

wbh
WILHELM BÜCHNER
HOCHSCHULE

Eine Hochschule der Klett Gruppe

REFORGE: Benchmarking LLMs on Decompiled Binary Function Naming

Nicolas Koller (Systrion AG) · Andreas U. Schmidt
(WBH)

AC 2026 · Lisbon

wb-fernstudium.de

AI reverse-engineers everything?

Autumn 2026: within weeks, several public claims that AI had “*reverse-engineered*” software.

Adobe and Microsoft Office “clones”

ArtCraft, Sep–Oct 2026 (Akita, 2026; Al Jazeera, 2026)

- Photoshop, Lightroom, Word, Excel, PowerPoint re-implemented in Rust with Claude Opus 5.5
- By the developer’s account: clean-room builds from public file-format specs, no binary reversed
- Early alpha: 800+ open issues in two weeks

REA: “Reverse Engineer Anything”

Oct 2026 (Startup Fortune, 2026; Korben, 2026)

- #1 on GitHub trending, 10,000+ stars in days: a harness connecting coding agents to Ghidra, IDA, JADX
- Hands-on test: 4 of 4 tasks solved, but plain Claude + Ghidra also solved 4 of 4
- One “success” was recall: the agent searched for cheat codes before it had seen them

Also: Minecraft’s Steve now walks through Elden Ring, GTA V and Call of Duty via AI-assisted mash-ups, that are labeled as reverse-engineering.

Observation: “reverse engineering” now covers rebuilding, tool use and recall. What was recovered from a binary is rarely checked but rather guessed based on functionality

Project Ire: an LLM reverse engineer in production

How it works (Caswell et al., 2025)

- 1 Triage of an unknown binary
- 2 Control-flow recovery with angr and Ghidra
- 3 LLM analyses and summarises functions, one by one
- 4 Report with a verdict: malicious or benign
- 5 Validator checks claims against the agent's own evidence

Being integrated into Microsoft Defender as “Binary Analyzer”.

Hard files: when it says “malicious” it is right 89% of the time, but it catches only about 1 in 4 malicious files.

Reported evaluation: per file only

Test set	Precision	Recall	FPR
Public Windows drivers	0.98	0.83	2%
~4,000 hard real-world files	0.89	0.26	4%

- **Precision:** of files flagged malicious, share truly malicious
- **Recall:** of truly malicious files, only small share caught
- **FPR:** of harmless files, share wrongly flagged

Caswell, B. et al. (2025). Project Ire autonomously identifies malware at scale. Microsoft Research Blog.

On the Red Team Side: Autonomy is real, reliability is unmeasured

- **Campaign:** detected Sep 2025, attributed to a state-sponsored group (GTG-1002); ~30 targets, a handful of successful intrusions
- **Method:** attackers posed as a security firm; attack split into small, harmless-looking tasks; open-source pentest tools via MCP
- **Autonomy:** AI performed an estimated 80–90% of tactical work; humans approved key steps
- **Caveat:** the vendor reporting on its own model

“Claude frequently overstated findings and occasionally fabricated data during autonomous operations.”

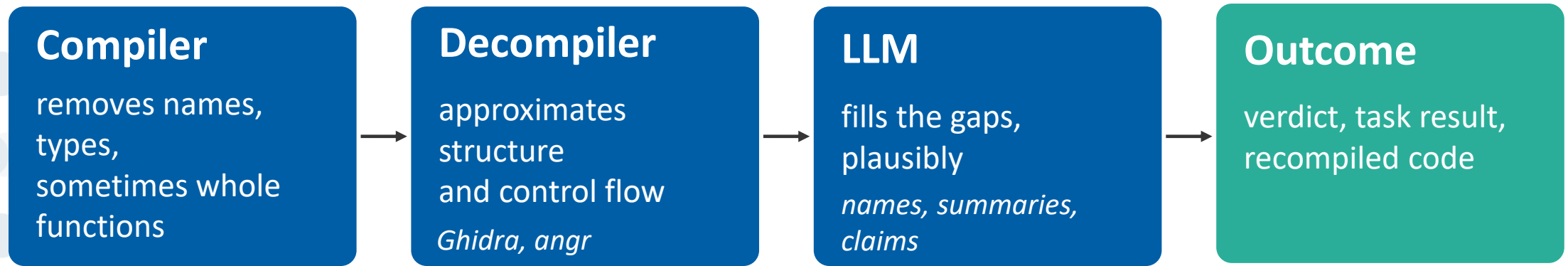
Operators had to validate results by hand; Anthropic calls this an obstacle to fully autonomous attacks.

Models over-report and invent. So what do these systems actually do, step by step?

Anthropic (2025). Disrupting the first reported AI-orchestrated cyber espionage campaign.

State of the Art I:

What is reverse engineering with LLMs?



Uncertainty accumulates at every stage (Leger et al., 2021)

*Task span all of reverse
engineering, that is not handled
by the decompiler*

Where systems are checked today

Project Ire: file verdict

SRE-Bench: task outcome

Decompiler benchmarks: does it recompile?

Not checked: which functions the model
actually understood, and which ones made
it fail.

