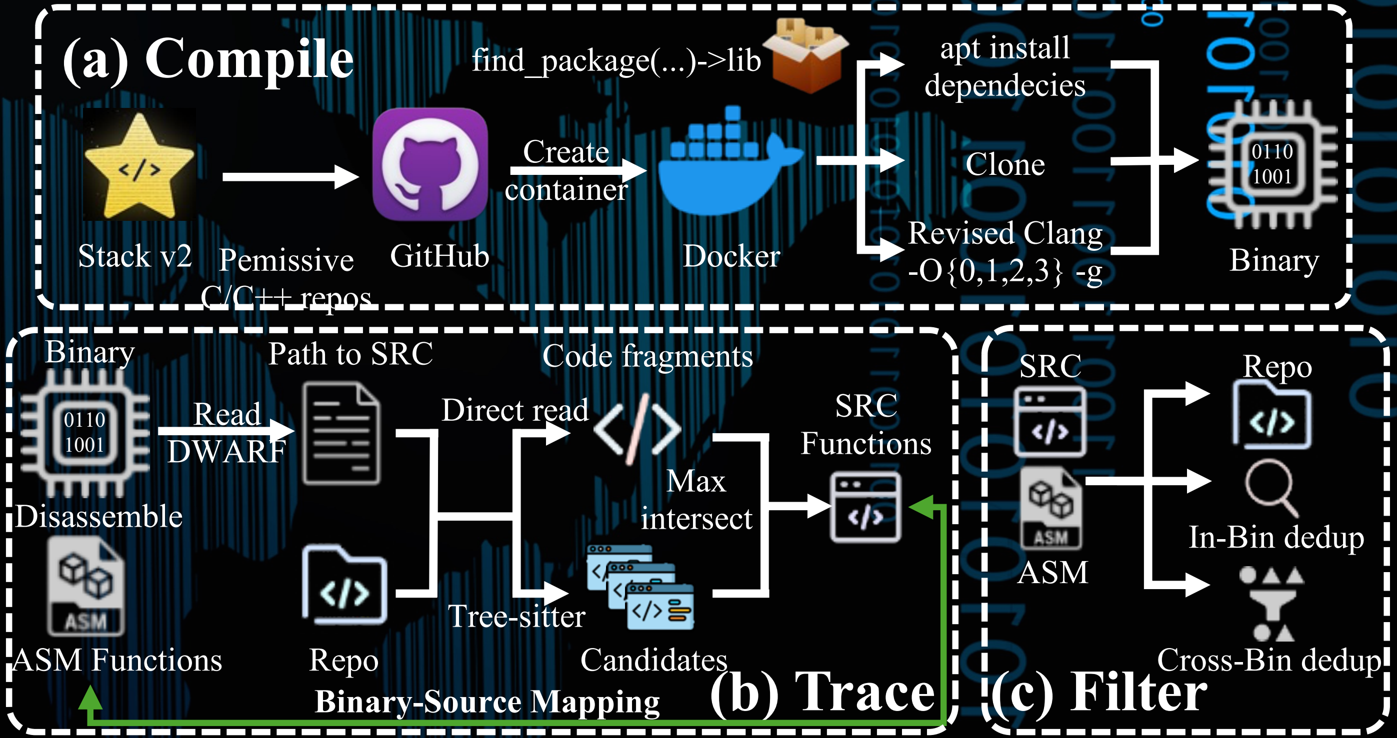
Decompile-Bench

Million-Scale Binary-Source Function Pairs for Real-World Binary Decompilation

Hanzhuo Tan



First and **largest** binary-source dataset, 2M pairs condensed from 100M

Evaluation set: Github2025 (least data leakage)

