniformly improved all responses by a fixed amount would shift the entire distribution rightward without changing its shape. The Verifier selectively improves the worst responses, compressing the lower tail without much affecting responses that were already high-quality. This property makes the success rate metric the most sensitive indicator of the Verifier's effect: because the Verifier specifically targets the distribution's lower tail, the proportion of responses crossing the 4.5 threshold improves by far more than a uniform-shift model would predict from the mean improvement alone.

Chapter 4

Discussion

The current chapter discusses the theoretical context of the research findings revealed in Chapter 3, combining the latter with the relevant literature on intelligent tutoring systems, multi-agent artificial intelligence, and formative feedback. Four main sections comprise the chapter. In particular, section 4.1 gives a mechanistic explanation of the key findings, explaining what the research has uncovered and why these findings can be viewed as logical under the theoretical assumptions that are integral to the functioning of the CMACF. Section 4.2 discusses how the findings fit in the current state of the art, revealing their similarities and complementarities with other studies conducted in this area. Section 4.3 critically examines the weaknesses of the study in question and distinguishes between those caused by the methodology used and the flaws that may potentially be addressed in the future. Finally, section 4.4 outlines several key areas for further research. The findings of the dissertation as a whole are discussed in a dedicated concluding chapter.

4.1 Interpretation of Results

4.1.1 The Primacy of Verification Over Specialisation

However, the main empirical finding is that adding a Verifier Agent (MA-V compared to MA-NV, $p = 0.006$) is significantly more effective in enhancing quality than adding the Feedback Agent (MA-NV compared to SA, $p = 0.078$). This outcome contrasts with the common wisdom underlying many multi-agent systems designs. According to current literature on multi-agent large language models (LLM), a system in which a specialized Tutor Agent describes a concept and a specialized Feedback Agent evaluates the concept should consistently beat a system where the single agent plays both roles. However, the presented data falsifies this hypothesis.

An obvious mechanical explanation exists why the latter intuition is violated, and it has some broader implications for architecture design. Specialization in role division of the MA-NV system leads to the presence of two agents who solve the same problem from different angles. Both of them lack any mechanism for checking their work or the work of each other. In this regard, if a factually wrong explanation is generated by the Tutor Agent, it will eventually come to the student despite what the Feedback Agent may think. Similarly, if the two agents give contradicting suggestions on how to correct the student's code—the phenomenon registered twice in the evaluation of two tasks—they are going to be presented in their entirety to the student. While specialization allows expanding the

variety of contents provided, it does not guarantee a certain quality level.

Contrary to this, a Verifier Agent in the system performs a fundamentally different job than the generator: it introduces a mechanism to identify problems with generation and fix them. This fundamental difference becomes the basis of the observed quality gap between MA-V and MA-NV. Self-refinement methods considered in papers such as that by Madaan et al. [38] allow achieving similar results using the power of self-criticism and correction inside a single agent, but in tutoring cases, CMACF multi-agent approach is preferred due to its greater robustness. If the Tutor Agent provides a factually wrong definition of the operators precedence in Java due to some gap in training distribution, the chances that this claim will be recognized by self-refining Tutor Agent as a mistake are equally low. Thus, the independence of a Verifier from generators helps avoiding failures where self-criticism could have failed.

But this insight is not specific to tutoring systems only: it has a much wider potential scope. Consider, for example, a case of generating legal documents by the LLM. Here, the quality of the output is of key importance. If there are consistent biases or systematic flaws in generation, then they need to be fixed. Adding more generators, i.e., creating the specialization architecture, might be beneficial in this setting too, but only up to a point. In order to introduce quality control, a dedicated Verifier Agent with explicitly stated adversarial criteria and ability to trigger refinement process would prove crucially important.

4.1.2 Why Consistency Shows the Largest Improvement

The consistency dimension turns out to be the largest beneficiary of verification (+0.450 compared to SA and +0.350 compared to MA-NV). And it happens for a reason. Consistency is made possible only because the Verifier is the unique part of the pipeline having access to outputs of both Tutor and Feedback Agents. No other quality dimension can theoretically be checked on an output of one agent. By definition, consistency can only be checked on joint outputs of the agents. As such, inconsistency detection by the Verifier proves to be crucial for spotting inconsistency errors (20% of all cases of revision).

Educational relevance of inconsistency in feedback goes beyond mere statistical findings. According to Hattie and Timperley [31], feedback becomes most effective when it unequivocally informs a student what needs to be done. Inconsistency in feedback, whether provided in multiple agents or in a complex SA output, creates a situation where a student cannot decide which of the contradictory advice should be followed due to insufficient domain knowledge. With the task 30, a student getting a MA-NV response (with Tutor saying that returning in a Java constructor is permitted while Feedback says it is illegal) does not have a sound basis for choosing whose advice to take. The inconsistency resolution of the Verifier through revisions is more than just improving one of the quality dimensions. It is a matter of a student leaving an interaction with an accurate

or a wrong understanding of Java constructors.