State of the Art II:

How the field measures today

SRE-Bench

(Spence et al., 2026)

Agents get a binary and a goal; deterministic graders check the behaviour. 19 programs built from scratch (5,000+ expert hours).

Best model fully solves 31.5%. The right answer is often among its candidates, but not the one picked.

Recompile more, preserve less

(Liu, Raff & Micinski, 2026)

Build rate: does the decompiled function compile again? Behaviour preserved: same outputs as the original on generated inputs?

LLM rewriting: build rate 75% → 90%, behaviour preserved 74% → 62%. Invented fields and types: 53% vs. 20% changed behaviour.

Decompile-Bench

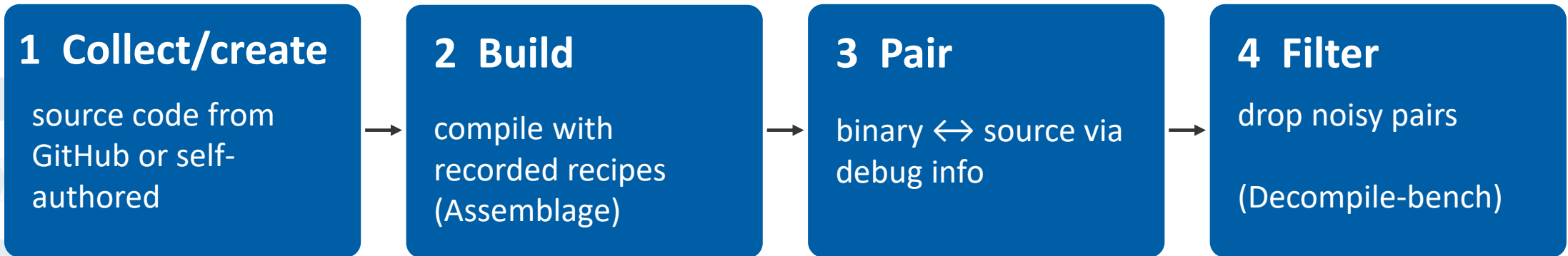
(Tan et al., 2025)

Million-scale binary–source function pairs from GitHub projects, for training and evaluation.

100M collected pairs → 2M kept (~2%). Losses per optimization level are not reported.

Open Question: What functions survived the compilation, decompilation and filtering?

Ground Truth construction – The Problem

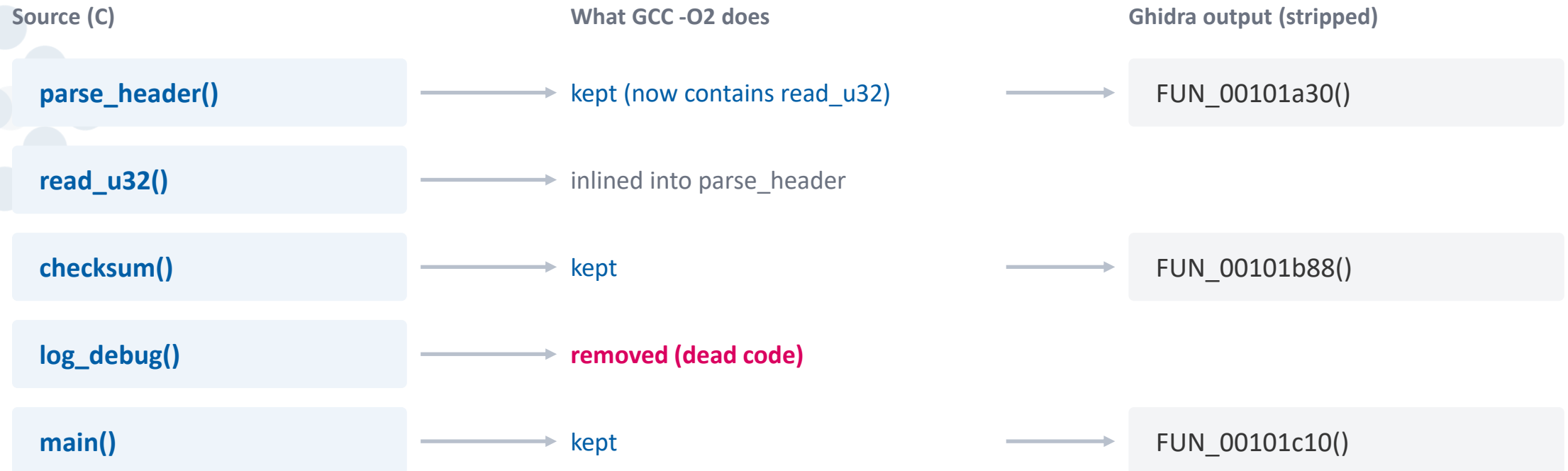


- Behavior is a good indicator of success or failure, but does not give us a reason for any of the two
- Functions are lost at every step: failed builds, ambiguous pairs, filtering
- Compiler-optimization changes the structure of the binary

Liu et al. (2024); Tan et al. (2025); Gao et al. (2025); Spence et al. (2026).

