Variable Name Reconstruction for Decompilers using Large Language Models and Program Analysis

Sergio D. Rodríguez De Jesús Rafael Arce Nazario

Department of Computer Science, University of Puerto Rico at Rio Piedras

Introduction

Reverse engineers rely on decompilers for effective binary analysis. While contemporary decompilers excel in many aspects, a persistent challenge lies in the recovery of variable and function names. Typically, these identifiers are discarded during compilation, resulting in decompilers outputting names devoid of semantic meaning—often mere representations of memory addresses. Addressing this limitation, Xu et al. [3] introduced LmPa, a system that employs the synergy between language models and program analysis to reconstruct variable names.

However, it's notable that LmPa [3] utilizes GPT-3.5, and there is an observable trend toward employing larger models for improved outcomes. In this context, we propose a hypothesis: by employing smaller, more specifically trained models, comparable results can be achieved, thereby enhancing accessibility and minimizing costs. We aim to implement the strategy introduced with LmPa [3], but utilizing lower-cost alternatives like Mixtral 8x7B and CodeLlama instead of GPT-3.5.

Methodology

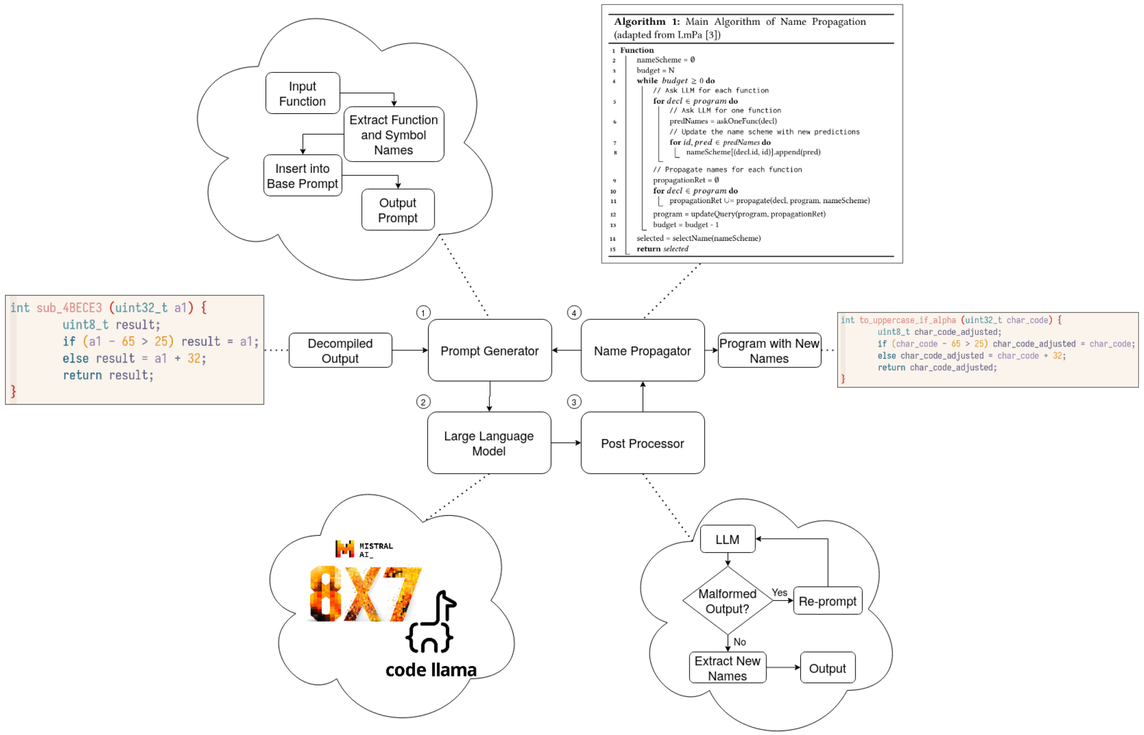

Figure 1. Flowchart of the methodology.

Our proposed architecture, adapted from LmPa [3], comprises four key components. Firstly, a prompt generator extracts relevant data from decompiled output, generating a series of prompts. Subsequently, a Language Model (LLM) processes these prompts, attempting to produce variable names. A post-processor ensures the validity of the LLM's output. Finally, a name propagator enriches the context for the LLM, contributing to more accurate reconstructions. We expect better results as we repeat these four steps, making the approach iterative in nature.