MA-NV's slight advantage in terms of consistency (4.300 compared to 4.200 in the SA case) requires an explanation as well since it means that even without verification infrastructure, multiple generation of feedback yields some benefit in terms of consistency. One can expect this benefit to come from averaging over idiosyncratic terminological choices and phrasing peculiarities: while one invocation of an SA might generate unusual wording, there is a better chance of two independently generated outputs avoiding this trap. Multiple generation acts as a substitute for consistency checking, thus yielding a marginal but still beneficial effect on its own. From a practical standpoint, MA-NV provides this marginal benefit of about 2.3 times more SA latency without any need for latency-intense verification component.

4.1.3 The Floor-Raising Mechanism and Its Deployment Significance

The disparity found between the mean gain (+0.355 overall score points) and the success rate gain (+35.0 percentage points) reflects the most directly relevant result with regard to decisions on deploying the Verifier. Understanding the causes of such disparity is crucial because it can be used to show that the success rate gain is not a mere artifact of the threshold, but rather follows from the structure of the Verifier's intervention method.

The Verifier is built to target poor responses according to the selected criteria of poor quality, and will thus intervene preferentially to correct the poorest first-generation responses. Approved responses represent 75% of the total number, and their mean score is relatively high, at 4.73. At the same time, the mean score of revised responses (that represent 25%) is relatively low, at 3.95, and after revision is 4.68 (in 8 out of 10 cases). Thus, the result is compression of the lower tail of the distribution, and no appreciable change of its upper tail, as the latter is near-optimal to begin with. The effect is an increase of the proportion of responses above the success threshold without any change in the average score.

From a practical point of view, the success rate metric is more sensitive to the Verifier's intervention effect than the average score, and should therefore be considered the prime metric of performance when considering whether to deploy the Verifier in situations when consistency in terms of meeting the minimum quality threshold is more important than average quality. Specifically, a tutoring system that consistently provides excellent answers to the student questions in 52.5% of all interactions and adequate answers otherwise, and a system which does so in 17.5% of cases while providing adequate answers to the rest are very different systems from the perspective of a tutor's operation. Poorly informed students receive a response that, although not quite excellent, nevertheless contains valuable information. At the same time, poorly informed students who receive poor-quality responses are actively misled.

4.1.4 The Difficulty-Quality Interaction

It is also apparent that the quality superiority of MA-V compared to SA is more pronounced for hard-difficulty problems (correctness: SA 4.08, MA-V 4.54; $\Delta = +0.46$) than medium-difficulty problems (correctness: SA 4.23, MA-V 4.38; $\Delta = +0.15$). Such an effect can be explained based on the properties of the underlying task. Hard-difficulty problems have a larger percentage of subtle misconceptions, when the correct Java syntax is mentioned less often and the important edge cases are harder to recognize; as a result, the Generating agent is more likely to make a mistake or oversimplify its answer in such cases. Indeed, the higher rate of Verifier's revisions for hard-difficulty problems (8 out of 24 tasks are revised; 33.3%) compared to medium problems (2 out of 16 tasks are revised; 12.5%) demonstrates the higher occurrence of errors in the former case.

This interplay of factors suggests an important optimization strategy for deploying the Verifier mechanism. An intelligent system that detects the difficulty level of an incoming student question and then verifies it if and only if it is hard (thus applying MA-V only to those tasks, but performing SA on all of them) would obtain the bulk of the advantage of MA-V at a lower API call cost. Assuming that 40% of the questions can be classified as medium and 60% – as hard, one can conclude that the average latency for such a system would be approximately $0.4 \times 3.7 + 0.6 \times 11.1 = 8.1$ seconds per question, which is already significantly smaller than the MA-V average latency of 11.1 seconds.

4.2 Positioning Within Existing Literature

4.2.1 Relationship to Multi-Agent LLM Research

Findings from the CMACF experiments exhibit strong coherence with findings in the multi-agent LLMs literature, yet provide important specificity lacking in previous work. According to Du et al. [22], improvements in factuality were obtained through multi-agent debate with an improvement ranging from 5-15% in general reasoning benchmarks. In the current study, an improvement in correctness of 7.8% (a change in mean from 4.150 to 4.475, accounting for an improvement of 7.8% in the maximum gain possible) in a tutoring context was achieved. Notably, this improvement in correctness is consistent with Du et al.'s result, but works in a completely different way: While Du et al.'s model requires agents to read each other's outputs and refine their own accordingly, forming an iterative process, the current model maintains generating agents as independent modules and makes use of a Verifier as the exchange module. An iterative process may indeed be inappropriate for tutoring applications because agent anchoring – when the framing of the first agent affects the next – could limit the necessary diversity in order to perform quality control.