The function-pairing problem

Scoring a model per function needs a reliable pair: binary function $\leftrightarrow$ source function. Compilation breaks it.



Consequence: when a model gets a function wrong, we cannot tell whether the model, the decompiler or the compiler is to blame.

Illustrative example. Many-to-one compilation: Liu et al. (2024). Decompiler fidelity: Dramko et al. (2024).

Our approach: a lab with C-code test specimen instead of a field study

t01 cross-file calls

t02 shared header
macros

t03 header-
dominant

t04 static dup.
names

t05 fn-pointer
callbacks

t06 recursion +
inlining

t07 switch parser

t08 loop-heavy

t09 string formatting

t10 libm math

t11 mixed stress

t12 state machine

t13 goto labels

t14 anonymous
aggregates

t15 deep nesting

226

Function records at -O0
(source inventory)

4 × 3

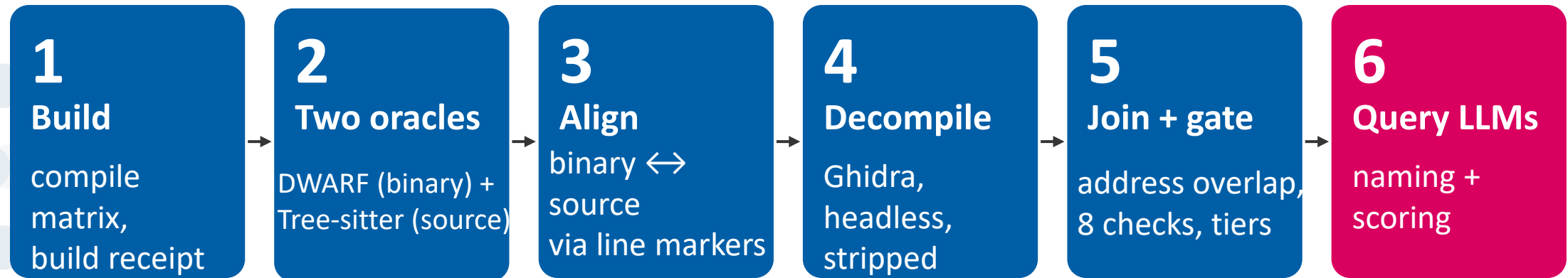
optimization levels ×
binary variants

1 toolchain

GCC 14.2.0 · x86-64 ELF · Ghidra 12.0.3 (pinned)

Debug build → ground truth. **Stripped** build → what the decompiler and the LLM see. Same machine code in both.

Reforge: follow every function



- **Non-destructive:** checks add flags and reasons, they never delete a function record.
- **Deterministic core:** stages 1–5 reproduce byte-identically; Ghidra checked: 1,884 / 1,884 functions identical across runs and machines.
- **Provenance:** every artifact carries SHA-256 hashes back to the build receipt.

The final join, eight checks, three tiers

Join 2: ground truth ↔ Ghidra function, by address overlap (debug and stripped builds share the same machine code)

JOINED_STRONG ≥ 90% address overlap · **JOINED_WEAK** some overlap, below 90%

1 Debug info accepted	5 Strong decompiler join	GOLD passes all eight checks
2 Alignment = MATCH	6 Not compiler noise	SILVER strong join, fails ≥ 1 check (reason recorded)
3 Exactly one candidate	7 CFG not incomplete	BRONZE weak join, or debug info not fully accepted
4 Line overlap ≥ 0.95	8 No fatal decompiler warning	no tier no address range: cannot be joined

Caveat: GOLD functions are smaller (median 34 instructions vs. 52 for SILVER), so tiers mix alignment quality with complexity.

Finding a stable key

Binary identity changes

86% of DWARF function identifiers change between -O0 and -O1. Expected, but it rules out tracking by binary-side IDs or addresses.

Names are not unique

17 functions in our corpus share a name with a different function in the same program, e.g. `static process()`, `validate()`, `report()` in three modules.

Stable key, anchored in the source declaration

test case

file

line

column

name

The declaration site does not move when compiler flags change. This enables paired comparisons: the same function at -O0 and at -O3.

RQ1: how much ground truth survives?

Fig. 1: High-confidence yield (%)

