Technical University of Munich

School of Computation, Information and Technology
-- Informatics --

Bachelor's Thesis in Information Systems

AI-Driven Workflow for Inconsistency Remediation in Programming Exercises

KI-gesteuerter Workflow zur Inkonsistenzbehebung in Programmieraufgaben

Author:	Mikhail Khinevich
Supervisor:	Prof. Dr. Stephan Krusche
Advisors:	Tobias Wasner, M.Sc., Felix T.J. Dietrich, M.Sc.
Start Date:	06.11.2025
Submission Date:	07.04.2026

Abstract

Automated quality assurance for multi-artifact programming exercises is an important component of scalable computer science education. Existing systems that use large language models (LLMs) to detect inconsistencies between exercise components demonstrate fundamental feasibility, but suffer from significant limitations in detection precision, leading to high rates of false positives. The current approach does not explicitly model the relationships between exercise components, such as problem statement, code templates, sample solutions and tests repositories.

We extend the existing system with a new approach for continuous context representation to address these limitations. This approach models semantic and structural relationships between exercise artifacts and provides the LLM-Based consistency check system with structured domain knowledge to improve the precision of detecting inconsistencies. This thesis further extends the concept beyond detection and moves towards automatic correction by generating suggested fixes in the form of applicable `git patch` files. This work aims to create a more reliable and sustainable system for ensuring the quality of education.

1 Introduction

The increasing number of students studying computer science presents a serious challenge for manual grading and creation of high-quality programming exercises. Modern learning management systems (LMS) address this challenge by providing instructors with tools to manage complex exercises, consisting of multiple elements. These exercises often include components such as problem statements, code templates, sample solutions, and automated test suites that reflect real-world software development practices. Successful management of these related components is critical to ensuring effective interactive learning at scale.

Technical University of Munich developed the Artemis platform, an open-source LMS, specifically designed for this environment. It supports interactive learning and automatic assessment in programming courses, based on version control and continuous integration [KS18]. A central feature of Artemis is its robust support for multi-component programming exercises, which are essential for developing practical skills in students. While Artemis automates the assessment workflow, creating high-quality complex exercises remains a challenging task for instructors.

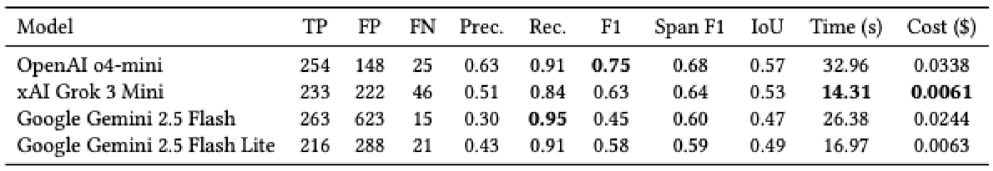
To support this process, the Hyperion service provides artificial intelligence (AI) based assistance within the Artemis ecosystem. This intelligent tool improves the exercise creation process. It aligns with the broader goals of artificial intelligence in education [TZG22], which aims to use intelligent systems to support instructors and create personalized learning experiences. Hyperion's core service provides automated feedback to instructors on the consistency of programming exercises. It analyzes various exercise artifacts to help them identify potential issues before delivery to students.

2 Problem

Inconsistencies in multi-artifact programming exercises often arise when instructors modify one component, such as the problem statement, but overlook the necessary corresponding changes in others, like the code template. For example, the assignment may specify that students should implement a function named `calculateAverage`, while the provided code template contains a placeholder for a function named `computeMean` [Die+25]. These unintended inconsistencies between different components of the exercise are a major problem that has significant negative consequences for the two main participants in the learning process: the students, who solve the exercises, and the instructors, who create them.

Contradictions between the assignment and the provided code template create unnecessary cognitive load for the students. This cognitive load forces them to divert intellectual resources away from the intended learning goal and direct them toward the unrelated task of aligning contradictory information. Such inconsistencies can lead to confusion, disrupt the learning process, and ultimately compromise the fairness and effectiveness of assessment [Die+25].