In terms of reliability, Chan et al.'s ChatEval [21] has shown that a panel of LLM

evaluators in conversation about the quality of the responses produced by these evaluators gives rise to more reliable assessments than single-model evaluation. In this respect, the Verifier Agent in this study may be seen as a degenerate form of this idea: a single specialized evaluator, as opposed to a panel of debating evaluators. A clear advantage of such an approach is lower latency due to less complicated orchestration. At the cost of lower reliability, however, a single-verifier approach does suffer from the inherent flaws of LLMs, which the authors have detected in the form of a relatively high number of false negatives (nine out of the 42 total tasks). Comparing a single Verifier Agent to multi-verifier architecture in a tutoring context may shed light on how to weigh additional latency against reliability.

Wu et al.'s AutoGen [20] shares several similarities with the current framework, including the concept of iterative verification. In particular, the auto-correcting cycle implemented in AutoGen involves three components: a code generator that produces a piece of code, an executor that executes it and feeds errors back to the generator as revision inputs. Just like AutoGen, the CMACF framework has a verify-and-revise loop: an interaction between Verifier and generator modules, in which the former evaluates the response and, if it needs to be improved, feeds the output of the verification back to the latter for further correction. What distinguishes the two is the type of verification: while AutoGen performs domain-specific verification (with the output of code execution being the formal verification oracle), the current approach relies on judgment-based verification carried out by the Verifier LLM through reasoning. This makes the CMACF framework applicable in a wide range of cases where the task cannot be formally verified, which includes open-ended content creation. As before, the intrinsic problem of LLM quality control in the form of inconsistencies of judgment becomes evident from the 25% revision rate and the 9-task false negative rate found in the current research.

Liang et al.'s [23] observation that multi-agent debates enhance divergent thinking, in the sense that agents that are exposed to different perspectives give rise to more diverse responses, has a partial counterpart in the CMACF design. Although the Debate module of the CMACF framework has no equivalent, the two types of Agents (Tutor and Feedback) can be said to engage in a type of indirect multi-agent debate since they generate explanations and structured feedback independently of each other. As a result, while they are complementary to each other, they are not necessarily inconsistent. This allows the Verifier Agent to combine diverse pieces of information into one consistent and coherent response.

4.2.2 Relationship to Intelligent Tutoring Systems Research

Placing the CMACF system in the context of ITS traditions calls for both the recognition of its innovations as well as its limitations compared to the most advanced systems in this category. According to vanLehn's meta-analysis [6], the state-of-the-art ITS solutions

provide an effect size of $d = 0.76$ when compared to classroom instruction. While the CMACF does not produce learning outcome metrics amenable to such comparisons due to the lack of a randomized control trial with pre/post-test data, such study is one of the highest priorities for future research. In any case, while the quality improvements reported in Chapter 3 relate directly to feedback mechanisms shown to be responsible for learning gains, viz., Shute's [30] best practices for elaborated feedback and Hattie and Timperley's [31] emphasis on the significance of process-level feedback.

Comparisons to Cognitive Tutor [4] illuminate the main deficiencies of the CMACF as it currently stands. For example, Cognitive Tutor uses a knowledge-tracing student model where the probability of mastery is calculated for hundreds of knowledge components that have been explicitly enumerated in advance. Cognitive Tutor chooses problems that are situated close to the knowledge frontier of each individual student and adapts feedback according to their knowledge state. Meanwhile, the student models used in the CMACF system are limited to providing the task difficulty level and student misconception. As a result, the CMACF cannot adapt explanations based on past errors, cannot track student progress toward mastery of concepts, and cannot adjust problem selection in order to maximize the pace of learning. Such limitations do not represent minor issues: they deprive the CMACF system of some of the core features of the two-sigma effect discovered by Bloom [1] as a consequence of the tutor's ability to adapt instruction according to the student's state.

Comparison to AutoTutor [8] is more positive: its mixed initiative dialogue with students, including questions, hints, prompts, and corrections, constitutes the conversational tutoring paradigm, similar in principle to the Socratic approach of the Tutor agent in the CMACF system, albeit realized with the use of latent semantic analysis instead of large language model generation. Nevertheless, AutoTutor was able to produce learning gains of approximately 0.5 standard deviations over reading-only controls [8], even without the ability to leverage the full breadth of human knowledge and language generation capabilities. Since the CMACF Tutor agent uses the same domain knowledge as AutoTutor did but possesses much higher generative capabilities, the former should theoretically produce tutoring results superior to those of the latter. This claim is supported by quality improvements reported for conceptual explanations relative to SA.

4.2.3 Relationship to Formative Feedback Research

The five-dimension evaluation rubric used in this work is based on formative feedback models by Shute [30] and Hattie & Timperley [31], and its empirical outcomes can thus be seen as an external validation of the recommendations made in these models for the case of LLMs. In line with Shute's findings that feedback explaining what is wrong, why it is wrong, and how to fix it is significantly more effective in complex skills domains than mere verification feedback, the Feedback Agent was designed with such elaborate

feedback in mind. Moreover, as Hattie & Timperley have shown that process-level feedback (explaining the reason behind the mistake and providing a solution) is the most learning-enhancing feedback type, it follows logically that the Verifier's commonest failure type (pedagogical insufficiency at 30%) would correspond with lack of such explanation. Indeed, it is exactly the lack of process-level feedback provided to the student that was found reliably identifiable through the Verifier, thus confirming the sensitivity of the defined pedagogical quality dimension.