Our project is in its early stages and so we are only running tests of this system manually. Our focus has been on testing out different LLMs and crafting careful prompts in order to get what we need. After some testing, we settled on comparing Mixtral-8x7B-Instruct-v0.1 [2] and CodeLlama-34b-Instruct-hf [1]. For context, GPT-3.5 has 175 billion parameters, while Mixtral-8x7B-Instruct-v0.1 [2] has a total of 56 billion (but uses 13 billion at inference time), and CodeLlama-34b-Instruct-hf [1] has 34 billion parameters.

Preliminary Results

These preliminary results are based on a single-pass approach where the decompiled output gets manually inserted into a prompt and fed into an LLM. The prompts used tend to have a big impact on the results. Base Prompt 1 emphazises giving context about the role of the LLM and what it is used for, as well as specifying the required output format. Base Prompt 2 expands on this by using prompting techniques like providing example inputs and outputs. Examples of both prompts are available at: bit.ly/llmdecompgist

Model	Decompiled Symbols	Expected Symbols	Output
mistralai/Mixtral-8x7B-Instruct-v0.1	a1, result, sub_4BECE3	N/A	a1 -> char_code result -> char_code_adjusted sub_4BECE3 -> to_uppercase_if_alpha
codellama/CodeLlama-34b-Instruct-hf	a1, result, sub_4BECE3	N/A	Completes the prompt, doesn't answer properly.
mistralai/Mixtral-8x7B-Instruct-v0.1	param_1, param_2, param_3, cVar1, iVar2, lVar3	param_1 -> pParse, param_2 -> pTab, param_3 -> pTrigger	param_1 -> db param_2 -> pTab param_3 -> flags cVar1 -> isVtab iVar2 -> i lVar3 -> vtab
codellama/CodeLlama-34b-Instruct-hf	aram_1, param_2, param_3, cVar1, iVar2, lVar3	param_1 -> pParse, param_2 -> pTab, param_3 -> pTrigger	Completes the prompt, doesn't answer properly.

Figure 2. Table of Preliminary results with Base Prompt 1.

Model	Decompiled Symbols	Expected Symbols	Output
mistralai/Mixtral-8x7B-Instruct-v0.1	param_1, param_2, param_3, cVar1, iVar2, lVar3	param_1 -> pParse, param_2 -> pTab, param_3 -> pTrigger	param_1 -> db_ptr param_2 -> table_name param_3 -> flags cVar1 -> table_status iVar2 -> error_code lVar3 -> vtable_ptr
codellama/CodeLlama-34b-Instruct-hf	aram_1, param_2, param_3, cVar1, iVar2, lVar3	param_1 -> pParse, param_2 -> pTab, param_3 -> pTrigger	param_1 -> db param_2 -> table_name param_3 -> p_trigger cVar1 -> is_virtual_table iVar2 -> is_read_only lVar3 -> vtable_ptr

Figure 3. Table of Preliminary results with Base Prompt 2.

Future Works

Our primary focus moving forward is to continue experimenting in comparing LLMs until we can be confident to choose one. In addition, it is critical to implement the name propagation algorithm proposed by Xu et al. [3]. A key part of this process is to use a wide variety of functions, so we can test the consistency of the system, and its performance with increasingly complex inputs. After all of this, we will focus on automating the pipeline in order to have a seamless system.

References and Acknowledgements

We want to thank ACCESS for allowing us to use resources from Bridges2 PSC.

[1] CodeLlama.
codellama/codellama-34b-instruct-hf.
`https://huggingface.co/codellama/CodeLlama-34b-Instruct-hf`.

[2] MistralAI.
mistralai/mixtral-8x7b-instruct-v0.1.
`https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1`.

[3] Xiangzhe Xu, Zhuo Zhang, Shiwei Feng, Yapeng Ye, Zian Su, Nan Jiang, Siyuan Cheng, Lin Tan, and Xiangyu Zhang.
Lmpa: Improving decompilation by synergy of large language model and program analysis.
arXiv preprint arXiv:2306.02546, 2023.