DecompileBench

Summary

100M	500G	5K	20%
Raw Pairs	**Binaries**	**Projects**	**Performance**

DecompileBench

Previous Work

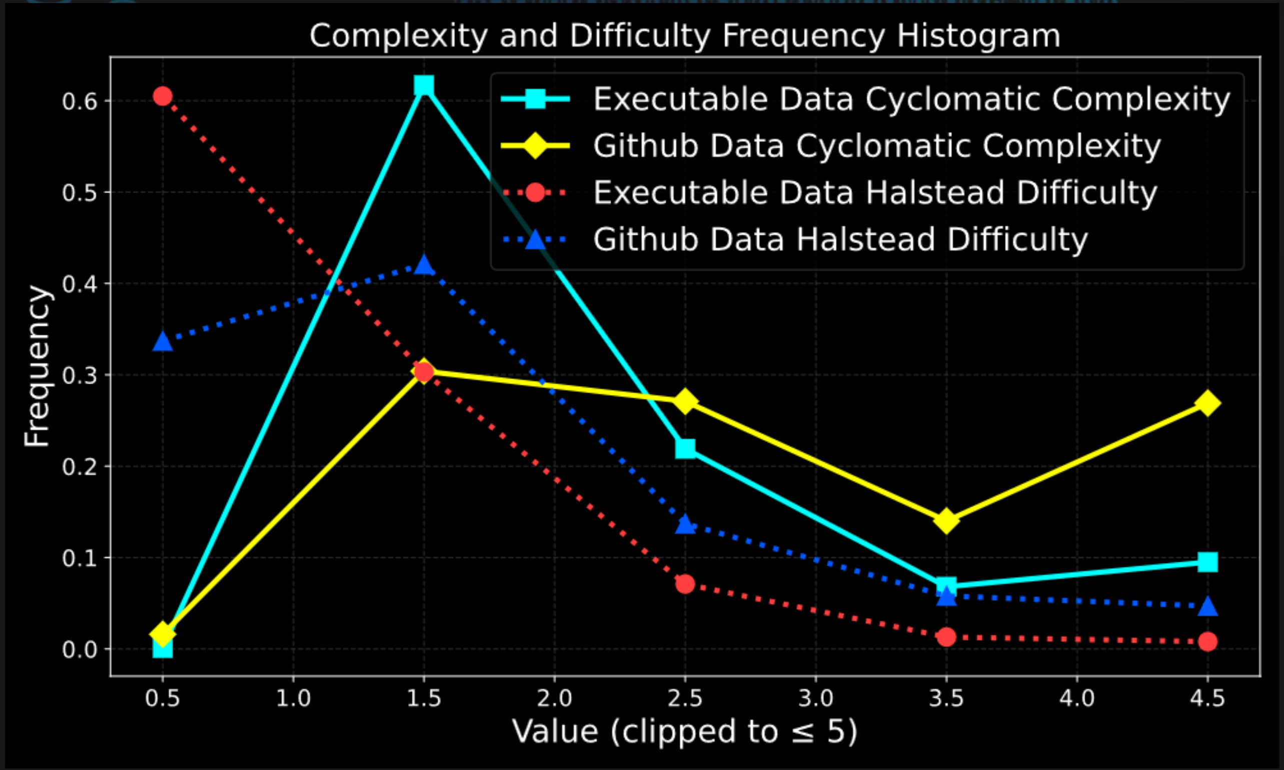
Category	Code Contest	Synthetic
Benchmarks	Decompile-Dataset [26] HumanEval-Decompile [11]	AnghaBench [27] ExeBench [28] CSMith [48]
Samples	`int func0(float num [], int size, float threshold){ int i, j for (i = 0; i < size; i++) for (…) if (…)…}` **a**	`struct of_device{int dummy; } ; struct device {int dummy; } ; struct device* bus_find_device (int /*<<< orphan*/ *,…) ; int /*<<< orphan*/ …;}` **b**

Category	Fragment-Level	Real-World Function-Level
Benchmarks	Dire[9] Dirty [29] Resym [10] Assemblage [30]	CodeCMR [53] Idioms [54] **Decompile-Bench**
Samples	source code / rva / func: `return J();…` 0x00004320 46; `return t->size();…` 0x00001A80 55; `column++;…` 0x000034F0 58; `if (lookahead &…` 0x0000B050 **c**	`Game::Game(std::size_t grid_width, std::size_t grid_height) : snake(grid_width, grid_height), snake2(grid_width, grid_height), engine(dev()), … { snake.setPosition(…)…}` **d**

We need
more real data

DecompileBench

Why not synthetic data?



Cyclomatic Complexity (CC)
Equation: CC=E−N+2
E = number of edges in the control flow graph
N = number of nodes
P = number of connected components (usually 1 for a single program)

Halstead Difficulty (D)
Equation: $D = \frac{\eta_1}{2} \times \frac{N_1}{\eta_2}$
η_1= the number of distinct operators (+-*/)
η_2= the number of distinct operands (variables)
N_1= the total number of operators
N_2= the total number of operands

We need
more real data

Compile-Trace-Filter (CTF)

Compile

1. Patch Clang driver and invocation logic to force our desired -O{0,1,2,3} and -g flags
2. parse CMakeLists.txt for find_package(...), query GPT for the correct install commands
3. Leverage Docker to isolate install environment

Trace

Algorithm 1 Pairing Binary Functions with Source Functions

Input: Binary project B, source project S
Output: Map M, containing the mapping between each binary function in B to a source function

```
for all binary function f_b in B do
    Extract DWARF information in f_b to obtain corresponding source-code lines; group them as func_segment
    Candidates ← ∅
    for all source line ℓ in func_segment do
        Use Tree-sitter on S to extract the complete source function f_s containing ℓ
        Candidates ← Candidates ∪ {f_s}
    end for
```

$$f_s^* \leftarrow \arg\max_{f_s \in \text{Candidates}} \left| \textit{func\_segment} \cap \text{Lines}(f_s) \right|$$

```
    M[f_b] ← f_s*
end for
return M
```

