Scope description

Jason liu
Master Student Software Engineering
University of Antwerp
Antwerp, Belgium
jason.liu@student.uantwerpen.be

***Abstract*—Large Language Models (LLMs) have become the solution to many problems in today's software development landscape [1]. As readability remains a key factor in understanding how a codebase works, and with automated testing becoming increasingly popular, it is crucial that the identifiers we use are well-chosen and meaningful. This paper focuses on the problem of naming identifiers generated by automated tests, leveraging the capabilities of LLMs to improve the clarity and quality of these names.**

***Index Terms*—LLM, Testing, Readability, Identifiers, Code refactor**

I. Introduction

Naming identifiers is a crucial aspect of the readability and maintainability of codebases. Additionally, automated testing has become increasingly popular due to its high return on investment (ROI) compared to the effort required by a tester to implement a test. However, combining these two aspects presents a significant challenge: automated testing often lacks a deep understanding of what these identifiers actually mean.

This leads to additional problems. For new software developers, tests are often an easy way to understand the codebase by simply reading them. For long-time team members, the meaning of tests may gradually fade over time, making the tests less useful when errors occur—because we no longer clearly understand what the tests are meant to verify [2].

Well-defined identifiers have a significant impact on code quality and future development. However, determining what constitutes a well-defined identifier is not a trivial task. Even for humans, it can be difficult to decide whether an identifier is appropriately named. One developer might assign a name with a specific intent, while another might interpret it differently. This is why consistent coding conventions are crucial for maintainability and readability [3]. This is also where large language models (LLMs) have a major advantage: being trained on vast datasets allows them to adopt and follow a consistent naming convention, leading to more uniform and predictable interpretations.

II. Working title

"Naming identifiers generated by automated tests, using LLMs"

III. Research project 1 / Research project 2 / Thesis

This research will be conducted as a Master Thesis.

IV. The problem

In this project, we will explore how to leverage the power of Large Language Models (LLMs) to assign meaningful names to identifiers generated by automated test code. Automated test generation typically focuses on producing unit tests based on predefined requirements, often without adhering to proper code conventions. As a result, the generated tests may lack readability and be difficult for developers to understand. To address this issue, we will utilize LLMs trained to follow specific code conventions that we define, enabling them to generate well-structured and semantically meaningful identifiers.

A. Research questions?

Some research questions that might be interesting to keep in mind:

1) Does LLM perform better than existing ML renaming algorithms?
   - To assess the effectiveness of LLMs, a comparative analysis with existing machine learning techniques is necessary. Evaluating multiple dimensions of performance will allow us to determine whether the use of LLMs represents a meaningful improvement or an unjustified effort.
2) How do different prompt engineering strategies affect the quality and consistency of variable name suggestions generated by LLMs?
   - Working with LLMs allows us to experiment with different prompts to achieve varying results. To ensure we obtain the most optimal outcomes, it is essential to evaluate multiple prompt strategies and validate which one performs best. This process helps confirm whether our chosen approach is indeed the most effective.
3) How do developers perceive the readability, intent clarity, and maintainability of test code with LLM-generated variable names compared to traditional approaches?
   - This leads us to the central question of the project: does this approach genuinely improve code quality? Ultimately, our goal is to enhance the clarity and maintainability of automatically generated test code. Therefore, it is essential to understand how developers perceive the results—do they find the generated identifiers clearer and more helpful, or not?

4) Do LLM-generated variable names maintain consistent semantics across similar code scenarios within and across projects?
   - A previously mentioned advantage of LLMs is their ability to consistently apply a single coding convention, unlike human developers who may vary in style. However, it is important to investigate whether this consistency translates into tangible improvements in code readability, maintainability, or developer comprehension.
5) How does fine-tuning an LLM on test-specific code (e.g., unit tests from GitHub) affect naming quality compared to using a general-purpose code LLM?
   - Determining whether fine-tuning is required is critical, as it affects both performance and project cost. This evaluation helps managers and industry leaders make informed decisions when considering the adoption of LLM-based solutions.

