RevLlama: Recovering Function Names via Rationale Distillation from Large to Small Language Models

Minami Someya[1], **Akira Otsuka**[1]

[1]Institute of Information Security
2-14-1 Tsuruya-cho, Yokohama, Kanagawa 221-0835, Japan
someya@ai.iisec.ac.jp, otsuka@iisec.ac.jp

Abstract

Binary reverse engineering plays a crucial role in security tasks such as malware analysis and vulnerability discovery, uncovering program behavior without access to source code. Function names provide key insights into code, but these names are often stripped during compilation, making their recovery a considerable challenge even for experienced analysts. Large Language Models (LLMs), such as ChatGPT, have demonstrated strong code comprehension capabilities; however, their dependence on cloud-based environments renders them unsuitable for sensitive tasks such as malware analysis. This limitation underscores the need for Small Language Models (SLMs) that are both computationally efficient and effective in local environments. In light of this challenge, we propose RevLlama, an SLM specifically designed for binary code analysis. Built on Llama-3.1-8B, RevLlama leverages a knowledge distillation approach to transfer the step-by-step reasoning process from an LLM to an SLM. Our method first employs an LLM to extract the reasoning rationale for function name estimation from decompiled code. Then, both the extracted rationale and the function names are used to efficiently train an SLM. Evaluation results demonstrate that RevLlama outperforms existing methods in the function name recovery task. Specifically, RevLlama achieved an F1 score of 0.587 in function name estimation—a 20.3% improvement over GPT-4o in a local environment.

Code — https://github.com/som3ya/RevLlama

Introduction

Reverse engineering of binary programs is fundamental to software security analysis, enabling critical security tasks including malware investigation, vulnerability discovery, and software protection assessment. During analysis, security researchers decompile binary programs to examine their underlying behavior, a process that requires sophisticated technical expertise and tooling. For instance, in advanced malware analysis, researchers not only identify malicious behaviors and communication destinations but also attribute attacks through distinctive code characteristics, making reverse engineering an indispensable methodology in modern cybersecurity practices.


```
unsigned int __cdecl read32(const uint8_t **ptr, int is_big)
{
unsigned __int32 value32;

if ( is_big )
value32 = _byteswap_ulong(*(_DWORD *)*ptr);
else
value32 = *(_DWORD *)*ptr;
*ptr += 4;
return value32;
}
```

(a) Source code of `read32` function

```
__int64 __fastcall sub40B012F(unsigned int **a1, int a2)
{
unsigned __int32 v3; // [rsp+18h] [rbp-4h]

if ( a2 )
v3 = sub40A065(**a1);
else
v3 = **a1;
++*a1;
return v3;
}
```

(b) Decompiled code of `read32` function

Figure 1: Comparison between source code and decompiled code. During compilation, symbolic information such as function and variable names is lost, making decompiled code difficult to understand.

A key challenge in binary analysis is recovering the semantic information lost during compilation. Function names, which provide crucial contextual clues for program comprehension,are especially important in addressing this challenge. As illustrated in Figure 1, the compilation process strips away these semantic identifiers, forcing decompilers to generate arbitrary symbols (e.g., `sub40A065`). The absence of meaningful identifiers, such as `_byteswap_ulong` that explicitly indicate specific operations, further complicates program understanding and prolongs the analysis process.

Recent advances in Large Language Models (LLMs) have opened new possibilities for addressing the function name recovery challenge. Traditional machine learning approaches (Jin et al. 2022; David, Alon, and Yahav 2020; Xiong et al. 2023) relied on models with limited capacity that struggled to handle complex code contexts. In contrast, state-of-the-art LLMs, such as GPT-4 (OpenAI et al. 2024) and Gemini (Team et al. 2024), demonstrate advanced

program comprehension capabilities. Notably, Chain-of-Thought (CoT) reasoning (Wei et al. 2022) has shown exceptional performance in complex reasoning tasks, suggesting its potential to enhance function name recovery in binary analysis.

