LLM-assisted Methodology for Embedded Software Performance Estimation on RISC-V *

Weiyan Zhang[1] Muhammad Hassan[1,2] Rolf Drechsler[1,2]
[1]Institute of Computer Science, University of Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, Germany
{weiyan, hassan, drechsler}@uni-bremen.de

Abstract

In this extended abstract, we present a methodology that combines a Large Language Model *(LLM) with a traditional* Machine Learning *(ML) technique to estimate the performance of embedded software on RISC-V processors across different microarchitectures. In particular, we leverage a* Retrieval-Augmented Generation *(RAG)-based LLM to extract performance-related information from processor specifications and source code, while utilizing the predictive capabilities of ML models to create* Predictive Models *(PMs) for RISC-V processors. To demonstrate the effectiveness of our hybrid approach, we present results on the performance estimation of open-source benchmarks using the generated PMs, with open-source RISC-V-based* Register Transfer Level *(RTL) implementations as reference models. Our results demonstrate that our proposed LLM-assisted methodology provides highly accurate predictions in comparison with the state-of-the-art methodology.*

Introduction

Embedded systems are becoming increasingly important in automation and the *Internet of Things* (IoT) as silicon technology advances. RISC-V has gained popularity due to its open-source architecture, enabling customization while reducing costs. Its comprehensive ecosystem, including functional simulators and *Register Transfer Level* (RTL) implementations, facilitates both high-level customization and low-level optimization.

Accurate cycle-count estimation is essential but challenging due to increasing system complexity. Traditional RTL simulations, while precise, are slow, creating trade-offs between accuracy and speed.

Recent advances in *Large Language Models* (LLMs), like OpenAI's GPT series and DeepSeek, enable applications in natural language processing and code generation. *Retrieval-Augmented Generation* (RAG) improves LLMs by leveraging external data for performance analysis, facilitating the creation of *Predictive Models* (PMs) and accelerating workflows. However, LLMs have limitations when handling tasks that require precise calculations or numerical reasoning [1], which can be addressed by integrating Machine Learning (ML) techniques like *Artificial Neural Networks* (ANNs) to improve accuracy for complex tasks.

In this abstract, we propose an LLM-assisted methodology to extract performance information for RISC-V processors and ML models to train PMs. We evaluated the PMs on 10 benchmarks, measuring error percentages and calculating the overall *Mean Absolute Percentage Error* (MAPE) across tests.

*This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within projects Scale4Edge under grant no. 16ME0127, and EXCL under grant no. 01IW22002.

Methodologies

Our performance estimation methodology, illustrated in Fig. 1, integrates an LLM and an ANN in four phases: text generation, data preprocessing and RAG, model training, and model testing.

1. **Text Generation:** An LLM identifies performance-related parameters like divider_latency based on a prompt, generating a list for the next phase.
2. **Data Preprocessing & RAG:** Documentation and source code are stored in an SQL database, split into chunks, and embedded into vectors. Queries retrieve and rank relevant chunks, extracting parameter values to generate separate lists for documentation and source code.
3. **Model Training:** Programs are executed on a functional simulator and RTL implementation to obtain dynamic instruction counts and reference clock cycles. A conflict-resolution solver addresses discrepancies between values extracted from documentation and source code, prioritizing the source code as more reliable. Using these inputs, an ANN is trained to model the relationship between parameters, instruction counts, and clock cycles, enabling accurate performance predictions.
4. **Model Testing:** The trained PM is tested on standard benchmarks. Dynamic instruction counts from the simulator and performance-related parameters from the LLM serve as inputs to the PM. Predicted clock cycles are compared to actual execution cycles from the reference to assess accuracy and effectiveness.

Evaluation

This work leverages gpt-3.5-turbo-0125 [2] via OpenAI's API. Text was split using CharacterTextSplitter,