Filter

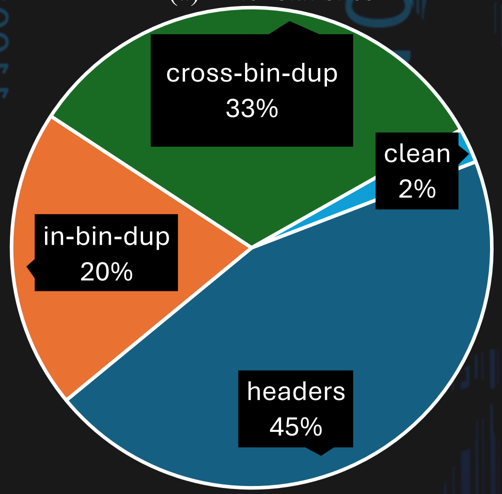
1. Project-scope filter. Remove any source function not in the target repository. E.g., get() and set()
2. In-binary deduplicator. Several functions within one binary claim the same source function
3. Cross-binary deduplicator. Apply MinHash-LSH to eliminate near-duplicate

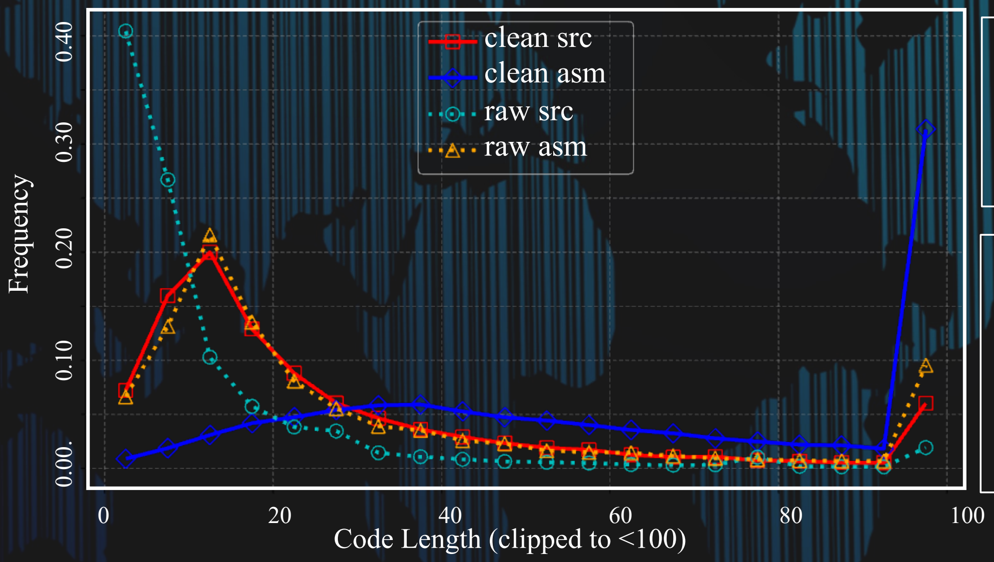
DecompileBench

Filter

(a) Filter statistics. (b) Length histogram. (c) Filtered samples

(a) Filter statistics





(c) Functions excluded by filters

```
~vector(){
std::_Destroy(
this._M_start,
this._M_finish,
_M_get_Tp_allocator());}
```

Vector Destructor

```
T* operator=(T* p){
  if (p)
    p->AddRef();
  if (_p)
    _p->Release();
  _p = p;
  return p;}
```

Overload

1.DecompileBench retains only 2% of the raw data (two million functions out of 100M)

2. Removing codes from header files or duplicates eliminates the vast majority (40% of all raw data) of short snippets (under five lines)

3. The project-scope filter mostly discards trivial helpers (e.g., get()), constructors, and destructors from system or dependency headers. In-binary duplicates stem from template instantiation.

Main Results

Re-Executability Rates	HumanEval					MBPP				
	O0	O1	O2	O3	AVG	O0	O1	O2	O3	AVG
LLM4Decompile-End	26.22	12.81	14.03	13.42	16.22	29.16	16.99	17.92	18.07	20.54
+Exebench	26.22	13.89	13.11	13.89	16.78	27.16	17.66	18.74	17.25	20.20
+Decompile-Bench-raw	24.70	13.41	13.11	12.20	15.86	26.49	16.48	15.40	14.32	18.17
+Decompile-Bench	33.23	18.60	16.47	15.24	20.89	35.06	21.56	22.80	20.28	24.93

1. LLM4Decompile-End is trained on ExeBench and additional training from the same source does not provide further benefit
2. Model trained on Decompile-Bench-raw (no filter), the low-quality nature leads to 2.2% and 11.4% re-executability **decline** against to the base model
3. Clean and compact data from Decompile-Bench significantly improves the re-executablility of the base model for **over 20%**