Despite these advances, applying LLMs to binary analysis still faces considerable constraints. Cloud-based LLMs, including ChatGPT, require transmitting user data to external servers, which poses risks of information leakage. In malware analysis, offline environments are standard practice to prevent malware propagation[1], rendering cloud-based solutions impractical. Consequently, there is a need for high-performance, compact models capable of operating effectively in local environments.

Knowledge distillation techniques for transferring capabilities from LLMs to Small Language Models (SLMs) have garnered attention in natural language processing (Hsieh et al. 2023). Knowledge distillation (Hinton, Vinyals, and Dean 2015) efficiently transfers the knowledge and reasoning abilities of a large model to a smaller one, thereby facilitating the development of lightweight models. However, its effectiveness in highly specialized tasks, such as reverse engineering, remains insufficiently validated.

Our Approach In this work, we propose RevLlama, which leverages knowledge distillation from an LLM for function name prediction in binary code. Based on Llama-3.1-8B, RevLlama achieves high-accuracy function name prediction in local environments by distilling the reasoning rationale from a large model into a smaller one. In our evaluation experiments, RevLlama achieved an F1 score of 0.587, realizing a 20.3% performance improvement over GPT-4o.

Contributions Our main contributions are as follows:

- We demonstrate the effectiveness of rationale distillation for function name recovery by transferring the reasoning rationale from an LLM to an SLM. To the best of our knowledge, this is the first work to apply knowledge distillation to reverse engineering.
- We develop RevLlama, a small language model using knowledge distillation, which achieves a 20.3% improvement over GPT-4o while enabling local execution.
- We release our research artifacts[2]. These include models, code, and a dataset containing reasoning rationales. These resources will contribute to advancing LLM research in binary analysis.

Related Work

Machine learning methods for reverse engineering address a range of tasks, such as variable name recovery (Chen et al. 2022; Xiong et al. 2023), function name recovery (Jin et al. 2022; David, Alon, and Yahav 2020), and code summarization (Al-Kaswan et al. 2023; Xiong et al. 2023). Among these, function name recovery poses a unique challenge because it requires advanced natural language understanding to concisely summarize program behavior. The absence of symbolic information and the intricate structure of decompiled code render this task far more difficult than analyzing source code.

Prior research has primarily employed approaches based on the Transformer (Vaswani et al. 2017) architecture, which has proven successful in natural language processing. For example, Nero by David et al. (David, Alon, and Yahav 2020) exploited control flow graph structural information, while SymLM by Jin et al. (Jin et al. 2022) enhanced function name recovery by incorporating function call contexts. Moreover, HexT5 by Xiong et al. (Xiong et al. 2023) introduced a multi-task model for function name recovery by retraining CodeT5 (Wang et al. 2021) on decompiled code. However, these methods rely on relatively small models, which limits their ability to capture complex code contexts.

More recent research has explored the use of LLMs for function name recovery. Shang et al. (Shang et al. 2024) demonstrated that LLMs offer superior generalization performance compared to traditional deep learning techniques in binary code understanding. Nevertheless, high-performance LLMs are typically available as cloud-based services, which restricts their use in security-critical scenarios such as malware analysis. Furthermore, these studies have not specifically addressed methods to improve the function name recovery capabilities of LLMs.

Building on these prior works, our research aims to develop a lightweight, high-performance SLM through knowledge distillation. In particular, we transfer complex reasoning capabilities from an LLM to an SLM by explicitly leveraging the reasoning processes generated by the LLM.

Method

We construct a high-performance function name prediction model by efficiently transferring knowledge from an LLM to an SLM. This section first presents an overview of our approach, followed by detailed explanations of each component.