Instructors face the challenge of ensuring the quality and consistency of all learning materials. Detecting and resolving these issues is a mostly manual, time-consuming, and error-prone process. While Hyperion's AI-based quality assurance system aims to facilitate this workload and increase the value of automation, its performance has notable limitations, as detailed in Figure 1. The system's low precision leads to a high rate of false positives. This thesis hypothesizes that a root cause for this low precision is that the current approach treats the exercise artifacts as a single block of text, without explicitly modeling the relationships between them.



Model	TP	FP	FN	Prec.	Rec.	F1	Span F1	IoU	Time (s)	Cost ($)
OpenAI o4-mini	254	148	25	0.63	0.91	**0.75**	0.68	0.57	32.96	0.0338
xAI Grok 3 Mini	233	222	46	0.51	0.84	0.63	0.64	0.53	**14.31**	**0.0061**
Google Gemini 2.5 Flash	263	623	15	0.30	**0.95**	0.45	0.60	0.47	26.38	0.0244
Google Gemini 2.5 Flash Lite	216	288	21	0.43	0.91	0.58	0.59	0.49	16.97	0.0063

Figure 1: Overall results across the 1 049 analysed runs. F1-score measures a system's overall accuracy by calculating the harmonic mean of precision and recall. A high F1-score indicates a good balance between avoiding false positives (high precision) and not missing true issues (high recall). [Die+25].

3 Motivation

Ensuring the quality and consistency of educational materials is one of the main challenges in scalable computer science education. The scientific significance lies in the need to develop reliable automated tools capable of analyzing both natural language and formal source code. An empirical analysis of the baseline system strengthens the motivation for a new approach. This analysis, illustrated in Figure 2, reveals that the system's limitations do not significantly correlate with factors like input length, indicating that this thesis must address a more fundamental, structural problem.

Developing systems capable of bridging these structural and semantic gaps between natural language instructions and source code is a complex interdisciplinary task at the intersection of software engineering and artificial intelligence in education. A reliable solution to this problem opens up new perspectives for both students and instructors. For students, the key benefit is a learning environment that is free from confusing and contradictory information. This allows students to engage in a seamless learning process, dedicating their full cognitive resources to mastering the necessary programming concepts, which leads to a smoother, more efficient, and fairer educational experience that focuses on skill development.

For instructors, having a reliable automated workflow is equally important. This tool provides them with an efficient and trustworthy process for ensuring the quality of their educational materials. By trusting the system with this quality control task, instructors can devote their valuable time and expertise to higher-level pedagogical tasks. This allows them to focus on designing more creative exercises and providing individualized student feedback, which is essential for the scalable delivery of high-quality computer science education.

