AI Applied to Software Refactoring

Alexia Jennings, Jenan Qasem, Sanskriti Gyawali, Diwashna Poudel, Arushi Chaturvedi

April 15, 2025

Table of contents

Introduction

The evolution of artificial intelligence is markedly changing how developers practice software refactoring. Traditionally, it was manually revising code and reorganizing files with the intention of improving readability, structure, and performance without changing functionality, or looking at the software in a stepwise fashion. While the basic idea is simple, practically doing it has never been easy. If there are more developers contributing to a project, the codebase is guaranteed to become a tangled mess. Decades of irrational legacy logic, volatile naming conventions, duplicated code, and an assortment of garbage code led to the codebase being a bloated and tangled monstrosity. The cleanup process is rather brutal; it consumes an absurd amount of resources that can be riddled with mistakes, and it is all too often neglected when new features are waiting for development.

This is where modern tools powered by AI would assist with developing software. Today's AI instruments can assist through the entire process of cleaning up and optimizing code, from spotting redundant routines and dead code to helping the developer with naming variables and even suggesting automated replacements that enhance the application's structure, quality, and ease of future maintenance. The AI does not merely operate on predefined guidelines; it uses machine learning and deep learning to work off large datasets of code to determine patterns that even the most meticulous human engineers might miss.

Artificial Intelligence is trained on massive open-source and enterprise codebases, which perform deep learning and machine learning using complex algorithms and neural networks, resulting in well-structured, clean, efficient code, and verbose multi-language support. With this, AI not only flags coding standard violations but also intelligently performs meaningful refactorings.

Natural Language Processing

As powerful as Natural Language Processing (NLP) is, AI-assisted refactoring is taken a notch higher once it is integrated with NLP. Models flexing NLP capabilities can understand human language, and with that comes the ability to match workflows within code to the intent of the programmers at a more refined level. Having the ability to understand naming conventions, the AI can confirm computation of averages in functions with names such as calculate_average. In case other calculations are done, the AI has every capacity to flag that as well. This type of validation is greatly helpful in legacy documentations riddled with comments and function parts diverging as time marches on.

Reinforcement Learning

AI algorithms facilitate the improvement of static code analysis, as well. Compared to traditional analyzers, AI-enhanced dynamics are a leap forward in functionality and efficiency. The latter still works with algorithms that follow strict boundaries. AI-based analyzers, on the contrary, can identify issues that regard unclear logic within the code irrespective of its accuracy regarding syntax and logic structures. Such cases may involve lengthy methods and classes, numerous parameters, or even tight relationships between modular components. AI can identify issues without having to rely on rules, as it derives the classification from intelligent and problematic codebases where practices differ.

An example of this is the increasingly adopted cutting-edge practice known as reinforcement learning. An AI agent, for instance, modifies code and monitors whether the test cases, performance, or compilation succeed. It can adjust its strategies accordingly, becoming more adept over time. This demonstrates how human developers learn through trial and error, solving problems, and progressively refining their constructive approaches.

Advanced models, like CodeBERT and GraphCodeBERT, also deploy these late innovations. These models analyze not just the sequential order of the lines of code, but also the relation between different parts of a programmable structure. These models are capable of providing a summary of complicated functions, an outline of better-suited names, undetected documentation, or contradictions between methods and their name usages. The effects of artificial intelligence on software development are advancing at an unprecedented speed as it touches issues of integration into daily routines. Artificial Intelligence is no longer a concept of the research units or tailored instruments. It is on the verge of becoming incorporated into the surrounding development. Famous IDEs such as Visual Studio Code or JetBrains IDEs become IntelliJ with AI contained suggestions for code improvement, style corrections, autocompletions, and other features powered by Copilot from GitHub and IntelliCode.

Such tools are exceptionally useful, especially when bearing in mind the unrivaled quality of code produced because the cognitive load and time consumed by developers is lowered when a new codebase, and everything contained within it, is provided to offer to novices. Even with swift growth, the pace of advancement relics of the past strongly combine with a rule-based approach that still remains key for supplementing static systems into the structures of modern advanced ML tools.

Potential Challenges

Their speed, reliability, and ease of verification make systems particularly valuable in controlled environments or where explainability is needed. However, their inflexibility can be a problem in more complicated or rapidly changing codebases. On the other hand, AI systems learn from context and improve over time, which lets them deal with complex, ever-changing software systems.

Of course, there are challenges. AI models need vast amounts of data and computational power to train. If not validated correctly, they can introduce subtle errors that can go unnoticed. Still, the

potential is enormous. As these systems mature, we will likely see deeper integration where AI not only assists with cleaning up code but actively participates in writing clean and maintainable code from the very beginning.

Conclusion

Soon, AI tools could operate as collaborative intelligent assistants, working with developers in real-time to not only complete code, but also understand the problem space, enforce architectural patterns, optimize system performance, and ensure security compliance. The intent is not to displace developers, but to enhance their work and enable them to direct their efforts towards creative problem-solving paradoxes, or towards high-level design, while freeing them to think constructively.

At the end of the day, the impact of AI can be felt across the entire spectrum of software development, including software refactoring. AI tools enable smoother, manageable, dependable, and optimized software systems, which allow developers to create better applications with ease.

References

Alenezi, M., Akour, M. (2025). AI-driven innovations in software engineering: A review of current practices and future directions. Applied Sciences, 15(3), 1344. https://www.mdpi.com/2076-3417/15/3/1344

Beecrowd. (2024, November 6). AI & intelligent refactoring: How artificial intelligence is transforming code refactoring. beecrowd. https://beecrowd.com/blog-posts/ai-intelligent-refactoring/

Cogent University. (n.d.). AI-powered code optimization: Redefining software engineering standards. Cogent University. https://www.cogentuniversity.com/post/ai-powered-code-optimization-redefining-software-engineering-standards

DePalma, K., Miminoshvili, I., Henselder, C., Moss, K., AlOmar, E. A. (2024). Exploring ChatGPT's code refactoring capabilities: An empirical study. Expert Systems with Applications, 249(B), 123602. https://www.sciencedirect.com/science/article/pii/S0957417424004676

Gruman, G. (2024, December 2). Refactoring AI code: The good, the bad, and the weird. InfoWorld. https://www.infoworld.com/article/3610521/refactoring-ai-code-the-good-the-bad-and-the-weird.html

Orosz, G., Osmani, A. (2025, January 5). How AI-assisted coding will change software engineering: Hard truths. The Pragmatic Engineer. https://newsletter.pragmaticengineer.com/p/how-ai-will-change-software-engineering