Overview

Our approach specializes the concept of *Distilling Step by Step* (Hsieh et al. 2023) for the function name prediction task. As shown in Figure 2, this method consists of two main steps:

1. **Rationale Extraction**: An LLM is employed to generate the thought process (i.e., reasoning rationale) during function name prediction from decompiled code. Chain-of-Thought prompting is utilized to encourage step-by-step reasoning.
2. **Knowledge Transfer**: An SLM is fine-tuned via multi-task learning using the extracted reasoning rationale and function names. This process enables the efficient transfer of reasoning capabilities from the LLM to the smaller model.

[1]Malware analysis typically employs isolated virtual machines with host-only networking to minimize risks of infection and data exfiltration. Even local network access is often restricted.

[2]https://github.com/som3ya/RevLlama

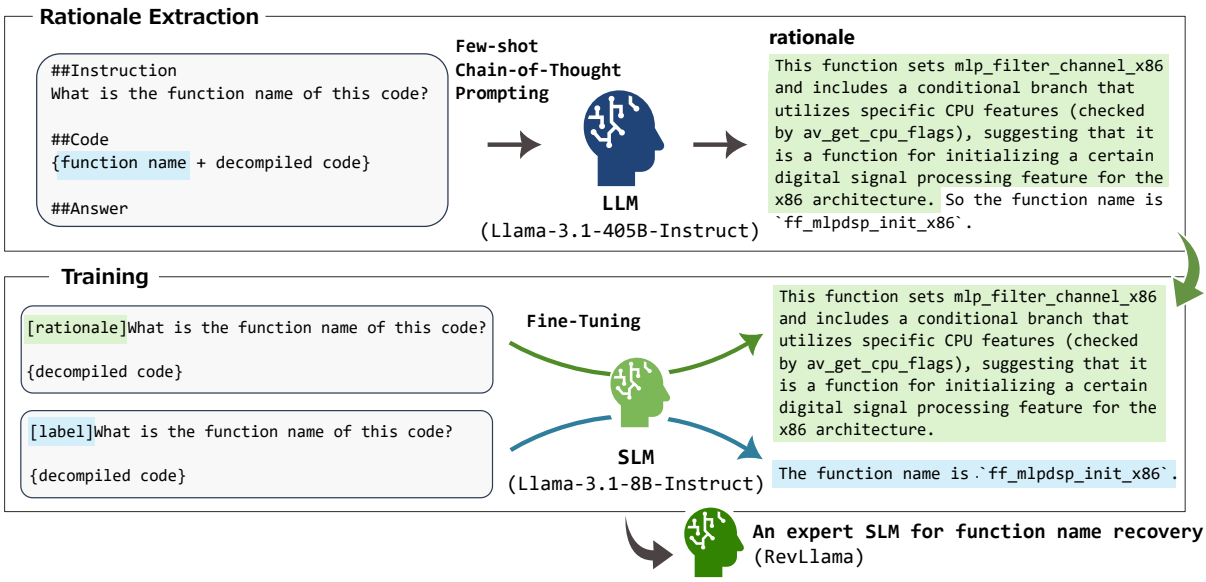

Figure 2: Overview of the RevLlama construction method. The reasoning rationale for function name prediction is extracted from a teacher model (LLM) and transferred to a student model (SLM).

This approach facilitates the development of an SLM that inherits the sophisticated reasoning capabilities of an LLM for function name prediction.

Rationale Extraction

This phase captures the reasoning process of the LLM during function name prediction. For each decompiled function, the model generates natural language explanations that analyze key elements such as variable usage, control flow structures, and algorithmic patterns to justify its predictions.

The prompt construction process employs a labeled dataset $(x_i, y_i) \in D$, where each prompt comprises three components: decompiled code x^{p}, reasoning rationale r^{p}, and the corresponding function name y^{p}. The framework uses few-shot prompting (Brown et al. 2020) with chain-of-thought (Wei et al. 2022) examples to guide the LLM in its reasoning. The output format follows the instruction: `"Please give the function name after the rationale"`.