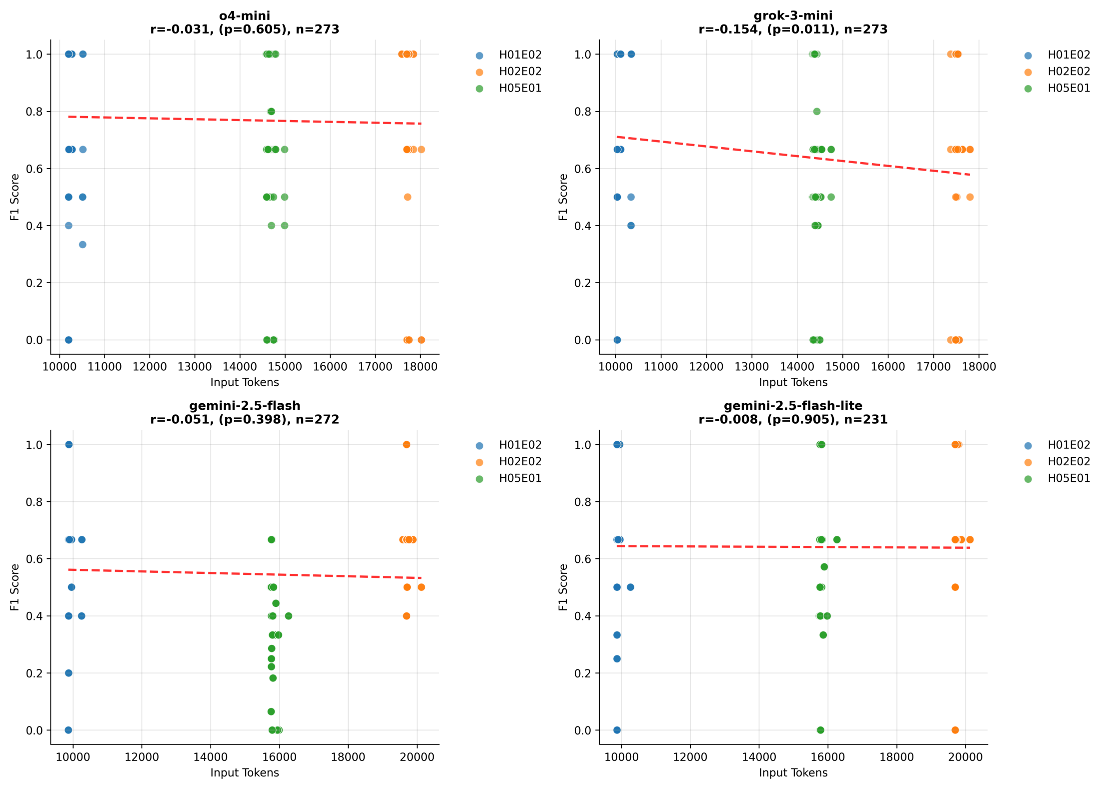


Figure 2: Relationship between prompt input token and F1-value for 4 different LLMs, revealing a weak and statistically insignificant correlation for all models.

4 Objective

This thesis aims to enhance an existing AI-driven consistency checker for programming exercises. To provide a foundation for this experimental work, the first two objectives focus on establishing a robust benchmarking pipeline and extending the evaluation dataset for more comprehensive testing. Building upon this, the two subsequent objectives implement and evaluate the core technical contributions: enhancing the system's reasoning with a structured context model and extending its functionality from detection to automated remediation. We define the following objectives to achieve these goals:

1. Establish an On-Demand Benchmarking Workflow.
2. Extend Consistency Check Dataset.
3. Develop a Workflow for Inconsistency Remediation.
4. Enhance Context Representation.

4.1 Establish an On-Demand Benchmarking Workflow

The first objective is to establish an integrated and on-demand benchmarking workflow, using PECV (Programmin Exercise Consistency Verification)[1] Dataset for the Consistency Check within the Artemis platform. This approach is necessary to systematically evaluate new approaches. Instead of a continuous process, a developer will trigger this workflow on-demand through a GitHub Actions. This capability allows developers to conveniently run the benchmark against a specific version of consistency check approach to measure its performance, so that developers can contribute measurable improvements.

When a developer triggers the workflow, it will clone the necessary repositories, create an evaluation course and programming exercise for each variant in Artemis. Execute the full benchmark, and generate an evaluation report. It serves as the primary tool for evaluating new approaches this thesis develops in subsequent objectives and ensures that all results are reproducible and directly comparable to the baseline.

[1]PECV-Bench Repository

4.2 Extend Consistency Check Dataset

The second objective is to extend the existing benchmark dataset to enable a more comprehensive evaluation. The foundational dataset from [Die+25] is a crucial starting point but is limited in the scope and variety of inconsistencies it covers. This objective will focus on expanding the dataset to better reflect the range of potential issues found in real-world educational materials and support different programming languages.

Furthermore, this process will include the creation of corresponding "ground truth" git patch files for these inconsistencies. These patches will serve as the correct answers, and the final evaluation will compare them against the remediations our system generates from Section 4.3. These new artifacts will provide the necessary data to test the consistency checker and validate the output of the automated remediation workflow.

4.3 Develop a Workflow for Inconsistency Remediation

Beyond improving detection, this thesis aims to make the remediation process more practical for Artemis Instructors by extending the system's functionality from simple detection to automated correction. This aligns with the established research area of Automated Program Repair [GMM19]. This work will enhance the system to not only describe a consistency issue, but also to generate a suggested fix in the commonly accepted `git patch` format. This transforms the system's output from a simple report into an actionable tool for instructors.