V. Why is this a relevant / challenging problem?

As previously discussed, code quality is a fundamental aspect of reliable and maintainable software systems [4]. This issue becomes even more significant with the rise of automated testing, which, despite its benefits, often produces test cases that are difficult to interpret due to the absence of meaningful and consistent identifiers. As a result, the potential value of these tests can be diminished, particularly in collaborative and long-term development environments.

A key difficulty in this context stems from the large number of test cases typically generated by automated testing tools, making it a non-trivial task for developers to interpret and maintain them. To mitigate this, various automated naming algorithms have been proposed to improve test comprehensibility. Yet, these methods often produce suboptimal or inconsistent results. As a potential solution, we propose the use of Large Language Models (LLMs) to generate more meaningful and human-readable identifiers. However, this introduces additional complexity, as LLMs are not inherently optimized for this task. Therefore, careful model selection and configuration tuning are essential to realize their full potential in this domain.

VI. Why are you the one to solve it?

I am personally a strong advocate for code quality. This is one of the reasons why I enjoy working with software testing, as it naturally encourages developers to write cleaner, more maintainable code. However, when test code itself fails to meet standards of code quality—particularly in terms of readability and maintainability—it undermines the very goal it aims to support.

Another key motivation for this project is my passion for automation. I believe that automating improvements in code quality is not only efficient but also essential in modern software development practices.

Finally, I have a deep interest in Large Language Models (LLMs). I actively follow the latest developments in this field and possess a solid understanding of the capabilities and limitations of current-generation LLMs. This makes the intersection of code quality, automation, and LLMs an ideal and exciting area for me to explore.

VII. How will I solve it? In what direction?

To tackle this challenge, we propose the use of Large Language Models (LLMs) to generate more descriptive and human-readable identifier names in test code. Our methodology builds on existing literature in the field of identifier renaming within general codebases, which we will tailor and apply specifically to the domain of automated test generation.

VIII. Rough plan

1) Research Fundamentals
   - We will begin by conducting foundational research on the various aspects of the problem. This includes reviewing related work, particularly studies focused on identifier renaming in general codebases. By understanding existing approaches and methodologies, we can identify strategies that may be applicable or adaptable to test code.
2) Benchmark Setup
   - Before any implementation begins, we will establish a benchmark framework. This allows us to have a point of comparison, so that when we generate results later on, we can directly assess whether our approach performs better, worse, or comparably to existing methods.
3) Solution Exploration
   - In this phase, we will brainstorm and evaluate potential solutions. This involves analyzing which LLM to use as a baseline, identifying any existing tools or libraries that can assist us, selecting appropriate datasets, and determining how all these components can be effectively integrated. Planning and prioritization will also be key—deciding what is feasible within the scope and timeframe of the project.
4) Implementation
   - We will implement the proposed solution(s) based on our earlier planning. This includes integrating the selected model, applying it to test code, and developing any additional tooling required to support the pipeline.
5) Evaluation and Conclusion
   - In the final phase, we will apply the predefined metrics and use the established benchmark to evaluate the effectiveness of our approach. This comparison will help determine whether the LLM-based identifier naming provides a measurable improvement and offer insights into its practical value.

References

[1] T. Leppilampi, “Define and measure quality in AI-powered systems.” [Online]. Available: https://hiddentrail.com/blog/define-quality-in-ai-systems

[2] Test Smell Catalog, “Bad Naming — Violating Coding Best Practices.” [Online]. Available: https://test-smell-catalog.readthedocs.io/en/latest/Code%20related/Violating%20coding%20best%20practices/Bad%20Naming.html

[3] M. Cechetto, “Why Naming is Hard in Programming and Why LLMs Struggle Write Real Code.” [Online]. Available: https://dev.to/mateuscechetto/why-naming-is-hard-in-programming-and-why-llms-struggle-write-real-code-3bee

[4] SonarSource, “What is Code Quality? Definition Guide.” [Online]. Available: https://www.sonarsource.com/learn/code-quality/