During this phase, the ground truth function names y_i, derived from the source code, are provided. These source-level function names serve as reliable references to ensure that the generated reasoning aligns with the actual semantics of the functions. By using precise source-level function names as ground truth, the framework ensures high-quality reasoning and accurate function name prediction.

The LLM outputs are processed by removing final function name declarations (for example, `The function name is 'xxx'`) to isolate the pure reasoning process. These rationales serve as training signals for the knowledge transfer phase, providing detailed insights into function behavior and purpose.

Knowledge Transfer

Training of the student SLM is performed using multitask learning on the processed dataset $(x_i, y_i, r_i) \in D$. The model $f(x_i) \rightarrow (\hat{y}_i, \hat{r}_i)$ is trained to generate both function names $\hat{y}_i$ and reasoning rationales $\hat{r}_i$ for each input x_i.

The training objective combines the tasks of function name prediction and rationale generation. The total loss $\mathcal{L}$ is defined as:

$$\mathcal{L} = \mathcal{L}_{\text{label}} + \mathcal{L}_{\text{rationale}}$$

$$\mathcal{L}_{\text{label}} = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i), \hat{y}_i),$$

$$\mathcal{L}_{\text{rationale}} = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i), \hat{r}_i)$$

Task identifiers (`[label]` or `[rationale]`) are prepended to the input during training, enabling the model to learn the appropriate output format for each task.

Inference

During inference, the `[label]` identifier is prepended to the input decompiled code. The model then generates function names in a tuned format (`The function name is 'xxx'`), ensuring compatibility with established reverse engineering workflows.

Evaluation

We evaluate the effectiveness of RevLlama, our proposed function name prediction model, by conducting two experiments that address the following research questions (RQs):

RQ1: How effective is RevLlama's knowledge distillation approach for function name recovery?
RQ2: How does RevLlama's performance compare to existing approaches and state-of-the-art LLM?

Experimental Setup

This section outlines the experimental settings shared across both experiments.

Implementation We used Llama-3.1-405B-Instruct[3] as the teacher LLM, one of the largest publicly available models with permissive licensing terms. We employed Llama-3.1-8B-Instruct[4] as the student SLM. Our framework can be adapted to other model architectures.

The experiments were conducted on a system running 64-bit Ubuntu 22.04, equipped with an Intel(R) Xeon(R) CPU @ 2.20GHz and an NVIDIA A100-SXM4-40GB GPU. For the inference (evaluation) phase, we utilized vllm (Kwon et al. 2023) to accelerate the generation process. To ensure reproducibility, we set the temperature parameter to 0 and used greedy decoding, minimizing probabilistic variations in model outputs.

Rationale Extraction For rationale extraction from the teacher model, we used Llama-3.1-405B-Instruct via the Fireworks AI[5] API. As shown in Figure 3, we guide the model's thinking process for function name prediction using few-shot prompting. Specifically, we provided pairs of decompiled code and function names of the source code, prompting the model to explain why each function name was appropriate. To facilitate accurate rationale generation, function names were left unmasked in the decompiled code, enabling the model to reference the source code function names. The reasoning rationales were extracted by removing the final sentence of the output, which contained the function name declaration.

Fine-Tuning For fine-tuning the student model, we employed LoRA (Hu et al. 2022) and organized the training data using the `[label]` and `[rationale]` templates. Based on preliminary experiments, we set the number of epochs to 1 and the learning rate to 1×10^{-4}. The LoRA hyperparameters were configured with a rank of $r = 32$ and a scaling coefficient of $\alpha = 16$. Losses were computed solely on the output portion, and after fine-tuning, the LoRA weights were merged into the base model to facilitate efficient inference using vllm.

