Does Representation Matter? Evaluating IRs for LLM-based Binary Decompilation

Tomás Pelayo-Benedet[1], Kevin Borgolte[2], Ricardo J. Rodríguez[1]

[1]Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Spain

[2]Ruhr-Universität Bochum, Germany

Universidad Zaragoza

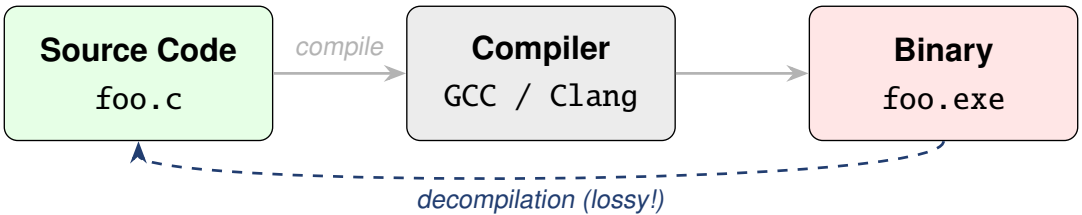
RUHR UNIVERSITÄT BOCHUM

RUB

Workshop on Binary Analysis Research (BAR) 2026

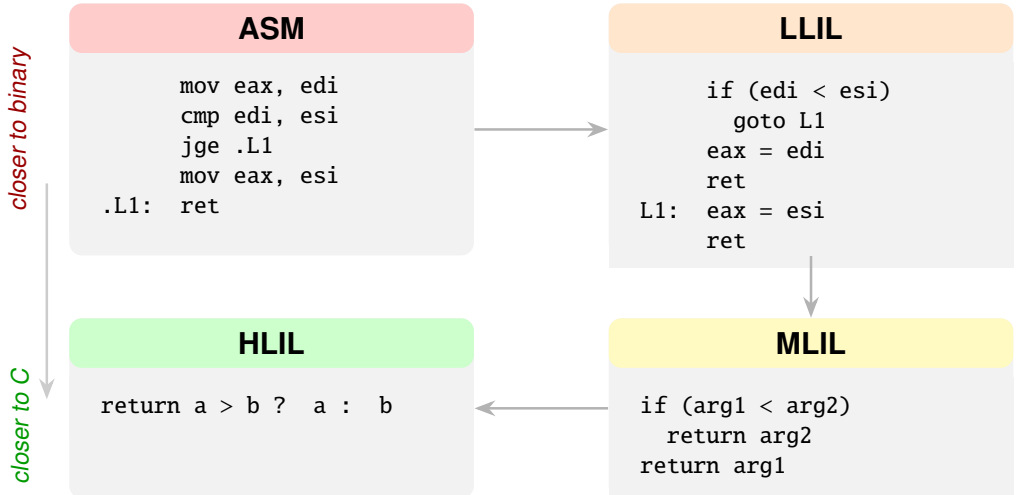
Motivation

Binary Decompilation: Recovering What Compilation Discards



Motivation & Problem

Four Levels of Abstraction, from Raw Bytes to High-Level Constructs (max Function)



Motivation & Problem

Existing work treats decompilation as a direct translation

Self-Constructed Context Decompilation with Fined-grained Alignment Enhancement

Yunlong Feng, Dechuan Teng, Yang Xu, Honglin Mu, Xiao Xu, Libo Qin, Qingfu Zhu*, Wanxiang Che
Harbin Institute of Technology, China
{ylfeng,dcteng,yxu,hlmu,xxu,lbqin,qfzhu,car}@ir.hit.edu.cn

Abstract

Decompilation transforms compiled code back into a high-level programming language for analysis when source code is unavailable. Previous work has primarily focused on enhancing decompilation performance by increasing the scale of model parameters or training data for pre-training. Based on the characteristics of the decompilation task, we propose two methods: (1) Without fine-tuning, the Self-Constructed Context Decompilation (sc^{2}dec) method recompiles the LLM's decompilation results to construct pairs for in-context learning, helping the model improve decompilation performance. (2) Fine-grained Alignment Enhancement (FAE), which meticulously aligns assembly code with source code at the