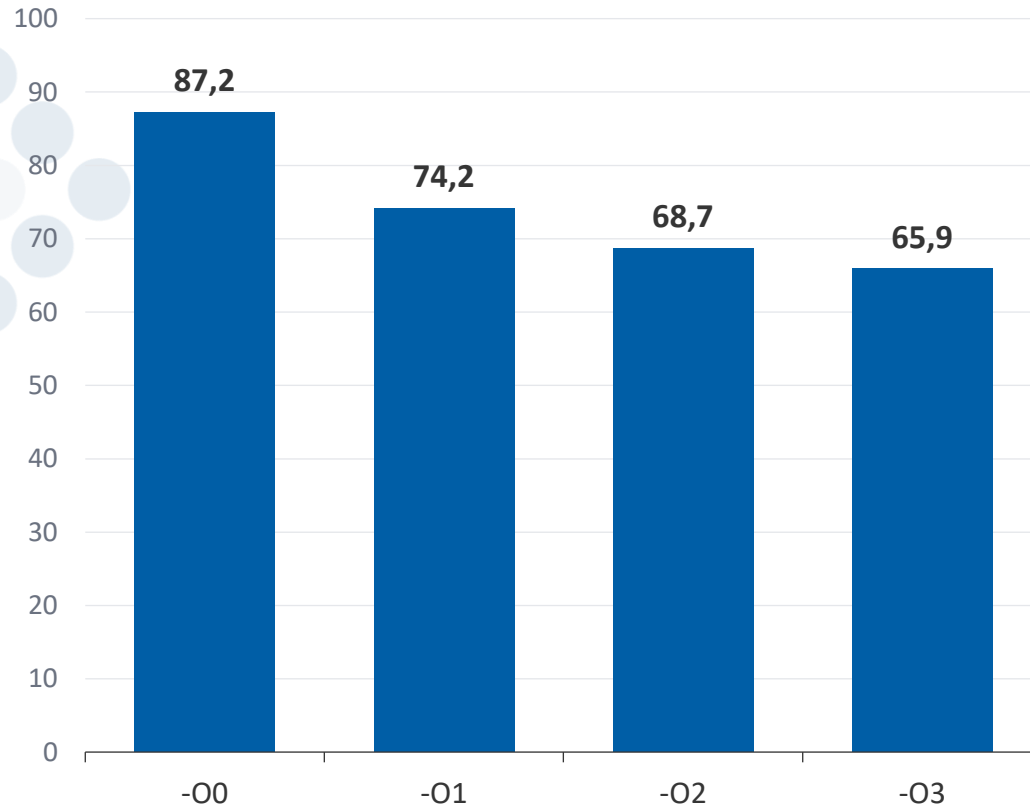
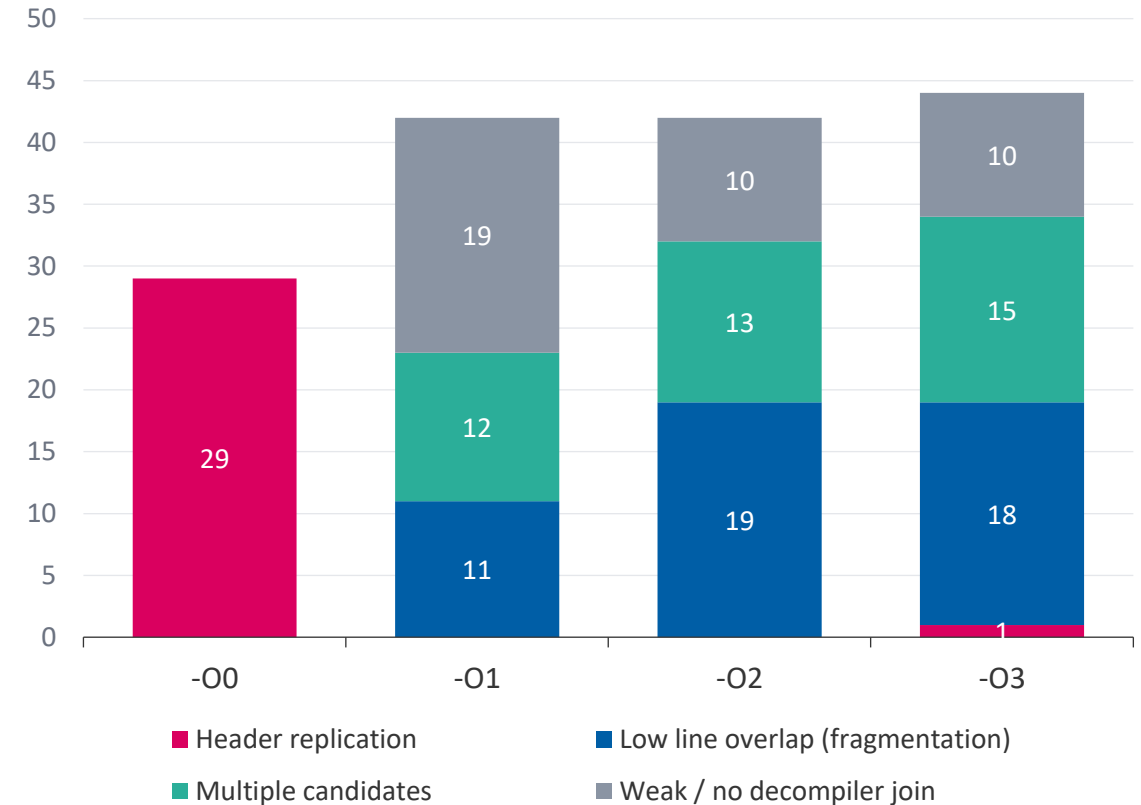


Fig. 2: Why eligible functions fail (count)