Evaluation Metrics

To comprehensively assess our method, we employ two types of evaluation metrics. Traditional word-matching metrics, commonly used in previous studies, are limited in evaluating expressions that convey similar meanings. For example, function names such as `get_value` and `fetch_data` convey similar meanings yet may receive a score of zero due to the absence of overlapping words. To address this limitation, we supplement conventional word-matching metrics with a metric based on Sentence-BERT, which captures semantic similarities between function names.

[3]https://huggingface.co/meta-llama/Llama-3.1-405B-Instruct
[4]https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct
[5]https://fireworks.ai/

```
### Instruction
What is the function name of this code? After stating the rationale,
answer the function name in the last sentence.

### Code
unsigned __int64 __fastcall notifier_list_notify(__int64 *a1)
{
__int64 i; // [rsp+18h] [rbp-18h]
__int64 v3; // [rsp+20h] [rbp-10h]
unsigned __int64 v4; // [rsp+28h] [rbp-8h]

v4 = __readfsqword(0x28u);
for ( i = *a1; i; i = v3 )
{
v3 = *(_QWORD *)(i + 8);
(*(void (__fastcall **)(__int64))i)(i);
}
return __readfsqword(0x28u) ^ v4;
}

### Answer
This function iterates through a list of notifier callbacks, calling
each one in sequence. The loop starts with the pointer i set to the head
of the list (provided by a1), and each iteration sets i to the next entry
in the list, calling the function pointer at i in the process. It includes
a stack canary check to detect stack corruption by comparing the initial
and final values of __readfsqword(0x28u).
So The function name is `notifier_list_notify`.
```

Figure 3: Demonstration examples for rationale extraction. Each example includes decompiled code, correct function name, and reasoning process.

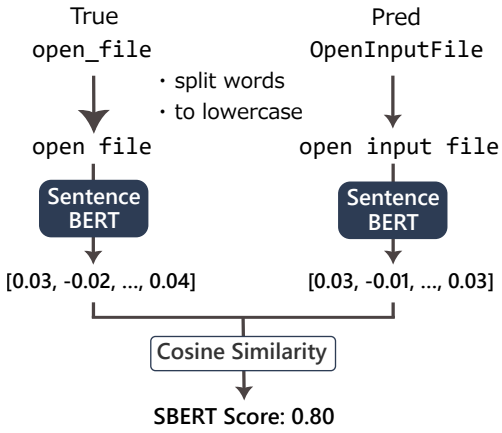

Figure 4: Calculation method for Sentence-BERT scores.

Word-Level Matching Following previous studies (Jin et al. 2022; Xiong et al. 2023), we compute word-level precision, recall, and F1-score between the predicted and ground truth function names. Function names are first segmented into words based on three common naming conventions: `CamelCase`, `snake_case`, and `kebab-case`, and then normalized to lowercase. For example, `getUserValue` is segmented into the words `get`, `user`, and `value`. The scores for each sample are computed based on these word sets, and the final metric is obtained by averaging the scores across all samples.

Sentence-BERT To evaluate semantic similarities between function names, we employ a metric based on Sentence-BERT (Reimers and Gurevych 2019). As illus-

Table 1: Performance of function name prediction.

	SBERT	Precision	Recall	F1
base_model	0.391	0.353	0.279	0.312
FT with label	0.474	0.390	0.365	0.377
FT with CoT	0.494	0.414	0.396	0.405
RevLlama	**0.528**	**0.454**	**0.442**	**0.448**

Top scores for each metric are shown in **bold**.

trated in Figure 4, function names are first segmented into words using the same method as in the word-level matching. These word sequences are then converted into 768-dimensional vectors using the all-mpnet-base-v2 model[6], which captures semantic relationships between words. Finally, semantic similarity is quantified by computing the cosine similarity between the predicted and ground truth vectors, and the scores are averaged across all samples.

Exp. 1: Effect of Learning with Reasoning Rationale

We evaluate the effectiveness of incorporating reasoning rationale by comparing the performance of RevLlama against baseline models.

