2025 IEEE/ACM 47th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion) | 979-8-3315-3683-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICSE-COMPANION66252.2025.00071

MUARF: Leveraging Multi-Agent Workflows for Automated Code Refactoring

1st Yisen Xu
Gina Cody School of Engineering and Computer Science
Concordia University
Montreal, Canada
yisen.xu@mail.concordia.ca

***Abstract*—Refactoring is crucial for maintaining a project, but it requires developers to understand code structure and system design principles well. Recent research on Large Language Models(LLMs) has shown their great capability for handling complex tasks, making them a possible solution for overcoming these challenges. In this paper, we propose *MUARF*, an LLM-based solution designed to automate method-level code refactoring, aiming to generate correct, high-quality, and human-like refactored code. *MUARF* leverages Contextual Retrieval-Augmented Generation to search for similar refactoring samples for few-shot learning, uses Multi-Agent Workflow to simulate the human refactoring process, and integrates advanced software engineering tools (e.g., RefactoringMiner, PurityChecker, StyleChecker) to assist refactoring. Evaluation results show that *MUARF* achieves a compilation pass rate of 86.5% and a test success rate of 83.8% for the refactored code it generates. Additionally, metrics such as CodeBLEU score and AST Diff accuracy—which compare human-refactored code with the output of *MUARF* —highlight the generated code is human-like. The ablation results show that *RefactoringMiner* and *Agentware* made the greatest contribution to *MUARF*.**

***Index Terms*—Code Refactoring, Large Language Model, Multi-Agent Communication, Contextual Retrieval-Augmented Generation, Prompt Engineering**

I. Introduction

Refactoring code is essential for maintaining and evolving a software project, as it improves code quality, enhances maintainability, and ensures that the system remains adaptable to new requirements [1]. While many automated tools exist to assist developers with refactorings, such as WitchDoctor [2] and BeneFactor [3], these tools exhibit some challenges. For instance, these tools often rely on predefined rules, which lack a deep understanding of the project's domain-specific context, fail to capture the underlying semantic meaning of the code, and lack the flexibility to adapt to new refactoring patterns or evolving rules. Addressing these challenges requires the development of more capable solutions.

Recent research on Large Language Models(LLMs) has shown their great capability for handling complex tasks, making them a possible solution for overcoming these challenges. These are important steps for automating the refactoring job of existing code [1]. However, due to the complexities of refactoring tasks, it is not effective to guide LLMs to perform refactoring by designing simple prompts, because while optimizing the design of existing code, it is also necessary to ensure that refactoring does not destroy the logic of existing code. LLMs cannot obtain this feedback information from the initial prompt and are likely to generate code that changes the behavior of the program. Meanwhile, there are many different ways to refactor the same piece of code. The implementation of a refactoring may cause design problems elsewhere. To address the challenges faced by LLMs in refactoring tasks, we propose *MUARF*. *MUARF* leverages core technologies—*Prompt Engineering*, *Agentware*, and *Contextual Retrieval-Augmented Generation(RAG)*—to enhance LLMs with additional refactoring knowledge, including historical examples of similar refactoring solutions. By integrating these advanced techniques, *MUARF* aims to help LLMs produce refactored code that is accurate, high-quality, and human-like.

We summarize the main contributions as follows:

1) **Originality**: We introduced several advanced techniques to enhance the capabilities of LLMs in refactoring code that is not only correct but also high-quality and human-like. Furthermore, we evaluated the individual contributions of these techniques to the overall performance. Additionally, we examined their effectiveness across various refactoring types to better understand their impact.
2) **Evaluation**: We conducted experiments to evaluate *MUARF*'s effectiveness, perform an ablation study, and compare it with existing tools. We used *Gson* as the benchmark and the *Oracle Refactoring Dataset* as the source for few-shot examples. Furthermore, we analyzed *MUARF*'s performance by comparing it with *RawGPT*, *human-written code*, and other LLM-based refactoring tools such as *EM-Assistant* and *Move-Method Assistant*.

II. Methodology

In this section, we introduce ***MUARF* (*MU*lti-*A*gent workflow for *ReF*actoring)**, an LLM-based, agent-driven solution designed to automatically refactor method-level code exhibiting bad smells (e.g., long methods). Fig. 1 illustrates the key components of our methodology: *Contextual Refactoring Extraction*, *Multi-Agent Workflow*, and *Self-Refinement*. Each phase of our approach is detailed below, along with an explanation of its implementation.

2574-1934/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSE-Companion66252.2025.00071

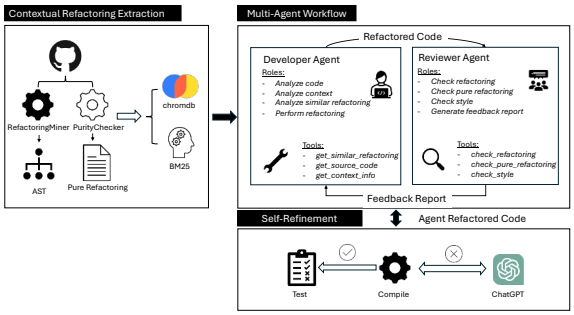


Fig. 1. Overview of MUARF, an LLM-based approach for automatically refactoring method-level code.