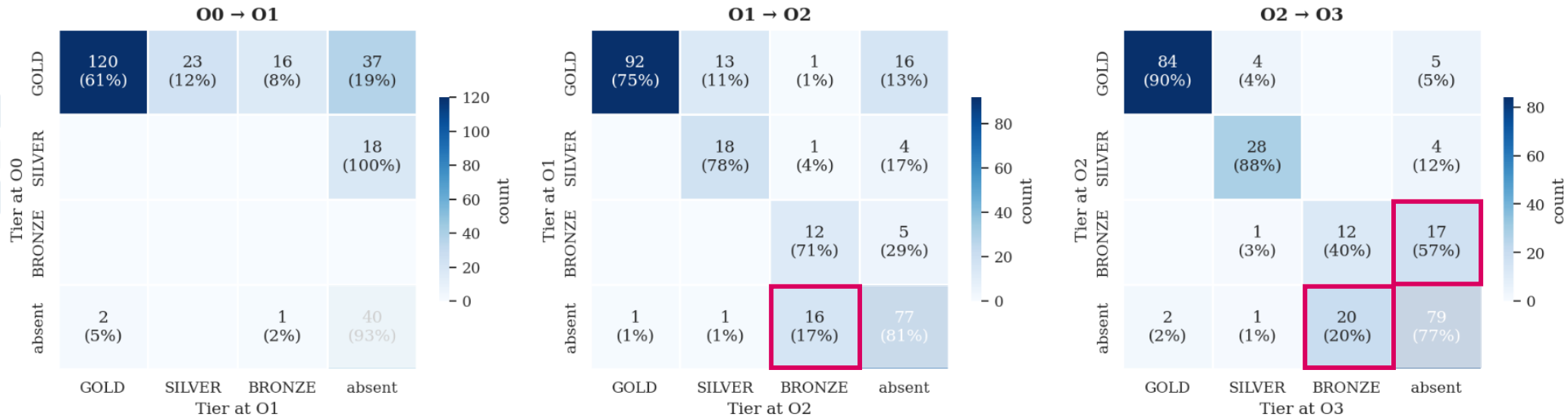


197/226 → 121/163 → 92/134 → 85/129 eligible functions pass all eight checks. -O0 losses: header copies only.
 Higher levels: fragmentation and ambiguity.

Survivorship (1/2): where functions go

Figure 4: Tier Transition Matrices Across Optimization Levels

□ = nameless copies (Case 4)



- **The first step does the most damage:** from -O0 to -O1, 55 functions leave the evaluable set (37 GOLD, all 18 SILVER).
- **Surviving GOLD and SILVER functions keep their tier:** 75% (-O0→-O1), 88% (-O1→-O2), 97% (-O2→-O3).
- **Boxed BRONZE cells are not real movement:** nameless out-of-line copies get no stable identity, so they “leave” and “re-enter” at every level.

Attrition removes functions; it rarely degrades the survivors. At -O3 we evaluate a different population than at -O0.

Survivorship (2/2): paired vs. unpaired

Figure 3: Mean Token F1 by Model × Optimization Level
 (left: unpaired per-opt means; right: paired Δ with bootstrap 95% CI)