Dataset We employ DIRT dataset (Chen et al. 2022), which was created by decompiling C-language GitHub repositories using IDA Pro. The dataset comprises high-quality pairs of decompiled code and their corresponding function names. In our experiments, 40,000 functions are used for training and 5,000 for testing, with the function names in the decompiled code replaced by the token `<FUNC_NAME>`.

Baselines Using Llama-3.1-8B-Instruct as the base model, we define the following three baselines:

- **Base Model**: The base model without additional training. To minimize formatting errors, the following instruction prompt was used:

```
What is the original function name of this code?
The original name is masked by '<FUNC_NAME>'. Follow
the output format: "The function name is 'xxx'."
{code}
```

- **FT with Label**: A model fine-tuned using only function names, representing a conventional approach without incorporating reasoning evidence.
- **FT with CoT**: A model fine-tuned directly on LLM outputs generated with Chain-of-Thought prompting, learning both reasoning evidence and function names simultaneously as a single task.

We evaluate the effect of reasoning evidence by comparing RevLlama with the FT with Label baseline and assess the effect of multi-task learning by comparing it with the FT with CoT baseline.

[6]https://huggingface.co/sentence-transformers/all-mpnet-base-v2

Results Table 1 shows the evaluation results for RevLlama and the baseline models. RevLlama consistently outperformed all baselines across every evaluation metric. Specifically, it achieved an SBERT score of 0.528, a 6.9% improvement over the next best model (FT with CoT at 0.494). For traditional metrics, RevLlama obtained an F1 score of 0.448, a 10.6% improvement over FT with CoT (0.405).

The progression in performance across the different training approaches highlights clear improvements at each stage. Fine-tuning with labels increased the base model's F1 score by 20.8% (from 0.312 to 0.377). Incorporating Chain-of-Thought reasoning further boosted the F1 score to 0.405. Finally, RevLlama's knowledge distillation approach yielded the most substantial improvements, with consistent gains across all metrics, demonstrating its effectiveness in capturing both semantic understanding and naming accuracy.

RevLlama's superior performance compared to FT with CoT suggests that training on separate datasets for function names and reasoning rationales is more effective than learning them together as a single task. This separation likely enables the model to better capture the unique characteristics of each aspect, leading to improved overall results.

Exp. 2: Comparison with Existing Methods

Dataset To ensure fair comparison with existing methods, we conducted evaluation experiments on the widely-used Nero dataset (David, Alon, and Yahav 2020). Since the dataset is provided in executable format, we decompiled it using IDA Pro and extracted decompiled functions with matching file names and function names from the training and test sets. A dataset for function name prediction was constructed by masking function names in the decompiled code with `<FUNC_NAME>`.

Baselines We compared RevLlama against three existing methods specialized for binary analysis: SymLM (Jin et al. 2022), Nero (David, Alon, and Yahav 2020), and HexT5 (Xiong et al. 2023). For these methods, performance values (Precision, Recall, F1-score) were adopted from their respective papers, evaluated on the same dataset. Additionally, we measured and included the performance of GPT-4o, a state-of-the-art LLM, for comparison.

Results As shown in Figure 5, RevLlama outperformed all existing methods and GPT-4o across evaluation metrics. RevLlama achieved the highest F1 score of 0.587, representing an 8.7% improvement over the previous state-of-the-art method, HexT5 (F1: 0.54). In precision, RevLlama scored 0.594, marking a 7.9% improvement over HexT5 (Precision: 0.55).

Notably, RevLlama also surpassed GPT-4o (F1: 0.488) by 20.3%. These results demonstrate that task-specific knowledge distillation from LLMs can outperform general-purpose LLMs, offering superior performance while maintaining compatibility with secure local environments.

Qualitative Analysis

To better understand the characteristics of RevLlama's predictions, we analyze concrete examples of predicted func-

Table 2: Examples of predicted function names and their SBERT scores.

