Detection of Numerical Bugs

with Large Language Model (LLM)

Thesis advisors:

- David DEFOUR, Full Professor, Univ de Perpignan Via Domitia
- Eric PETIT, Research engineer, INTEL Portland USA
- Pablo DE OLIVEIRA, Full Professor, Univ. Versailles – Paris-Saclay

Location: University of Perpignan Via Domitia, France

Duration: 4-6 months

Keywords: IEEE754, floating-point computation, LLM

Compensation: ~630€/months

---

1. Introduction :

The advent of LLMs (Large Language Models) is generating a great interest in the research community for their potential to enhance productivity in numerous sciences and engineering. For example, GitHub Copilot, a model trained on natural language and a wide range of programs, helps developers write code faster and more efficiently by using automatic code completion and answering complex directives in natural language (prompt). However, this potential remains to be explored/demonstrated in many challenging areas, such as writing programs that manipulate floating-point numbers due to the specificity of these numbers and the issues related to their use.

The objective of this internship is to assess the usefulness of LLMs for static detection (without executing the program) of numerical bugs during code writing. It is worth noting that this internship is part of a significant project funded for the period 2025-2029 by the National Research Agency (ANR project Floating-Point Transformer 4), bringing together academic and industrial partners (University of Perpignan Via Domitia, University of Versailles – Paris-Saclay, Sorbonne, EDF, CEA, ANEO, Intel). A continuation through a PhD is possible.

2. Floating-point arithmetic

The IEEE-754 [6] standard is the cornerstone of floating-point number representation in modern computer systems, ensuring consistency and precision in numerical calculations across various hardware and software platforms. Universally adopted, this standard defines several formats for representing real numbers, the most used being single precision (32 bits) and double precision (64 bits). These numbers are represented by three fields: the sign, which determines whether the number is positive or negative, the biased exponent, which represents the range of possible values, and the mantissa, which represents the fractional part of the number.

Manipulating these numbers requires a thorough understanding of the entire software stack (hardware operators for calculation, memory hierarchy for storage, buses and networks for data

transfer, languages, OS, and compilers for interpretation, and developers/users for meta-knowledge). However, using this type of representation is subject to various types of problems with varying degrees of severity. The most common issues are capacity overflows when a result can no longer be represented (e.g., overflow/underflow), the appearance of undefined values like Not A Number (e.g., calculating the square root of a negative number), division by zero, and precision losses (e.g., catastrophic cancellation or accumulation of rounding errors). These problems can lead to dramatic consequences (e.g., Ariane 5, Patriot Missile, Vancouver Stock Exchange Index[1]).

Today, these issues can be detected at the end of the process through static or dynamic code analysis. Static analysis involves examining the source code without requiring its execution and focuses on identifying errors or coding standard violations. It allows for early problem detection, reducing development time and costs. However, this type of analysis may generate false positives or false negatives, affecting the quality of the analysis. On the other hand, dynamic analysis requires code execution and evaluates its behavior in real-time on one or more test data sets, but it is often incomplete (examples of tools: CADNA[2], FPChecker**Erreur ! Source du renvoi introuvable.**, Verificarlo[3], FramaC[5]).

3. LLM

Large Language Models (LLMs) are large-scale machine learning models designed to perform various complex natural language processing tasks. These tasks include text analysis and generation, conversational question answering, and translating text between different languages. LLMs are distinguished by their massive size, containing billions of parameters, allowing them to capture linguistic and contextual nuances. Generally, the more parameters a model has, the more likely it is to provide accurate and reliable answers across a wide range of topics. For comparison, OpenAI's first model, GPT-1, had 0.12 billion parameters, while GPT-4 is assumed to have a trillion parameters. Their success is partly based on an advanced transformer architecture, which allows LLMs to understand and generate text fluidly and contextually. Models like CodeBERT and CodeT5 are particularly suited for code-specific applications, such as code synthesis, generation, and similarity analysis.

Several methods exist to improve the relevance of the results produced by these tools. For example, *prompt engineering* involves designing precise and relevant textual instructions, *fine-tuning* adjusts a pre-trained model with new data for a specific domain, and *RAG* (Retrieval-Augmented Generation) enriches queries with information retrieved from external sources. Each of these solutions has its advantages and disadvantages. Fine-tuning is generally more suitable for stylistic adjustments but has some drawbacks (e.g., susceptibility to forgetting due to overtraining or altering the importance of information located at the core of its context window), while RAG is better suited for knowledge enhancement but comes at the cost of more complex prompts, leading to higher processing costs or scalability issues. Lastly integrating one or multiple Llm in an agentic system, such as langchain, and enabling reflection and reasoning strategies like chain of thought or self-iterating, have proven to bring significant progress in complex task at the expense of time complexity, in math and coding challenges.

4. Methodology

The candidate will need to familiarize themselves with a test program base (InterFLOPBench Benchmarks) used to evaluate the capabilities of numerical analysis tools (small test programs containing various bugs) to assess the behavior of different LLMs (general-purpose or specialized, and of varying sizes).

To achieve this, the candidate will:

1. Set up a software infrastructure based on **Ollama** to evaluate LLMs on these programs, first on a test machine and then on a production machine (e.g., Jean-Zay).
2. Make the necessary modifications to **InterFLOPBench** to automate the analysis of responses (e.g., adding metadata and scripts).
3. Propose metrics to evaluate the quality of the LLMs' responses.
4. Assess the value of different solutions for improving relevance, considering **RAG** first and then **LangChain**.

5. Application

The candidate should demonstrate skills or, at a minimum, a strong interest in software development, AI, and, more broadly, Large Language Models and computer arithmetic.

Motivated candidates should send their CV and transcript to:

david.defour@univ-perp.fr, eric.petit@intel.com, pablo.oliveira@uvsq.fr

6. References

[1] https://sites.math.rutgers.edu/~sg1108/Math373/Matherrors.html
[2] https://cadna.lip6.fr/index.php
[3] https://github.com/verificarlo
[4] https://github.com/LLNL/FPChecker
[5] https://frama-c.com/
[6] https://fr.wikipedia.org/wiki/IEEE_754