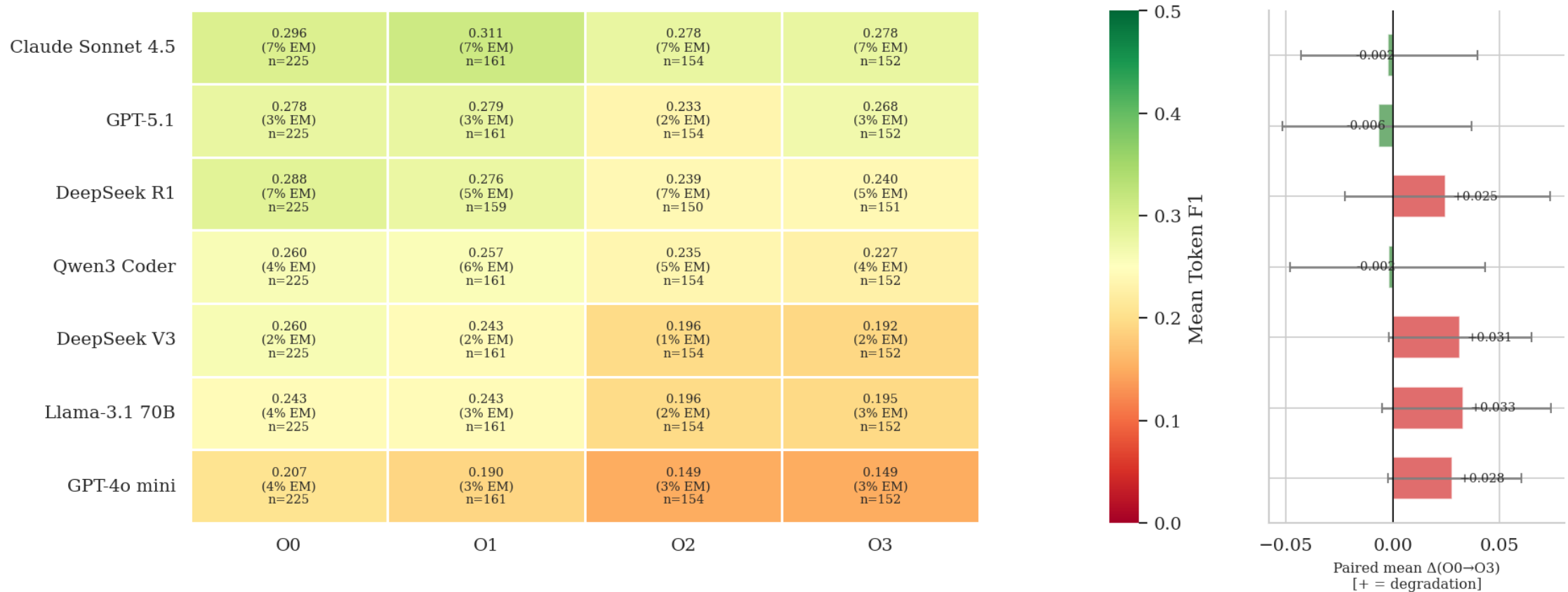


Fig. 5 (thesis Fig. 3). Left: level averages (different populations). Right: paired change per function, 95% bootstrap CI.

Token F1 Example

Ground truth	Prediction	Token F1	Semantic (bge)	Why
mat_multiply	matrix_multiply	0.50	0.95	"mat" ≠ "matrix" as tokens
mat_multiply	matrix_multiplication	0.00	0.88	correct, no shared token
deep_mixed_nest	calculate_weighted_array_score	0.00	0.51	structure name vs. purpose name
deep_loop_nest	find_max_weighted_sum	0.00	0.60	structure name vs. purpose name

- t15 deep nesting: all 9 functions GOLD at every level, the compiler leaves them intact
- Yet the lowest-scoring program (mean F1 0.04): ground-truth names describe structure, models describe purpose

Ground truth is a choice, and it shapes the score. Token F1 is a lower bound; most zeros are still real failures (next slide).

What we got wrong, and how we found it

	Published	Recomputed from the logged artifacts
Text	Survivors called “simpler”	They are larger: median 45 vs. 21 instructions
Text	“40 % lost to inlining and dead-code elimination”	29 of 83 lost functions are still in the binary; our oracle ignores DW_AT_abstract_origin. Compiler-attributable: $\leq 25\%$
Text	Counts: 220 $\rightarrow$ 132; later steps remove 24 and 10	215 $\rightarrow$ 132; later steps 22 and 6
Scoring	259 predictions scored against an empty name	Nameless copies of inlined functions entered as BRONZE; Token F1 = 0 by construction (5 % of rows)
Scoring	Key collision in the scoring API	DWARF IDs repeat across programs: 43 predictions scored against the wrong name, 7 silently dropped
Stats	GOLD band 0.20–0.32; Kendall’s W 0.66	Recomputed: 0.18–0.30; W $\approx$ 0.74 (0.77 for tier-hopping reproduces)

Unchanged: yield 87.2 $\rightarrow$ 65.9 %, the paired result (Holm $p \geq 0.28$ with corrected keys), shortlist bounds, semantic re-scoring, Ghidra determinism.

After the fixes: 3 of 7 models show no unpaired decline at all. The scoring artifact had inflated the very “decay” the paper warns about.

Found by tracing every function that left the evaluable set. Erratum and arXiv v3 in preparation; no new model runs needed.

What the pipeline can discover next

1 Recover lost functions

follow
DW_AT_abstract_origin; bring
back the 29, incl. the recursive
ones

2 Structure probes

program families varying one
construct (nesting, recursion,
goto, headers), paired across
levels

3 Fix the naming confound

purpose-based ground truth +
semantic metrics:
understanding, not naming
style

4 Evaluate harnesses

REA-style tools vs. bare
models on the same GOLD
functions, per function

5 Adversarial probes

which code patterns make
models miss or misread
behaviour (cf. Crawford et al.,
2026)

6 Scale and replicate

real corpora with yield +
reasons reported; Clang,
ARM64, a second decompiler

From benchmark to instrument: ask which functions, which structures and which tools make the difference, not only whether the task succeeded.

Limits, and what generalizes

Limits

- 15 synthetic programs, 226 functions
- GCC · x86-64 · Ghidra only
- Token F1 is lexical (semantic check: lower bound)
- Tiers mix alignment quality and complexity
- Shortlist gain is an upper bound
- Oracle misses abstract_origin functions (29)
- scoring-join bugs found after publication (fixed in v3)

What generalizes

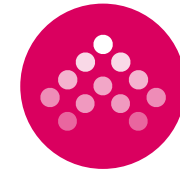
- Ground truth erodes with optimization
- The causes change with the level: header copies, then fragmentation
- Level averages compare different populations

These come from how compilation and alignment work. The absolute scores and the model order do not generalize.

Before evaluating how good LLMs are at reverse engineering, we need to know what the right answer still is, after compilation and decompilation.