True	RevLlama		GPT-4o	
	function name	SBERT	function name	SBERT
__update_last_label	set_last_label	0.872	set_label	0.662
utf8len	utf8_strlen	0.727	getStringLength	0.306
skip_past_newline_1	skip_past_line	0.811	find_next_line	0.633
do_tutorial	do_tutorial	1.000	process_script_line	0.099

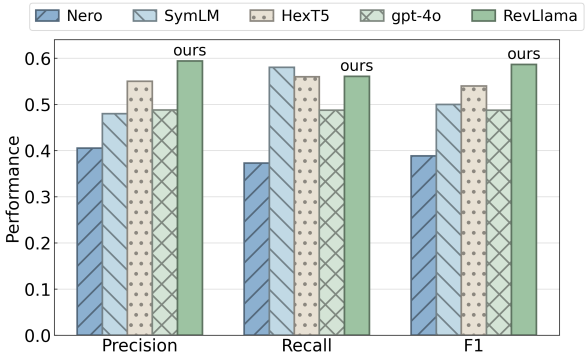

Figure 5: Evaluation results on the Nero dataset. Scores for RevLlama and GPT-4o were obtained from our measurements, while those for the other methods are cited from their respective papers.

tion names. Table 2 shows examples of function names predicted by RevLlama and GPT-4o, along with their Sentence-BERT scores. Analysis of RevLlama's predictions reveals that it more accurately captures the functionality contained in the original function names.

For instance, for the function `utf8len`, RevLlama predicts `utf8_strlen`. This prediction retains not only the string length calculation functionality but also the contextual information about handling UTF-8 encoding. In contrast, GPT-4o predicts `getStringLength`, which captures the basic functionality but lacks information about character encoding. This difference in prediction is reflected in the SBERT scores, with RevLlama scoring 0.727 and GPT-4o scoring 0.306.

Similarly, for `skip_past_newline_1`, RevLlama predicts `skip_past_line`, accurately expressing the essential line-advancing operation. In comparison, GPT-4o's prediction of `find_next_line`, while referring to a similar operation, results in a more general and ambiguous expression. A notable example is the case of `do_tutorial`, where RevLlama achieves perfect prediction (SBERT score 1.000), while GPT-4o makes a less relevant prediction of `process_script_line`.

These examples demonstrate that by leveraging reasoning evidence, RevLlama attains a deeper understanding of a function's behavior and generates more precise function names. In particular, it effectively exploits the contextual information present in decompiled code to yield more specific and accurate predictions.

Discussion

This section discusses the effectiveness and limitations of our approach.

Effectiveness

Our experimental results highlight the effectiveness of knowledge distillation with reasoning rationales for function name prediction. RevLlama achieves substantial improvements over existing methods and GPT-4, with up to a 51.0% increase in F1 score compared to prior approaches. These gains can be attributed to three key aspects of our knowledge distillation methodology.

First, learning reasoning rationales enables RevLlama to develop enhanced program comprehension capabilities. The model systematically analyzes decompiled code, leveraging both structural patterns and semantic relationships. This facilitates more accurate interpretation of function behaviors, resulting in precise name predictions.

Second, our approach offers significant practical advantages in computational efficiency and deployment flexibility. By utilizing fine-tuning instead of in-context learning, we substantially reduce input length requirements, optimizing resource utilization. The model can operate efficiently in CPU-only environments using standard quantization techniques, making it accessible for security analysts in restricted settings.

Third, RevLlama provides superior output control compared to general-purpose LLMs. While LLMs often produce inconsistent or verbose outputs, RevLlama generates standardized function names suitable for seamless integration with reverse engineering tools such as IDA Pro or Ghidra. Additionally, reasoning rationales can be displayed as code comments, enabling practical features like automatic function renaming.

Limitations

Our approach has several limitations that warrant future exploration. First, the model currently treats each function independently, without considering function call relationships or broader program-wide context. Real-world programs often involve complex interdependencies that could enhance name prediction accuracy if leveraged effectively.