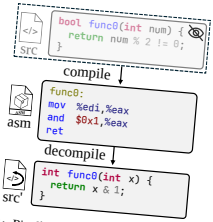
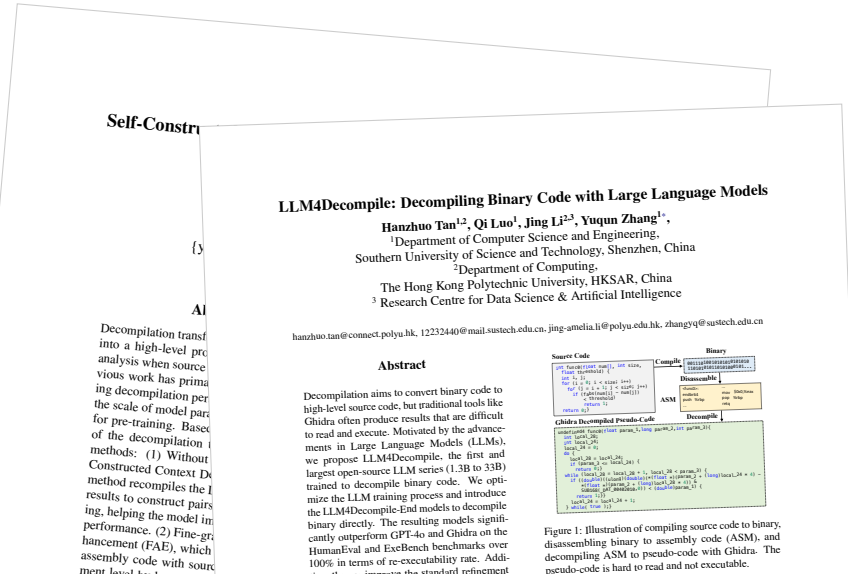


Figure 1: Pipeline of the decompilation. The input for decompilation tasks is typically assembly, with the source code invisible

Motivation & Problem

Existing work treats decompilation as a direct translation



LLM4Decompile: Decompiling Binary Code with Large Language Models

Hanzhuo Tan[1,2], Qi Luo[1], Jing Li[2,3], Yuqun Zhang[1*],
[1]Department of Computer Science and Engineering,
Southern University of Science and Technology, Shenzhen, China
[2]Department of Computing,
The Hong Kong Polytechnic University, HKSAR, China
[3] Research Centre for Data Science & Artificial Intelligence

hanzhuo.tan@connect.polyu.hk, 12232440@mail.sustech.edu.cn, jing-amelia.li@polyu.edu.hk, zhangyq@sustech.edu.cn

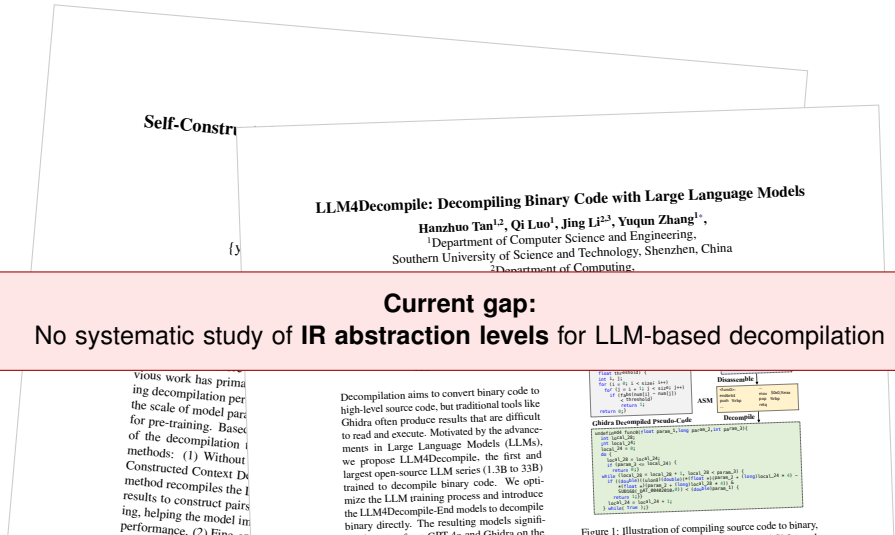
Abstract

Decompilation aims to convert binary code to high-level source code, but traditional tools like Ghidra often produce results that are difficult to read and execute. Motivated by the advancements in Large Language Models (LLMs), we propose LLM4Decompile, the first and largest open-source LLM series (1.3B to 33B) trained to decompile binary code. We optimize the LLM training process and introduce the LLM4Decompile-End models to decompile binary directly. The resulting models significantly outperform GPT-4o and Ghidra on the HumanEval and ExeBench benchmarks over 100% in terms of re-executability rate. Addi-

Figure 1: Illustration of compiling source code to binary, disassembling binary to assembly code (ASM), and decompiling ASM to pseudo-code with Ghidra. The pseudo-code is hard to read and not executable.