Paper: [arXiv:2607.07738](https://arxiv.org/abs/2607.07738)

Code + data: github.com/NicolasKol/reforge



wbh
WILHELM BÜCHNER
HOCHSCHULE

Eine Hochschule der Klett Gruppe

wb-fernstudium.de

References

- Akita, F. (2026). AI reverse engineering: Is Adobe dead? akitaonrails.com, 13 Oct 2026.
- Al Jazeera (2026). How AI apps are offering free substitutes for Big Tech software, 14 Oct 2026.
- Anthropic (2025). Disrupting the first reported AI-orchestrated cyber espionage campaign.
- Caswell, B. et al. (2025). Project Ire autonomously identifies malware at scale. Microsoft Research Blog.
- Crawford, B., Phillips, J., McClure, P. (2026). Automatically attacking software RE AI agents. arXiv:2605.30667.
- Dramko, L. et al. (2024). A taxonomy of C decompiler fidelity issues. USENIX Security 24.
- Gao, Z. et al. (2025). DecompileBench: A comprehensive benchmark for evaluating decompilers in real-world scenarios. arXiv:2505.11340.
- Koller, N., Schmidt, A. U. (2026). REFORGE. arXiv:2607.07738.
- Korben (2026). REA: I tested AI agent reverse engineering. korben.info.
- Leger, M. A. et al. (2021). Exploring explicit uncertainty for binary analysis (EUBA). Sandia, SAND2021-14600.
- Liu, C. et al. (2024). Assemblage: Automatic binary dataset construction for ML. NeurIPS 37.
- Liu, C., Raff, E., Micinski, K. (2026). When LLM decompilers recompile more and preserve less. arXiv:2609.05370.
- Spence, J. et al. (2026). A realistic, contamination-free reverse engineering benchmark (SRE-Bench). arXiv:2608.11469.
- Startup Fortune (2026). An AI agent that reverse engineers any software just hit number one on GitHub, 9 Oct 2026.
- Tan, H. et al. (2025). Decompile-Bench: Million-scale binary-source function pairs. arXiv:2505.12668.
- Know Your Meme (2026). AI video game merging. / Decrypt (2026). AI video game mod mash-ups.

Case 1: bubble_sort disappears

t02_shared_header_macros · sorting.c:5 · bubble_sort()

Level	Status	Size	Model answers (7 models)
-00	GOLD	66 instr.	bubble_sort ×4 (F1 1.0) · bubble_sort_array · bubble_sort_int_array · sort_integers
-01	gone	–	inlined into its caller: no separate function in the binary
-02	gone	–	–
-03	gone	–	–

The benchmark loses an item the models solve almost perfectly. Its code now sits inside the caller, whose name no longer describes everything it does. No model got worse; the item list changed.

Case 3: run_bytecode, ground truth degrades

t13_goto_labels · interpreter.c:36 · run_bytecode()

Level	Instr.	Cyclomatic	Tier	Join 1 overlap / candidates	Typical answer
-O0	482	57	GOLD	1.00 / 1	interpret_bytecode (F1 0.5)
-O1	261	43	GOLD	1.00 / 1	interpret_bytecode / execute_bytecode
-O2	189	37	SILVER	0.75 / 2 (not unique)	interpret_bytecode / bytecode_interpreter
-O3	189	37	SILVER	0.75 / 2 (not unique)	interpret_bytecode / execute_bytecode

Ground-truth confidence fell, the answers did not. A GOLD-only benchmark drops this function at -O2, although the models name it consistently.

Case 4: ackermann goes missing

t06_recursion_inline · math_recurse.c:30 · ackermann()

Level	Size	Tier	What happened	Models naming it “ackermann”
-00	137 B	GOLD	clean match	3 of 7
-01	104 B	BRONZE	overlap 0.68, 4 candidates	3 of 7
-02	836 B	none	recursion partly inlined into itself	not evaluated
-03	1,198 B	none	still in binary: FUN_00401880, 329 instr.	not evaluated

Why lost: DWARF splits it into an abstract entry (name, no addresses) and a concrete copy pointing back via DW_AT_abstract_origin. Our oracle reads only the first.

The hardest version of the function silently left our benchmark. 29 functions are affected. We found it only because every decision is logged.

B1 Cheat sheet: binary and debug information

Term	Meaning
ELF	Linux executable format; holds machine code, data, optionally symbols and debug info
-O0 ... -O3	GCC optimization levels: none → aggressive (inlining, code motion, dead-code elimination)
Stripping	strip --strip-all removes symbols and debug info; machine code stays byte-identical
DWARF	debug-info format written by gcc -g; a tree of entries (DIEs) describing the program
DW_TAG_subprogram	DIE for one function: name, address range, declaration file/line/column
low_pc / high_pc, DW_AT_ranges	start/end address of a function; ranges = several fragments (after inlining)
.debug_line	line table: maps machine-code addresses to source file and line
DW_AT_inline / abstract_origin	inlined function: abstract entry (name, no addresses) + concrete copies pointing back
DECLARATION_ONLY / MISSING_RANGE	DIE is only a declaration / has no address range → no tier (NO_RANGE)
FUN_00401880	Ghidra's auto-name for an unnamed function at that address