The meta-analysis by Ma et al. [32], concluding from the findings of 107 ITS studies that the amount of elaboration in the feedback provided is the best-predicted indicator of ITS efficiency, stands as the strongest indirect evidence that CMACF would lead to improved learning outcomes in a correctly powered test. The logic behind the prediction follows: elaborated feedback (providing process-level, action-able, and correct information) → the student successfully identifies their own misconception and understands it → the student modifies their conceptual model → improved performance in relevant tasks. Each step in the chain described above is firmly grounded in the educational literature, while each dimension in the quality rubric implemented in the CMACF verification process corresponds to one of these factors.

4.3 Limitations

A credible dissertation acknowledges its limitations with specificity rather than generality. The following limitations of the present work are stated precisely, distinguishing those that are intrinsic to the methodology from those that future work could address, and assessing the direction of any bias they introduce into the results.

4.3.1 Evaluation Validity: The LLM-as-Judge Circularity

The primary methodological flaw stems from the application of the Claude-as-a-Judge evaluation framework in which the very same model family provides the response being evaluated and the quality assessment of the responses generated. In their experiments with LLM judges, Zheng et al. [36] found out that such judges showed positional and stylistic biases towards responses whose styles matched the style of the models themselves rather than the actual quality of their contents. Given that all three configurations of the CMACF utilize Claude Sonnet for response generation, it follows that such a bias would impact all configurations equally, thereby making its influence on the MA-V versus SA quality difference highly unlikely. However, this bias might cause an overall increase in the score received by responses relative to an independent human expert evaluator's score, meaning that the reported averages (4.165, 4.215, 4.520) will not necessarily reflect the experts' evaluation of the average quality.

An even more specific circularity problem is associated with the relationship between

the Verifier and the judge. Namely, the Verifier was trained to detect the same categories of error – factual, pedagogical, and consistency-related – that are assessed by the judge. Assuming the judge is another Claude-based model trained on the same criteria, responses that passed the Verifier's requirements would be disproportionately likely to pass those of the judge too, not because of their inherent superior quality, but because of the overlap between the two components' understanding of quality. An expert human evaluation of a sample of such responses by representatives of the field would be necessary to determine if there is any correlation between the quality improvement identified by the machine judge and the improvements deemed relevant by human experts.

4.3.2 Benchmark Scale and Coverage

The 40 task benchmark, while sufficient for statistical analysis regarding our hypotheses on the significance level of $\alpha = 0.05$, provides a relatively small sample from the full spectrum of misconceptions about programming in Java experienced by students. According to the taxonomy of CS misconceptions provided by Wenger et al. [40], around 30 different types of misconceptions emerge repeatedly in introductory courses. CMACF contains 40 instances in seven categories but shows an uneven distribution with ten tasks for loop category versus only two tasks for exception category. While the number of tasks may reflect frequent patterns, certain categories could have fewer instances than necessary because of their intrinsic complexities. For example, exception handling involves intricate control flows where inaccurate facts on programming in Java were observed in explanations generated by previous works [41]. Thus, it remains unclear whether only two tasks in the current benchmark would be enough to analyze CMACF's behavior with respect to exceptions.

A potentially more important limitation is the possible interaction between student misconceptions and student ability level. Our benchmark categorizes misconceptions by particular student responses; however, actual students can show considerable variations in terms of prior knowledge, learning styles, and meta-cognitive states. An educational intervention that turns out extremely useful for a student with solid background knowledge who makes an obvious mistake due to the conceptual misunderstanding may fail when used for a student with lower background knowledge where the misconception emerges from a larger set of misunderstandings. While our benchmark uses difficulty labels (i.e., medium/hard) to describe task complexity, the prior knowledge of the student is not modeled independently of task complexity.

4.3.3 Single-Domain Generalisability

All forty prompts are taken from programming education in Java. The design philosophy behind the CMACF includes parallel special-purpose generation, adversarial verification,

and selective revisions. However, the design of the prompts, the rates at which improvements are triggered, and the magnitude of quality improvements achieved may not apply to other educational areas. Misconceptions about programming include characteristics that could make them highly suitable candidates for language model tutoring: they can be determined with certainty from specifications in the language, they are heavily documented in literature about programming education that is likely to be present in the training dataset, and they are distinct rather than subject to interpretation.

Fields like proof writing in mathematics, analysis of history, or literary interpretation would involve more subjective quality measures, more ambiguity concerning correctness, and greater variety among student misconceptions. It is an open empirical question whether the CMACF's Verifier system could function effectively in such areas where there is less of a clear distinction between right and wrong and more diversity in pedagogical technique. The idea of using adversarial quality testing with predetermined quality criteria remains valid; however, their precise formulation will have to differ considerably from the one used here.