Second, while our evaluation demonstrates effectiveness on benchmark datasets, further validation on real-world malware samples is necessary. Malware often employs code obfuscation techniques that pose significant challenges to our current method.

Conclusion

In this paper, we presented RevLlama, an approach to function name prediction in binary analysis that combines LLM reasoning capabilities with local execution requirements. Through knowledge distillation from LLMs, RevLlama achieves superior performance compared to both existing methods and GPT-4o, while maintaining the ability to operate in secure local environments. Our experimental results demonstrate a 20.3% improvement in F1 score over GPT-4o, validating the effectiveness of our approach.

Future work could focus on incorporating program-wide context and addressing challenges posed by obfuscated code to further enhance prediction accuracy. We believe this research represents a significant step forward in enabling secure and efficient binary analysis, making sophisticated capabilities accessible in restricted environments.

References

Al-Kaswan, A.; Ahmed, T.; Izadi, M.; Sawant, A. A.; Devanbu, P.; and van Deursen, A. 2023. Extending Source Code Pre-Trained Language Models to Summarise Decompiled Binarie. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 260–271. IEEE.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.

Chen, Q.; Lacomis, J.; Schwartz, E. J.; Goues, C. L.; Neubig, G.; and Vasilescu, B. 2022. Augmenting Decompiler Output with Learned Variable Names and Types. In Butler, K. R. B.; and Thomas, K., eds., *31st USENIX Security Symposium, USENIX Security 2022*, 4327–4343. USENIX Association.

David, Y.; Alon, U.; and Yahav, E. 2020. Neural reverse engineering of stripped binaries using augmented control flow graphs. *Proc. ACM Program. Lang.*, 4(OOPSLA): 225:1–225:28.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop*.

Hsieh, C.-Y.; Li, C.-L.; Yeh, C.-k.; Nakhost, H.; Fujii, Y.; Ratner, A.; Krishna, R.; Lee, C.-Y.; and Pfister, T. 2023. Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, 8003–8017. Association for Computational Linguistics.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Jin, X.; Pei, K.; Won, J. Y.; and Lin, Z. 2022. SymLM: Predicting Function Names in Stripped Binaries via Context-Sensitive Execution-Aware Code Embeddings. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022*, 1631–1645. ACM.

Kwon, W.; Li, Z.; Zhuang, S.; Sheng, Y.; Zheng, L.; Yu, C. H.; Gonzalez, J.; Zhang, H.; and Stoica, I. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In Flinn, J.; Seltzer, M. I.; Druschel, P.; Kaufmann, A.; and Mace, J., eds., *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, 611–626. ACM.

OpenAI; Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; et al. 2024. GPT-4 Technical Report. arXiv:2303.08774.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019*, 3980–3990. Association for Computational Linguistics.

Shang, X.; Cheng, S.; Chen, G.; Zhang, Y.; Hu, L.; Yu, X.; Li, G.; Zhang, W.; and Yu, N. 2024. How Far Have We Gone in Binary Code Understanding Using Large Language Models. arXiv:2404.09836.

Team, G.; Anil, R.; Borgeaud, S.; Alayrac, J.-B.; Yu, J.; Soricut, R.; Schalkwyk, J.; Dai, A. M.; Hauth, A.; Millican, K.; et al. 2024. Gemini: A Family of Highly Capable Multimodal Models. arXiv:2312.11805.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, 5998–6008.

Wang, Y.; Wang, W.; Joty, S. R.; and Hoi, S. C. H. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021*, 8696–8708. Association for Computational Linguistics.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E. H.; Le, Q. V.; and Zhou, D. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022*.

Xiong, J.; Chen, G.; Chen, K.; Gao, H.; Cheng, S.; and Zhang, W. 2023. HexT5: Unified Pre-Training for Stripped Binary Code Information Inference. In *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023*, 774–786. IEEE.