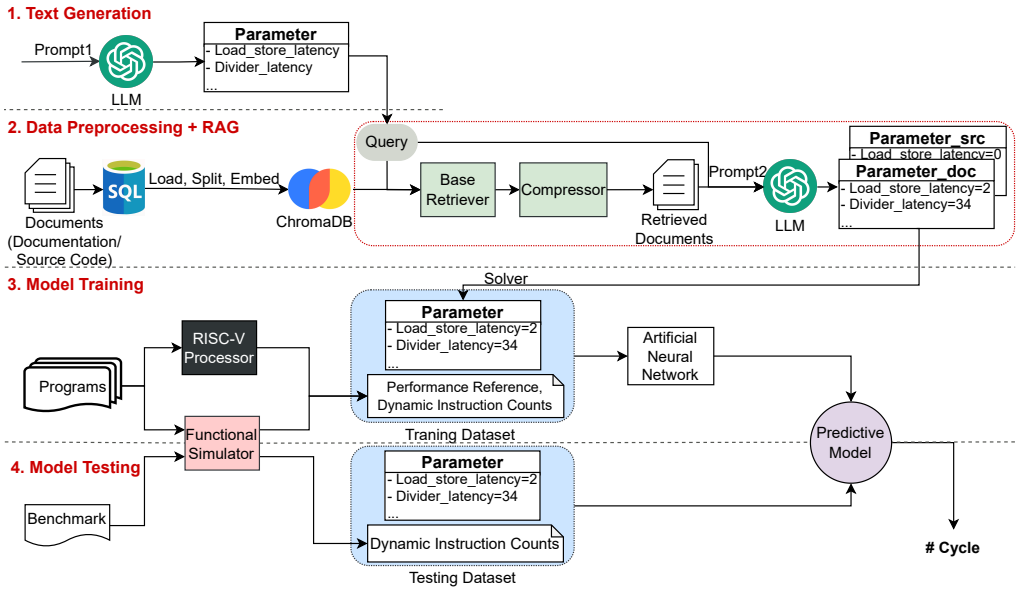

Figure 1: *Overview of the proposed LLM-assisted methodology for performance estimation of embedded software.*

Table 1: *Experimental Results of all Benchmarks used for Validation of PMs on RV32I*

benchmark	# instr-exec.	SweRV # Cycle	PM1 APE	RSD # Cycle	PM2 APE
adpcm_dec	2 880 766	3 644 041	0.21%	5 260 228	0.87%
adpcm_enc	2 898 909	3 224 628	13.59%	5 232 514	0.24%
cubic	28 338 773	34 071 398	1.34%	64221227	23.68%
deg2rad	510 731	573 745	2.51%	872 615	1.12%
fft	3 678 522	5 420 220	1.15%	4 010 734	43.26%
gsm_dec	9 168 156	14 387 983	2.50%	12 076 100	30.08%
isqrt	1 002 078	3 125 021	1.30%	1 220 580	0.72%
lms	5 814 943	7 063 929	0.79%	10 253 500	1.04%
rad2deg	420 103	482 789	0.51%	627 494	10.17%
st	3 684 066	4 445 458	1.05%	5 842 673	7.88%
MAPE			2.50%		11.90%

embedded as vectors via all-MiniLM-L6-v2, and stored in a Chroma database for efficient retrieval. Relevant chunks were identified through a retriever and reranked using FlashrankRerank, enabling precise parameter extraction. These tools were accessed using LangChain framework's APIs [3].

For model training, around 700 programs were generated using custom samples and TACLeBench benchmarks. Programs were compiled with the RISC-V GNU toolchain [4] and executed on SweRV [5] and RSD [6] RTL implementations to collect reference cycles and on Whisper [7] simulator to collect the dynamic instruction counts. An ANN, implemented in TensorFlow, trained on the collected data with hyperparameters tuned via random search. For model testing, 10 benchmarks distinct from the training set from TACLeBench were selected.

Table 1 shows the results, including the number of executed instructions, cycles, and APE for each benchmark. PM1 achieved a MAPE of 2.50%, outperforming PM2 (11.90%). Compared to the state-of-the-art method in [8], which reported higher MAPE for SweRV (6.5%) and RSD (18.6%), our methodology reduced MAPE by 61.54% for SweRV and 36.02% for RSD, demonstrating significant improvements in accuracy.

Conclusion and Future Work

This abstract presents a methodology for estimating the performance of embedded software on RISC-V processors by combining an LLM and an ANN. LLM extracts performance parameters from source code and documentation, while ANN predicts performance based on these parameters and dynamic instruction counts.

The approach demonstrates improved accuracy and efficiency over existing methods, showcasing its potential for hardware-software co-design. Future work will address challenges like handling large codebases, extend the method to more complex architectures, and explore alternative models to further enhance scalability and applicability.

References

[1] Janice Ahn et al. "Large language models for mathematical reasoning: Progresses and challenges". In: *arXiv preprint arXiv:2402.00157* (2024).

[2] *GPT-3.5*. `https://platform.openai.com/docs/models/gpt-3-5-turbo`.

[3] *LangChain*. `https://www.langchain.com/`.

[4] *RISC-V GNU Compiler Toolchain*. `https://github.com/riscv-collab/riscv-gnu-toolchain`.

[5] Western Digital. *SweRV RISC-V Core*. `https://github.com/chipsalliance/Cores-VeeR-EH1/tree/1.0`.

[6] Susumu Mashimo et al. "An open source FPGA-optimized out-of-order RISC-V soft processor". In: *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE. 2019, pp. 63–71.

[7] CHIPS Alliance. *Whisper*. `https://github.com/chipsalliance/VeeR-ISS`.

[8] Weiyan Zhang et al. "Efficient ML-based performance estimation approach across different microarchitectures for RISC-V processors". In: *2023 26th Euromicro Conference on Digital System Design (DSD)*. IEEE. 2023, pp. 693–699.