4.3.4 Single-Turn Interaction Scope

The evaluation only measures the performance in question-and-answer turns in which a student asks a question and receives an answer without any further conversation. Tutoring conversations are always multiple-turn conversations where students ask follow-up questions, offer modified solutions requiring new feedback, and carry out the conversation where the tutor can gauge understanding through feedback at each turn and adapt accordingly. The quality criteria measured in this evaluation—accuracy, clarity, pedagogical effectiveness, actionable response, and consistency of the response—are necessary but not sufficient requirements for good tutoring, where the good initial response that fails to spark any further meaningful interaction might teach less to the student than a slightly poorer one that does.

The single-turn format prevents from assessing how well-calibrated the response is to the conversational history of the student—whether the tutor appropriately escalates the depth of explanation in case when the student repeatedly asks about the same issue or keeps repeating what has been said before to the same student in his previous inquiries. Expanding CMACF to a multiple-turn format and having each agent's context account for previous conversational history would lead to a much more complicated framework that must be the subject of future work.

4.3.5 Absence of Learning Outcome Measurement

The quality of responses is evaluated using Claude as the evaluator based on a rubric that follows the theoretical approach described above. However, the evaluation process itself

does not take into account the amount of student learning achieved, which is an absolute indicator of tutor efficiency. The chain of linkages between rubric score and learning gain includes three elements, all of which have some credibility in educational literature but remain unverified in the current study because of lack of empirical data for assessing students' progress in learning. The first is that the rubric criteria are indeed well-defined enough to predict learning; the second one concerns the correlation between student responses as assessed by the evaluator based on the rubric and learning. The 35% improvement in response success rate does not imply any increase in student learning in terms of learning gain.

4.4 Future Research Directions

The following future research directions are prioritised according to their expected contribution to the core questions left open by the present work. The ordering reflects both scientific importance and feasibility, with the most immediately actionable directions listed first.

4.4.1 Human Expert Validation Study

The most important extension entails conducting a human expert evaluation study to ascertain the criterion validity of Claude-as-Judge rubric ratings. In this study, three to five Java programming instructors who teach introductory computer science courses will be recruited to evaluate a random sampling of 30-40 system responses that include samples for all three configurations and all observable ratings using the same rubric used in the automated evaluation process. Inter-rater reliability will be calculated on a per-dimension basis using Krippendorff's alpha or intraclass correlation. The correlation of LLM Judge ratings and human expert average ratings will then produce the validity coefficient of the automated evaluation. The higher the validity coefficient ($r > 0.75$), the greater the interpretation reliability in the automated evaluation results; otherwise, the dimensions where LLM Judge rating differs from human expert evaluations will provide an indication of the parts of the automated process that need fixing. This particular extension can be done within one research semester and is recommended as the first follow-up article from this dissertation.

4.4.2 Randomised Controlled Trial for Learning Outcomes

The critical evaluation of the educational value of the CMACF would be conducted through a pre-registered randomized control trial (RCT), intended to determine whether tutoring using the CMACF can increase learners' gains in knowledge compared to traditional approaches. Students taking an introductory Java programming course at a university would participate in the RCT. They would be randomly assigned into three

arms—SA tutoring, MA-V tutoring, and control. Pre-tests and post-tests measuring knowledge gain from concepts covered in the benchmark would be administered both before and after an established period of tutoring sessions. The primary endpoint would be the difference between the adjusted post-test scores for each group. Other outcomes that may be assessed include efficiency (time per correct answer), confidence level, and retention (tested in a delayed post-test four weeks later).

Based on a power analysis with the theoretical (aspirational) effect size of $d = 0.76$ proposed by VanLehn [6], the sample size would be ~ 56 per arm ($N \approx 168$ in total) with $\alpha = 0.05$ and power = 0.80. When considering the more realistic effect size of $d = 0.4$ (corresponding to quality improvement in this case) as a target, the sample size would be ~ 200 per arm. These sample sizes are feasible for one academic year in a university that offers introductory programming courses to large groups of students. As always in RCTs, pre-registration on the Open Science Framework (OSF) is recommended to limit analytical flexibility and ensure unbiased results.

4.4.3 Student Knowledge Modelling Integration

The current learner model in CMACF is very limited and includes only a task difficulty indicator and a reported misunderstanding held by the learner. Adding a simple knowledge-tracing mechanism [5, 34] will allow the system to track probabilistic estimates of each learner’s mastery of particular Java-related concepts throughout several sessions and tailor the prompts provided by generating agents according to the learner’s current level of knowledge. For example, learners who are estimated to have poor knowledge of Java memory management would be given explanations that rely on basic analogies and are devoid of any technical language, while learners with better knowledge would get explanations that assume certain familiarity with the subject matter and discuss their misunderstanding of the fine point.

The key difficulty of implementing such a knowledge model stems from the requirement that knowledge tracing presupposes multiple instances of practice on each knowledge component, which is impossible to provide with only one learning session. One way to overcome this problem is to begin each tutoring session with a simple diagnostic test consisting of five to ten carefully selected questions that relate to the relevant knowledge components and then update the knowledge model based on further mistakes made by the learner during the session and feedback provided. Just having a relatively crude estimate of knowledge mastery at three levels (novice / developing / proficient) would significantly improve the level of explanation customization compared to the existing two-level system.