A. *Contextual Refactoring Extraction*

To acquire suitable few-shot examples for LLMs, we constructed a pre-built database of refactored examples sourced from open-source projects. As shown in Fig 1, we first used RefactoringMiner [4] and PurityChecker[1] to extract pure refactored code from these projects. Next, we proceeded with the Contextual RAG construction to organize and build the reference database. Once a bad smell code is provided, a search algorithm retrieves relevant refactored code examples to serve as few-shot.

B. *Multi-Agents Workflow*

MUARF defines two key agents: the developer agent and the reviewer agent. The developer agent is responsible for generating and optimizing the code, while the reviewer agent reviews the code and provides feedback or suggestions to the developer agent for further improvement. By introducing communication via these agents, *MUARF* can iteratively refine the code based on the specific situation, allowing it to dynamically change the workflow for an expected outcome. This approach avoids the need for pre-set thresholds or fixed iteration counts, which are often required in traditional workflows, making *MUARF* more flexible and effective in handling diverse refactoring scenarios.

C. *Self-Refinement*

After obtaining the final refactored code, MUARF replaces the original code with the refactored version and proceeds to compile and test it. If the compilation fails, MUARF attempts automatic repair by providing the code and compilation log to ChatGPT, asking it to generate the corrected code. If the repaired code compiles successfully, it moves directly to the testing phase. If the compilation still fails, the process is retried up to three times before returning the final result.

III. Evaluation

RQ1: How Effective is MUARF in Refactoring Code?

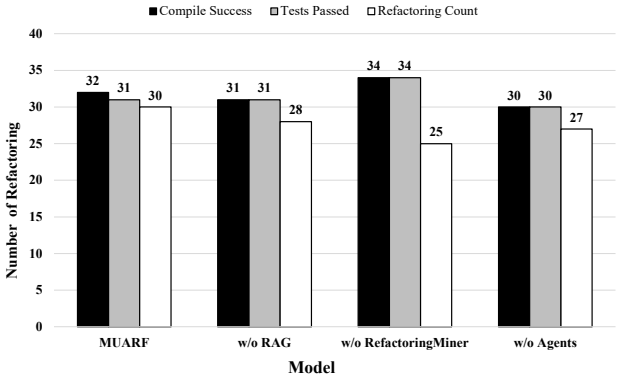
In comparison with *RawGPT*, *MUARF* demonstrated superior performance. The refactored code generated by *MUARF* was successfully compiled and tested in 32 out of the 37 refactorings. Additionally, using *RefactoringMiner*, 30 refactorings could be identified from the refactored code produced by *MUARF*.

[1]PurityChecker, https://github.com/pedramnoori/PurityChecker/

TABLE I
A TOTAL OF 37 BAD SMELL CODE SAMPLES WERE EXTRACTED FROM THE *Gson* PROJECT.

Approach	Metrics	Total Before Refactor	Total After Refactor
Human-written	Compile Success	37	37
	Tests Passed	37	37
	Refactoring Count	-	37
RawGPT	Compile Success	37	23
	Tests Passed	37	23
	Refactoring Count	-	15
	CodeBLEU Score	-	0.302
	AST Diff Accuracy	-	0.995
MUARF	Compile Success	37	32
	Tests Passed	37	31
	Refactoring Count	-	30
	CodeBLEU Score	-	0.589
	AST Diff Accuracy	-	0.997

Fig. 2. Contribution of Components in *MUARF*: The numbers represent the results across three metrics.



RQ2: What is the individual contribution of each component in MUARF?

In this ablation study, we analyzed the impact of removing specific components from MUARF. The results reveal that feedback from RefactoringMiner and the Multi- Agent communications are the most critical contributors.

References

[1] E. R. Murphy-Hill, C. Parnin, and A. P. Black, "How we refactor, and how we know it," in 31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings. IEEE, 2009, pp. 287–297. 347–367, 2009. [Online]. Available: https://doi.org/10.1109/TSE.2009.1

[2] S. R. Foster, W. G. Griswold, and S. Lerner, "Witchdoctor: Ide support for real-time auto-completion of refactorings," in 2012 34th international conference on software engineering (ICSE). IEEE, 2012, pp. 222–232.

[3] X. Ge, Q. L. DuBose, and E. Murphy-Hill, "Reconciling manual and automatic refactoring," in 2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012, pp. 211–221.

[4] N. Tsantalis, A. Ketkar, and D. Dig, "Refactoringminer 2.0," IEEE Transactions on Software Engineering, vol. 48, no. 3, pp. 930–950, 2022.