B2 Cheat sheet: alignment, joins and reasons

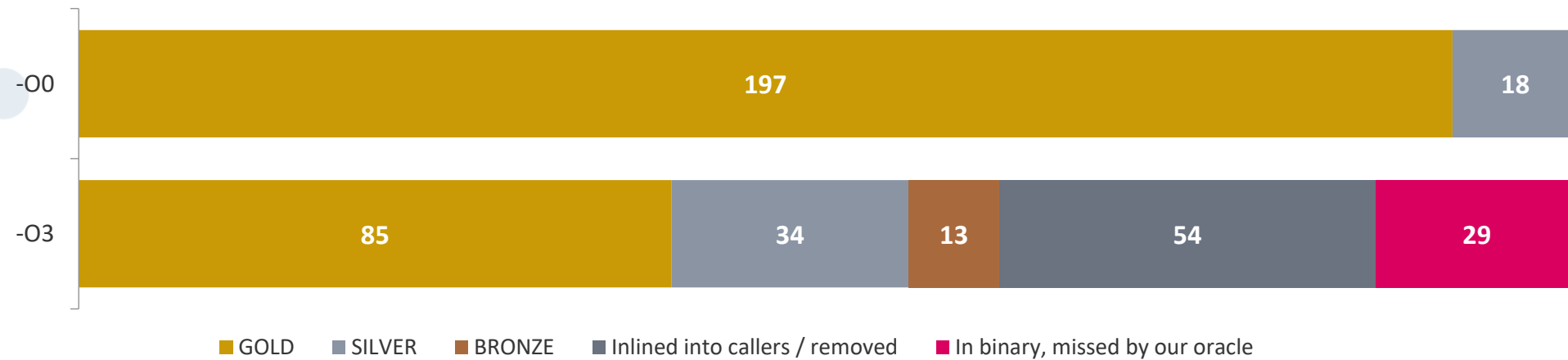
Term	Meaning
gcc -E, .i file	preprocessed source: includes and macros expanded; line markers keep original file:line
Translation unit	one .c file after preprocessing; a header's functions appear in every TU that includes it
Tree-sitter, span, context_hash	parser for C; per function definition: its span and a hash of its normalized text
overlap ratio	DWARF line rows inside a candidate's span $\div$ all DWARF line rows of the function
MATCH / AMBIGUOUS / NO_MATCH	best candidate ≥ 0.7 and unique / near-tie (≤ 0.02) or header copy / none above threshold
HEADER_REPLICATION_COLLISION	identical candidates from different TUs (same header function copied)
PC_LINE_GAP, LOW_OVERLAP_RATIO	some line rows unexplained by the best candidate / overlap below threshold
n_candidates	source functions with any overlap; gate 3 requires exactly 1
JOINED_STRONG / JOINED_WEAK	DWARF range $\leftrightarrow$ Ghidra function overlap ≥ 0.9 / > 0 but < 0.9
hc_reject_reason	first failed check: ALIGN_NOT_MATCH, ALIGN_NOT_UNIQUE, ALIGN_RATIO_LOW, NOT_JOINED_STRONG, ...
Stable key	(test case, decl file, decl line, decl column, normalized name)

B3 Cheat sheet: metrics and statistics

Metric	Meaning
High-confidence yield	share of gold-eligible functions passing all 8 checks (197/226 = 87.2% at -O0)
Token F1	split names into lowercase sub-tokens; harmonic mean of token precision and recall
Exact match	identical after lowercasing and removing separators
Top-K oracle / uplift	best of K candidates (picked with the answer) minus rank-1 score; ≥ 0 , upper bound
Paired Δ	F1 at -O0 minus F1 at -O3 for the same function (stable key)
Wilcoxon signed-rank	rank-based test for paired, non-normal differences (one-sided: decline?)
Holm correction	adjusts p-values for 7 tests; controls chance of any false alarm
Bootstrap 95% CI	resample functions 10,000× to get an interval for the mean Δ
Kendall's W	agreement of 7 models' rankings of functions (0 none, 1 identical)
AUC	chance a true name pair scores above a random pair (0.5 = coin flip)
bge cosine, noise floor	semantic similarity; random pairs: mean 0.53, 95th percentile 0.65

B4 Backup: 00-03 fate

Fig. 3: Fate of the 215 functions evaluable at -O0 (count)



75–96% of surviving functions keep their tier. Attrition removes functions, it does not degrade the survivors.

B5 Backup: is Ghidra deterministic?



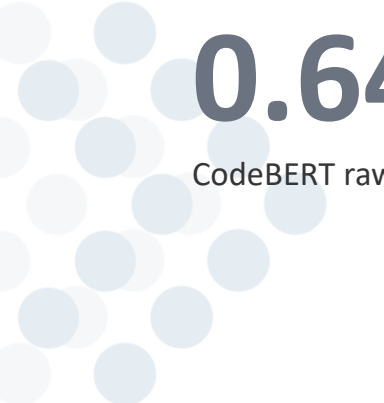
1,884 / 1,884

functions byte-identical across two runs (Ghidra 12.0.3, pinned container)

1,884 / 1,884

identical to the original output months earlier, on another machine

B6 Backup: why not CodeBERT?



0.64

CodeBERT raw (AUC)

0.66

CodeBERT whitened

0.78

bge-small-en-v1.5

AUC: how often a true name pair scores above a random pair (0.5 = coin flip).