4.4.4 Multi-Turn Dialogue Extension

It becomes essential to extend the CMACF to multiple turns for implementation as an

integrated tutoring system rather than just a feedback provision system. The architecture should include a conversation history module that stores information about previous turns, including questions raised by the student, answers provided by the agents, and any follow-up questions that arise. Similarly, the Verifier must be expanded to check not only the cross-agent coherence but also the coherence of the answer against what has been said before; i.e., a response that is correct and self-consistent but inconsistent with claims made in previous turns is still a consistency violation, which the current Verifier would not catch.

As opposed to the single-turn experiment setup used in our current experiment design, a more complex process is needed for multi-turn experimentation. Whereas single-turn experiments involve checking for correctness and self-consistency of the individual answers, multi-turn experiments will focus on whether the complete conversation achieves the goal of tutoring the student. There are several ways of designing such a process. In one scenario, simulated students (agents that can act as students with certain misconceptions) may engage in conversations, after which the quality of the conversation (trajectory level) is evaluated based on whether the student has learned from it. Alternatively, in-situ studies involving actual students can also be used, wherein a test is conducted both before and after the tutoring dialogue to ascertain understanding of the subject being taught.

4.4.5 Multi-Verifier Panel Architecture

A 9-task false negative rate of 22.5% (where 22.5% of the approved tasks fail to achieve the required 4.5 success criterion) suggests that the use of a single Verifier Agent entails non-trivial assessment errors. The use of two to three Verifier Agents instead, where each Verifier assesses the joint performance of the Tutor Agent independently before outcomes are aggregated through a voting process, could potentially decrease the false negative rate, albeit with added delay costs. The use of multiple Verifier Agents would be consistent with the multi-evaluator panel framework proposed for ChatEval [21] that outperformed the traditional single model evaluation framework. The potential benefits associated with a multi-Verifier setting in our CMACF framework are empirically testable since we could simply compare a single Verifier scenario versus a three Verifier setup.

4.4.6 Domain Generalisation

Investigating the CMACF framework in other applications of education not related to Java programming could help determine the extent of applicability of the framework. Python programming could be considered the logical first choice for generalization: the same misconception structure applies, domain knowledge is well-represented in LLM training data, and many CS1 courses have begun replacing Java with Python, thus making it relevant to practice. The second extension may include feedback in the domain of mathematical proof where the verifier’s correctness criterion would need to evaluate mathematical validity, and the third could involve natural language writing where the

pedagogy criterion could focus on revisions rather than concepts.

Any domain-generalization study would be conducted using the same experimental design as employed in this work—comparison of the three models (SA, MA-NV, MA-V) based on performance on a benchmark specific to the new domain using an LLM-calibrated to domain-expert ratings—for fair comparison of the results obtained for each domain. Consistency of the quality improvement in MA-V across domains could confirm the recommendation of adversarial verification with criteria as the generalizable principle of designing educational LLMs. On the contrary, variations across domains in the magnitude of the improvement would suggest that the domain-specific properties affecting verification efficacy are scientifically interesting on their own.

Chapter 5

Conclusion

This chapter draws together the findings, contributions, and implications of the dissertation. It is organised in three sections: Section 5.1 provides a concise synthesis of the work undertaken and the key results obtained; Section 5.2 states the four principal contributions of the dissertation with precision; and Section 5.3 offers final reflections on the broader significance of the work and the research questions that remain open.

5.1 Summary of the Work

The research question addressed in this dissertation is clearly stated and answerable. Is it possible for a collaborative multi-agent architecture, including the Verifier Agent, to provide noticeably better pedagogical feedback than the corresponding single-agent system based on a language model? How costly would that be? The findings obtained experimentally are rather definitive. MA-V achieves the average quality score of 4.52 on the scale from 1.00 to 5.00, considerably outperforming the single-agent version (4.165; $p < .001$, Cohen's $d = .54$) and MA-Pipeline (4.215; $p < .006$, $d = .46$). The proportion of successfully completed tasks grows from 17.5% in SA up to 52.5% in MA-V. Verifier initiates revisions in 25% of tasks. Revisions result in a mean improvement of 0.73 score points, that is, the magnitude of effect reaches approximately $d = 1.42$ per task. The associated latency cost is estimated at 3.04x (11.1 vs. 3.7 seconds mean) and is thus considered reasonable for an educational application.

5.2 Principal Contributions

Three findings stand out as the main scientific contributions of this research. Firstly, the demonstration of the null effect of role specialization in isolation takes place. The comparison of MA-NV to SA ($p = 0.078$, $d = 0.29$) shows that task decomposition into roles of Tutor Agent and Feedback Agent, sans the verification procedure, fails to bring a statistically significant increase in quality. In doing so, the research explicitly contradicts the widespread assumption that task decomposition into roles brings an automatic benefit to the multi-agent architecture of an LLM and shifts the focus of further work onto the role of quality control instead. The development of multi-agent LLM applications for educational purposes requires practitioners to recognize that simply increasing the number of agents does not lead to any quality gains, while it is the verification and revision component of the architecture that deserves closer consideration.

