AI Assisted/Automated code refactoring

How the development of AI may impact the future of code refactoring

Loris Tomassetti
Linz, Austria
loris.tomassetti@outlook.com

Alexander Weißenböck
Linz, Austria
alewei934@gmail.com

***Abstract*—This Paper aims to shed light onto developments in AI Assisted/Automated Code refactoring, how it can help the indtustry and which models work most efficiently to tackle different challenges code refactoring brings. This will be done by going over various literature describing first the challenges at hand and afterwards discussing several possible solutions that have been tested to gain a greater understanding and to generate an informed outlook into further developments of this technology.**

***Index Terms*—machine learning algorithms, software code refactoring, deep neural network**

I. Introduction

Refactoring, as defined by Fowler [3], is "the process of changing a software system in such a way that does not alter the external behaviour of the code yet improves its internal structure". More and more empirical studies have since established a positive correlation between refactoring operations and code quality metrics. All this evidence hints at refactoring being a high-priority concern for software engineers. [1].
However, deciding when and how to refactor can prove to be a challenge for developers. Refactoring in an early stage may cost too much for what you're getting out of it, and refactoring too late may cause the refactor to be an even bigger time commitment. [6]
Tools have been in the hands of many developers to make this process more streamlined for years now. Analytics tools to sniff out bugs or give hints on how to improve code quality such as PMD, ESLint, and Sonarqube can be integrated into different stages of a developers' workflow, e.g. inside IDEs, during code review or as an overall quality report. [1]
Taking a closer look at these tools, however, reveals that they commonly have a lot of false positives, making developers lose their confidence in them. Often, the detection strategies are based on hard thresholds of just a handful of metrics, such as lines of code in a file (e.g. PMD's famous "problematic" classification occurring once a method reaches 100 lines per default). These simplistic ways of detection simply aren't able to capture the full complexity of modern software systems.
Manually analyzing hundreds of metrics and figuring out which ones are the cause of technical dept is very hard and almost impossible for tool developers, which is where machine learning-based solutions come into play. [5] [7]
We will take a closer look at how exactly different models go about this task in section III

II. Approaches for Automation

This section will cover the different approaches for automation based on machine learning algorithms and will take a closer look at large language models used by thousands of software developers today. This paper will look at different metrics for each model but focus on three questions as far as data exists in these fields:

- How well can it detect refactoring opportunities? While also taking a closer look at false positives and false negatives.
- How well does the model refactor a given source code?
- How well can it detect refactoring opportunities and follow through with a potential refactor? Combining the two factors to determine how well it would do in real-world software development workflows.

A. Large Language Models

In the fields of ai assisted workflows, large language models (LLMs) have shifted from a niece speciality to an almost omnipotent tool which finds applications from image creation to code generation. [8] LLMs are huge deep-learning models pre-trained on enormous amounts of data. They are especially known for their ability to be trained on datatest of specific domains but can also be used on a broad spectrum of general knowledge, making them incredibly flexible. [2]

1) GPT Model: This section covers the experiments' results of the paper [2, AI-Driven Refactoring: A Pipeline for Identifying and Correcting Data Clumps in Git Repositories]. Being the best-known LLM, OpenAI's Generative Pre-trained Transformer series (GPT) with its versions GPT-3.5, GPT-3.5 Turbo and GPT-4. Temperature is a key parameter for GPT models, having a value ranging from zero to one, this parameter determines the predictability of the results. A higher temperature leads to more variety in the LLM and vice versa. Taking a look at detection itself, the median sensitivity of GPT-3.5-Turbo is 0 which indicates many data clumps are undetected and many false positives. Submitting all files in bulk also made the model trade off its sensitivity with the specificity parameters, with the median sensitivity reaching 50%, but the specificity only being 14%. Apparently, the model is looking at all the information and is finding more data clumps. But also potentially leading to more false positives. The temperature also has a similar trade-off. Higher temperatures lead to a

lower sensitivity and the other way around.
If you prompt a GPT model to refactor source code while also giving it the location of the data clumps, GPT-3.5 and GPT-4's median is identical, lying at 68%. These results show if you know where to look for data clumps and which places to refactor, both models can refactor the source code just as well as the other. GPT-3.5-Turbos arithmetic mean is less which can be explained by the existence of more overall compiler errors. On the same note: the median of the three instruction variants is also identical. Higher temperature values also resulted in a median of 0%, indicating more non-compilable code.
The final step of the experiment is combining detection and refactoring into one step. At this point, the limitations of GPT-3.5-Turbo become clear. The model scores a median score of 7% compared to 82% of GPT-4. Surprisingly, however, providing no definitions about data clumps leads to the best results, reaching a median of 46%. All other instruction types are 0% each. Another experiment, held in the paper mentioned in section I, [1, The effectiveness of supervised machine learning algorithms in predicting software refactoring], compared different machine learning models with each other, with the "Random Forest" Model performing best out of all the tested models.
It appears as if machine learning models commonly perform best in a close-to-random environment.

2) Github co-pilot: [4]

3) Fauxpilot Client:

B. Dedicated Models

1) DNNFFz:

III. Benefits of AI-Powered Refactoring

A. Improved Code Quality

B. Enhanced Maitainability

C. Reduction of Technical Dept

IV. Challenges and Limitations

A. Over-reliance on Automation

B. Potential for Unintended Consequences

C. Performance Concerns

V. Future Directions and Research Opportunites

A. Personalized Code Refactoring Suggestions

VI. Discussion

VII. Methodology

VIII. Conclusion and Outlook

References

[1] Mauricio Aniche, Erick Maziero, Rafael Durelli, and Vinicius HS Durelli. The effectiveness of supervised machine learning algorithms in predicting software refactoring. *IEEE Transactions on Software Engineering*, 48(4):1432–1450, 2020.

[2] Nils Baumgartner, Padma Iyenghar, Timo Schoemaker, and Elke Pulvermüller. Ai-driven refactoring: A pipeline for identifying and correcting data clumps in git repositories. *Electronics*, 13(9), 2024.

[3] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Signature Series (Fowler). Pearson Education, 2018.

[4] Saki Imai. Is github copilot a substitute for human pair-programming? an empirical study. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, pages 319–321, 2022.

[5] Y. Kataoka, T. Imai, H. Andou, and T. Fukaya. A quantitative evaluation of maintainability enhancement by refactoring. In *International Conference on Software Maintenance, 2002. Proceedings.*, pages 576–585, 2002.

[6] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. Technical debt: From metaphor to theory and practice. *Ieee software*, 29(6):18–21, 2012.

[7] R. Leitch and E. Stroulia. Assessing the maintainability benefits of design restructuring using dependency analysis. In *Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No.03EX717)*, pages 309–322, 2003.

[8] André Meyer-Vitali. Ai engineering for trust by design. In *Proceedings of the 12th International Conference on Model-Based Software and Systems Engineering-MBSE-AI Integration, Rome, Italy*, pages 21–23, 2024.