The remediation workflow, illustrated in Figure 3, is an evaluation loop in Artemis, that first structures the knowledge about programming exercise and performs a consistency check. When the system detect a Consistency Issue, it begings a multi-step evaluation process. First, the system generates a corresponding `git patch` file. Second, it verfies the patch's applicability and correctness in a temporary, simulated sandbox environment by applying the patch and re-running the exercise full test suite. The outcome of this verification leads to an automated acceptance decision [Zha+24].

If we accept the suggested fix, the process concludes by returning the valid Consistency Issues and its corresponding git patches. If not, the system generates an Evaluation Report, with the failure details and checks if it has a remaining retry budget. If the budget allows, the process loops back to the Structure Knowledge step to refine its approach for another attempt. When no budget remains, the workflow terminates. This entire verification process occurs locally: The system will not commit or push any changes to the Artemis repositories. This self-correcting mechanism improves the reliability of the automated remediation.

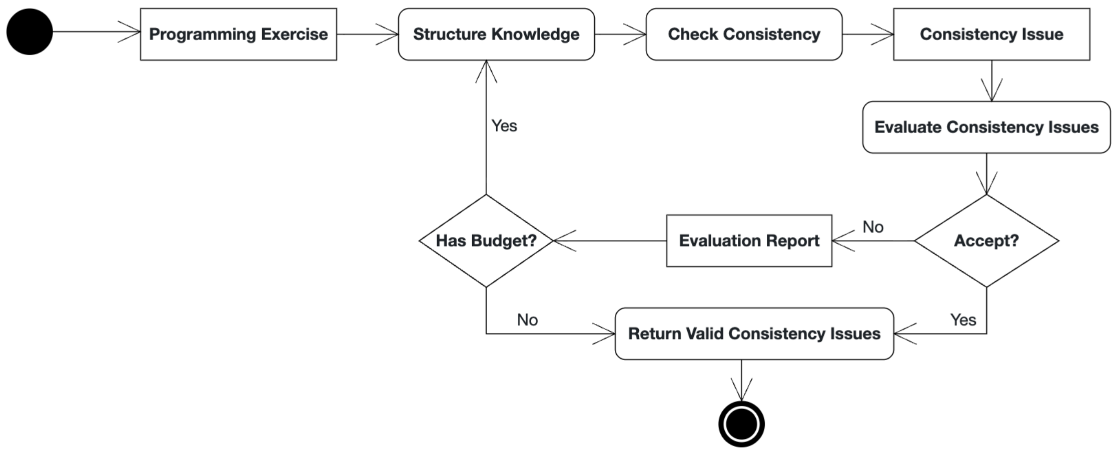


Figure 3: Activity diagram of the proposed iterative workflow for inconsistency remediation.

4.4 Enhance Context Representation

The fourth objective is to address the fundamental problem of the system's low detection precision. This thesis will implement and evaluate a system that uses a Knowledge Graph (KG) to formally model Artemis exercise artifacts and their interdependencies [PC24]. KG provides a machine-readable schema of the domain that captures the semantic and structural relationships between components like a problem statement, a specific method in a code template, and its corresponding test case. This approach provides the system with structured knowledge to guide its reasoning process [CJX20].

The implementation will explore a Graph Retrieval-Augmented Generation (GraphRAG) approach [ZC24]. This technique allows the LLM to query the structured KG to retrieve precise, relevant context about artifact relationships, rather than processing a single, large block of unstructured text. This method of structured information retrieval will provide the LLM with a more consistent and context-aware view of the exercise, which should reduce uncertainty, lower the false positive rate, and ultimately lead to a significant improvement in detection performance.