Secondly, this study highlights consistency as the quality dimension uniquely influenced by the multi-agent architecture. The necessity of cross-agent coordination that cannot be

achieved in single-agent or single-review systems means that consistency, which displays the biggest absolute increase among all dimensions (by +0.450 in relation to SA in MA-V), also stands out as the most significant in the context of Bonferroni-corrected hypothesis tests for $p = 0.001 < \alpha' = 0.01$. It is crucial in an educational environment as the inconsistency of feedback actually leads to misinformation and actively discourages the student rather than just failing to be helpful. The cross-agent responsibilities of the Verifier make it unique in its ability to tackle this particular issue.

Thirdly, it is stated that the verification mechanism is floor-raising rather than uniform. The Verifier does not improve each answer equally – it focuses on identifying and fixing the worst possible first-generation outputs and thus compresses the left side of the distribution. This is evident both from the increased success rate and decreased standard deviation of all configurations except MA-VN. This view on verification as a floor-raiser allows recognizing the areas where the mechanism is most valuable – in hard tasks, stakes are high, and when a student needs particularly high-quality information for optimal learning.

Finally, the methodological contribution of this research consists of the practical implementation of formative feedback theory. The correlation of the elaborated feedback taxonomy of Shute with the tripartite prompt used by the Feedback Agent, the connection of Hattie & Timperley's feedback on the process level with the Verifier's criterion, and the parallel between the emphasis of the ITS literature on addressing misconceptions and the reference feedback point structure all contribute to a solid link between theoretical considerations and design. These connections fill the current gap in the literature, which lacks scientifically based multi-agent system engineering approaches.

5.3 Final Reflections

Indeed, the limitations of the paper are clear and significant. First, the lack of actual learning outcomes data hinders confirmation of causation between the quality scores and student achievement. Next, the evaluation process based on judging the LLM's performance by another LLM creates a cycle, which could have been broken by the input of human experts. Also, the focus on one topic and conversation turn limits the extent of possible generalizations. Finally, the simplistic nature of the student model, which cannot even be considered a full-fledged ITS by today's standards, also restricts the range of possibilities. Every limitation implies a different line of research—human validation, randomized controlled experiments, extending the system to multiple domains, integrating knowledge tracing, conducting conversations over several turns—that will form the basis of further study on multi-agent LLM-based tutoring systems.

"Bloom's two-sigma problem" dates back more than forty years and addresses the need to develop systematic means for delivering instruction with instructional quality comparable to that of human tutoring on the scale necessary for education systems.

However, the CMACF is not a solution to this issue—any single method, regardless of sophistication, would fail to meet the objective. Instead, this paper presents a principled advancement—a proof-of-concept that shows it is feasible to significantly improve the quality of AI-driven tutor feedback via architectural verification that can be done within an established theoretical framework and implemented efficiently, with minimal infrastructure costs. Still, the realm of artificial intelligence for education remains underdeveloped regarding designing tutoring systems that could truly command educational trust. This study adds one such principle—verifying AI tutoring quality using a specially designed Verifier agent—and provides a benchmark for further improvements.

Bibliography

- [1] B. S. Bloom, "The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring," *Educational Researcher*, vol. 13, no. 6, pp. 4–16, 1984.
- [2] L. S. Vygotsky, *Mind in Society: The Development of Higher Psychological Processes*. Cambridge, MA: Harvard University Press, 1978.
- [3] A. C. Pal et al., "Intelligent tutoring systems: A review of the state of the art," *Journal of Computer Science*, vol. 17, pp. 123–141, 2021.
- [4] J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier, "Cognitive tutors: Lessons learned," *The Journal of the Learning Sciences*, vol. 4, no. 2, pp. 167–207, 1995.
- [5] A. T. Corbett and J. R. Anderson, "Knowledge tracing: Modeling the acquisition of procedural knowledge," *User Modeling and User-Adapted Interaction*, vol. 4, pp. 253–278, 1994.
- [6] K. VanLehn, "The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems," *Educational Psychologist*, vol. 46, no. 4, pp. 197–221, 2011.
- [7] B. P. Woolf, *Building Intelligent Interactive Tutors*. Burlington, MA: Morgan Kaufmann, 2008.
- [8] A. C. Graesser, P. Chipman, B. C. Haynes, and A. Olney, "AutoTutor: An intelligent tutoring system with mixed-initiative dialogue," *IEEE Transactions on Education*, vol. 48, no. 4, pp. 612–618, 2005.
- [9] V. Rus, S. DMello, X. Hu, and A. Graesser, "Recent advances in conversational intelligent tutoring systems," *AI Magazine*, vol. 34, no. 3, pp. 42–54, 2013.
- [10] S. DMello and A. Graesser, "AutoTutor and affective autotutor: Learning by talking with cognitively and emotionally intelligent computers that talk back," *ACM Transactions on Interactive Intelligent Systems*, vol. 2, no. 4, pp. 1–39, 2012.
- [11] V. Aleven, B. McLaren, I. Roll, and K. Koedinger, "Toward meta-cognitive tutoring: A model of help seeking with a cognitive tutor," *International Journal of Artificial Intelligence in Education*, vol. 16, pp. 101–128, 2006.
- [12] B. P. Woolf et al., "The effect of motivational learning companions on low achieving students and students with disabilities," in *Proc. ITS 2010*, Springer, pp. 327–337, 2010.
- [13] OpenAI, "GPT-4 technical report," arXiv preprint arXiv:2303.08774, 2023.
- [14] T. B. Brown et al., "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.
- [15] X. Wang et al., "Self-consistency improves chain of thought reasoning in language models," in *Proc. ICLR 2023*, 2023.
- [16] T. Kojima et al., "Large language models are zero-shot reasoners," in *Proc. NeurIPS 2022*, 2022.
- [17] R. Bommasani et al., "On the opportunities and risks of foundation models," arXiv