Motivation & Problem

Existing work treats decompilation as a direct translation



Current gap:
No systematic study of **IR abstraction levels** for LLM-based decompilation

Experimental Setup

LLMs Evaluated (5 Models)

Large

GPT-4.1
Apr 2025

Large

Claude Sonnet 4
May 2025

Small

GPT-4.1-nano
Apr 2025

Small

GPT-4o-mini
Jul 2024

Small

Claude Haiku 3.5
Oct 2024

Experimental Setup

Dataset: ExeBench

Cyclomatic Complexity

$CC = E - N + 2P$

E = edges in control flow graph
N = nodes in control flow graph
P = number of connected components

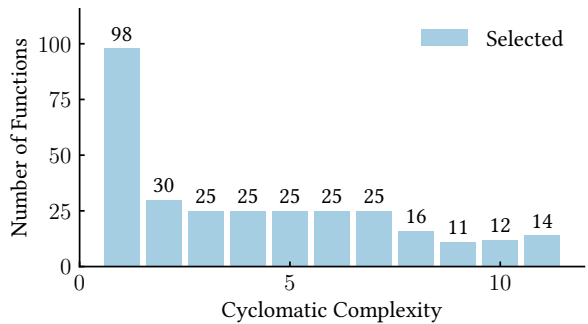
Selection strategy

Per complexity level, we select:

$$\max(25,\ 10\% \text{ of functions})$$

Functional verification

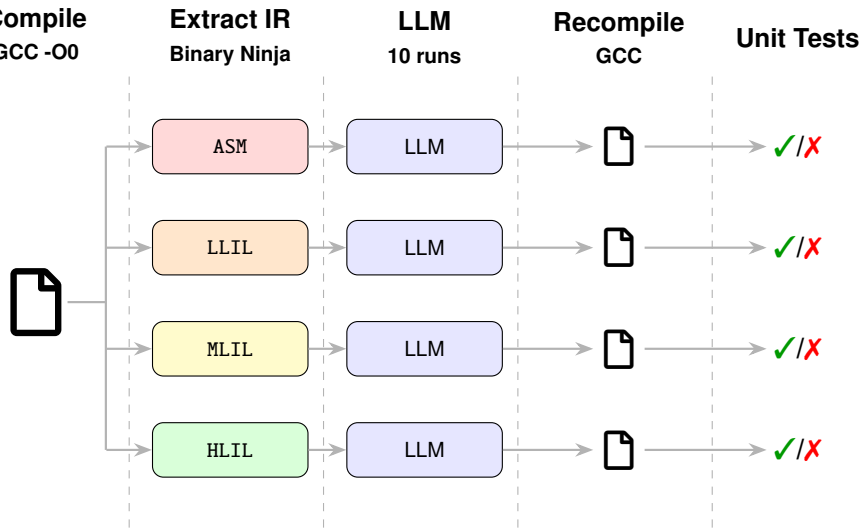
Each function includes **10 unit tests**