5 Schedule

The work for this thesis begins on November 6, 2025, and concludes on April 7, 2026. This thesis structures the project into the following nine iterations to ensure a clear focus and measurable progress throughout the five-month period:

1. **Iteration (Weeks 1-4): CI Pipeline Implementation** (Objective 4.1)
   - Integrate Consistency Check with the continuous integration capabilities of Artemis.
   - Automate the process of running the checker against the benchmark dataset for every new inconsistency remediation approach.
   - Ensure all experimental results are logged and reproducible within the pipeline.
2. **Iteration (Weeks 5-6): Dataset Expansion** (Objective 4.2)
   - Create new exercise variants with artificial consistency issues.
   - Expand the dataset to better reflect the range of potential issues found in real-world materials.
3. **Iteration (Weeks 7-8): Ground Truth Remediation** (Objective 4.2)
   - Generate the corresponding “ground truth” `git patch` files for new inconsistencies.
   - Finalize the extended dataset for use in subsequent evaluation tasks.
4. **Iteration (Weeks 9-10): Automated Remediation Workflow** (Objective 4.3)
   - Implement the core logic for the LLM to generate a `git patch` based on a detected inconsistency.

- Implement an automated check for the applicability of a generated `git patch`.

5. **Iteration (Weeks 11-13): Automated Validation Loop** (Objective 4.3)
   - Develop the automated validation loop to check `git patch` correctness in a simulated environment.
   - Integrate this validation check into a re-iteration loop to enable the system to self-correct invalid patch suggestions.
6. **Iteration (Weeks 14-16): Knowledge Graph Implementation** (Objective 4.4)
   - Design the formal schema for the Knowledge Graph to represent Artemis exercise artifacts, their dependencies and relationships.
   - Implement the logic to parse raw exercise files and populate the Knowledge Graph with the data.
7. **Iteration (Weeks 17-18): GraphRAG Pipeline Integration** (Objective 4.4)
   - Integrate the populated Knowledge Graph into the LLM analysis pipeline.
   - Engineer and test new GraphRAG prompt strategies that leverage the structured information.
8. **Iteration (Weeks 19-20): Final System Evaluation (All Objectives)**
   - Run the complete, enhanced system against the extended benchmark dataset using the Artemis CI pipeline.
   - Collect and analyze the final performance metrics for both detection and remediation (patch success rate).
9. **Iteration (Weeks 21-22): System Refinement and Replication Package**
   - Conduct end-to-end system testing to identify and resolve any remaining bugs.
   - Finalize the codebase and prepare the replication package with all necessary documentation.

Bibliography

[KS18] S. Krusche and A. Seitz, “ArTEMiS: An Automatic Assessment Management System for Interactive Learning,” in *SIGCSE '18: Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, Association for Computing Machinery, 2018.

[TZG22] O. Tapalova, N. Zhiyenbayeva, and D. Gura, “Artificial Intelligence in Education: AIEd for Personalised Learning Pathways,” *Electronic Journal of e-Learning*, 2022.

[Die+25] F. Dietrich, Y. Zhou, T. Wasner, S. Krusche, and M. Acosta, “LLM-Based Multi-Artifact Consistency Verification for Programming Exercise Quality Assurance,” 2025.

[GMM19] L. Gazzola, D. Micucci, and L. Mariani, “Automated Program Repair,” *ACM Computing Surveys (CSUR)*, vol. 52, no. 3, 2019, [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3318162

[Zha+24] Y. Zhao *et al.*, “CodeJudge-Eval: Can Large Language Models be Good Judges in Code Understanding?,” 2024. [Online]. Available: https://arxiv.org/abs/2408.10718

[PC24] J. Z. Pan and o. Chen, “Unifying Large Language Models and Knowledge Graphs: A Roadmap,” 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10387715

[CJX20] X. Chen, S. Jia, and Y. Xiang, “Knowledge reasoning over knowledge graph,” 2020. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0957417419306669

[ZC24] Z. Zhang and o. Chen, “GraphRAG: A Graph-Based Retrieval-Augmented Generation Method,” 2024. [Online]. Available: https://arxiv.org/abs/2405.16506

Transparency in the use of AI tools

In preparing this thesis, I utilized Grammarly and Apple Intelligence for grammar and style correction across all sections, ensuring clarity and coherence in my writing. I used DeepL to enhance language quality. I used GoogleAI to generate initial drafts and expand on ideas. I have carefully checked all texts created with these tools to ensure they are correct and make sense.