preprint arXiv:2108.07258, 2021.

[18] J. S. Park et al., "Generative agents: Interactive simulacra of human behavior," in Proc. UIST 2023, ACM, 2023.

[19] G. Li et al., "CAMEL: Communicative agents for Mind exploration of large language model society," in Proc. NeurIPS 2023, 2023.

[20] Q. Wu et al., "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," arXiv preprint arXiv:2308.08155, 2023.

[21] C. Chan et al., "ChatEval: Towards better LLM-based evaluators through multi-agent debate," arXiv preprint arXiv:2308.07201, 2023.

[22] Y. Du et al., "Improving factuality and reasoning in language models through multiagent debate," arXiv preprint arXiv:2305.14325, 2023.

[23] T. Liang et al., "Encouraging divergent thinking in large language models through multi-agent debate," arXiv preprint arXiv:2305.19118, 2023.

[24] A. Vaswani et al., "Attention is all you need," in Proc. NeurIPS 2017, pp. 5998–6008, 2017.

[25] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in Proc. NAACL-HLT 2019, pp. 4171–4186, 2019.

[26] L. Ouyang et al., "Training language models to follow instructions with human feedback," in Proc. NeurIPS 2022, 2022.

[27] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in Proc. NeurIPS 2022, 2022.

[28] Anthropic, "Claude model card and usage policies," Anthropic, San Francisco, CA, Tech. Rep., 2024.

[29] R. Nkambou, J. Bourdeau, and R. Psyche, *Advances in Intelligent Tutoring Systems*. Berlin: Springer, 2010.

[30] V. J. Shute, "Focus on formative feedback," *Review of Educational Research*, vol. 78, no. 1, pp. 153–189, 2008.

[31] J. Hattie and H. Timperley, "The power of feedback," *Review of Educational Research*, vol. 77, no. 1, pp. 81–112, 2007.

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

[33] A. S. Lan and R. G. Baraniuk, "A compressed sensing-based approach for subspace identification in feedback-based learning," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 2, no. 3, pp. 279–296, 2016.

[34] C. Piech et al., "Deep knowledge tracing," in Proc. NeurIPS 2015, pp. 505–513, 2015.

[35] I. Abdelaziz et al., "A survey of AI methods in autonomous planning and decision making for language-model-based agents," arXiv preprint arXiv:2308.11432, 2023.

[36] L. Zheng et al., "Judging LLM-as-a-judge with MT-bench and chatbot arena," in Proc.

NeurIPS 2023, 2023.

[37] M. Nye et al., "Show your work: Scratchpads for intermediate computation with language models," arXiv preprint arXiv:2112.00114, 2021.

[38] A. Madaan et al., "Self-refine: Iterative refinement with self-feedback," in Proc. NeurIPS 2023, 2023.

[39] Y. Shen et al., "HuggingGPT: Solving AI tasks with ChatGPT and its friends in Hugging Face," in Proc. NeurIPS 2023, 2023.

[40] E. Wenger, "Misconceptions in CS education: A taxonomy and implications for curriculum design," in Proc. SIGCSE 2015, ACM, pp. 432-437, 2015.

[41] B. Paassen et al., "Automated feedback for CS1 courses: An empirical study of misconception frequency and instructional impact," in Proc. EDM 2023, 2023.

[42] R. Agarwal and A. Singhal, "Effect of temperature on large language model outputs for code generation tasks," arXiv preprint arXiv:2309.10062, 2023.

[43] J. Cohen, Statistical Power Analysis for the Behavioral Sciences, 2nd ed. Hillsdale, NJ: Lawrence Erlbaum Associates, 1988.

[44] Q. McNemar, "Note on the sampling error of the difference between correlated proportions or percentages," Psychometrika, vol. 12, no. 2, pp. 153-157, 1947.

[45] D. M. Ziegler et al., "Fine-tuning language models from human preferences," arXiv preprint arXiv:1909.08593, 2019.