*306 selected from the **real_test** split of ExeBench*

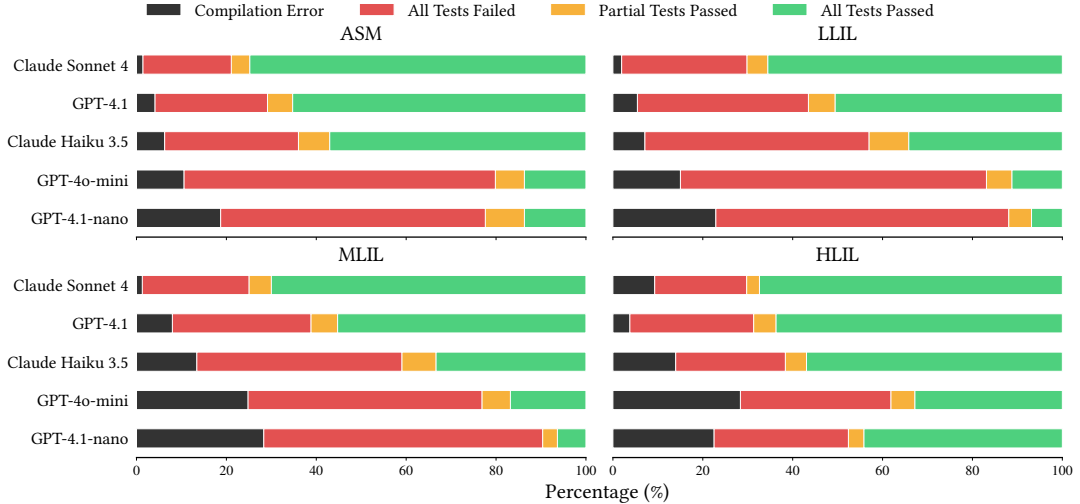
Experimental Setup

Evaluation pipeline



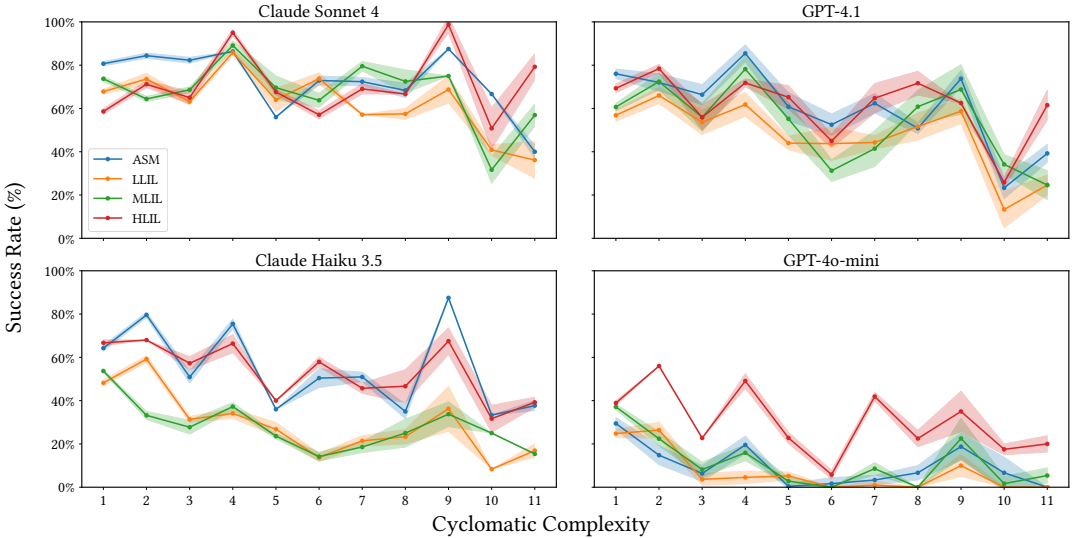
Results

Decompilation performance across models and IR levels



Results

Decompilation performance across models and IR levels



Code Complexity & Decompilation Difficulty

Halstead metrics: counting operators and operands

```
// max of two integers
int max(int a, int b) {
return ( a > b ) ? a : b ;
}
```

Operators **Operands**

η_1	unique operators	N_1	total operators
η_2	unique operands	N_2	total operands

$N = N_1 + N_2$ total program length

$\eta = \eta_1 + \eta_2$ total vocabulary

Halstead metrics

$$V = N \log_2 \eta$$

***Volume**: size of the program*

$$D = \frac{\eta_1}{2} \cdot \frac{N_2}{\eta_2}$$

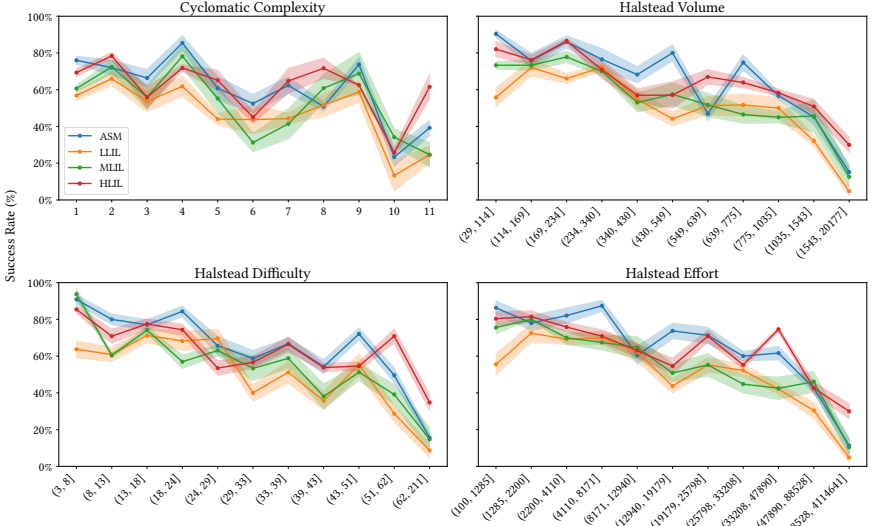
***Difficulty**: how hard to understand*

$$E = D \cdot V$$

***Effort**: mental effort required*

Code Complexity & Decompilation Difficulty

Halstead metrics beat cyclomatic complexity as predictors



Conclusions

(1) IR choice matters

Small models: HLIL improves correctness by >30 pp over ASM
Large models: robust at all IR levels

(2) Recompilability vs. Correctness trade-off

ASM excels at recompilability; HLIL excels at functional correctness

(3) Halstead > Cyclomatic complexity

Halstead metrics are stronger predictors of decompilation difficulty

(4) No universally optimal IR

Best choice depends on model size and task requirements

Dataset on GitHub

Does Representation Matter? Evaluating IRs for LLM-based Binary Decompilation

Tomás Pelayo-Benedet[1], Kevin Borgolte[2], Ricardo J. Rodríguez[1]

[1]Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Spain

[2]Ruhr-Universität Bochum, Germany

Universidad Zaragoza

RUHR UNIVERSITÄT BOCHUM

RUB

Workshop on Binary Analysis Research (BAR) 2026