

Humans vs. Machines: A Comparison With AI Agents in Software Reverse Engineering

SAMUELE DORIA, University of Padua, Italy

STEFANO DE ROSA, EURECOM, France

RUOYU FISH" WANG, Arizona State University, USA

ZION LEONAHENAHE BASQUE, University of Georgia, USA

ELEONORA LOSIOUK, University of Padua, Italy

SIMONE AONZO, EURECOM, France

Can an autonomous AI agent reverse engineer a binary as well as a skilled human analyst? If so, do they adopt the same analysis strategy, and can agents scale to real-world programs?

We study these questions through a controlled comparison of human and LLM-based software reverse engineering (SRE). We evaluate nine models on two compact, static-analysis reverse-engineering challenges previously solved by human analysts, using a common understanding rubric, and record their complete interactions with IDA Pro. This lets us compare not only *what* humans and agents recover, but also *how* they investigate a binary.

The best-performing agents (GPT-5.5 Pro, Claude Opus 4.8, and Kimi K3) achieve statistically human-competitive understanding on both tasks, while weaker models remain substantially behind. Ablation studies reveal that this gap is shaped by both the code representation and the demands of the workflow, rather than by model capability alone: while open-weight models improve substantially when provided with richer code representations, we detect no significant change for frontier models. Yet, these similar success rates hide fundamentally different workflows. Humans repeatedly revisit functions and follow local program structure; strong agents instead jump aggressively across the binary, retrieve information broadly, and rarely return to previously inspected code, a behavior we call *greedy fetching*. This strategy is not fixed: when agents receive semantically richer code, they fetch less of it without losing understanding, showing that agent behavior is strongly shaped by the information interface.

Because greedy fetching relies on aggregating massive amounts of retrieved code into the agent's memory, it raises a critical scalability question. To test these limits, we measure context-window saturation across 4,000 real-world binaries and find that, although large context windows accommodate many programs, larger applications and lower-level representations quickly consume this capacity.

Our results show that autonomous reverse engineering is becoming competitive, with the enormous advantage of speed, but scaling it will require better navigation, selective retrieval, and persistent analysis state, not merely larger context windows.

CCS Concepts: • **Security Usability and Measurement** → **Usable security and privacy**.

Additional Key Words and Phrases: Reverse Engineering, Large Language Models, Binary Analysis

Authors' Contact Information: Samuele Doria, University of Padua, Padua, Italy, samuele.doria.1@studenti.unipd.it; Stefano De Rosa, EURECOM, Biot, France, Stefano.De-Rosa@eurecom.fr; Ruoyu Fish" Wang, Arizona State University, Tempe, USA, fishw@asu.edu; Zion Leonahenahe Basque, University of Georgia, Athens, USA, zionbasque@uga.edu; Eleonora Losiouk, University of Padua, Padua, Italy, eleonora.losiouk@unipd.it; Simone Aonzo, EURECOM, Biot, France, simone.aonzo@eurecom.fr.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

FSE '27, Shenzhen, China

© 2027 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-XXXX-X/2027/07

<https://doi.org/XXXXXXX.XXXXXXX>

ACM Reference Format:

Samuele Doria, Stefano De Rosa, Ruoyu Fish” Wang, Zion Leonahenahe Basque, Eleonora Losiouk, and Simone Aonzo. 2027. Humans vs. Machines: A Comparison With AI Agents in Software Reverse Engineering. In *Proceedings of ACM International Conference on the Foundations of Software Engineering (FSE ’27)*. ACM, New York, NY, USA, 22 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 Introduction

Modern security practice depends on analyzing software that was not written, documented, or compiled for the analyst: malware, proprietary applications, legacy components, and third-party programs often arrive only as binaries, yet analysts must still secure and understand them. This is the role of Software Reverse Engineering (SRE): recovering meaning from programs whose semantics have been removed or were never available, a critical process that remains labor-intensive, primarily human-driven, and dependent on expertise [25, 30, 45].

Prior work has shown that Large Language Models (LLMs) offer a promising way to reduce this bottleneck. Recent studies have explored their use across SRE-related tasks, including code interpretation, function and symbol recovery, binary understanding, vulnerability analysis, and deobfuscation [8, 12, 15, 17, 18, 21, 22, 26, 31, 34, 37, 38, 42, 46–48], and human–LLM teaming has been found to improve SRE performance [4].

Recently, tool-using LLM systems (hereafter *agents*) have moved LLM-based security analysis from interactive assistance toward increasingly autonomous execution [7, 20, 23, 29]. In SRE, this shift raises a question that goes beyond whether an agent can simply solve a challenge: can an AI agent reach a program understanding comparable to that of a human reverse engineer? What strategies does it use to do so? Can those strategies scale to larger programs? While recent studies demonstrate substantial progress in autonomous binary analysis and security-agent evaluation [23, 39, 41], they typically evaluate agents against fixed suites of challenge binaries, scoring them purely on pass/fail flag recovery or scripted “golden” paths, rather than comparing how humans and agents investigate the program. The question of whether agents navigate and understand binaries using the same analytical strategies as humans remains largely unexplored in a controlled SRE setting.

SRE provides a particularly useful setting for such a comparison because the final answer is only part of the analysis: the functions inspected, relationships followed, and information revisited expose how an analyst progressively builds understanding. We therefore conduct a matched, instrumented comparison in which autonomous agents and human reverse engineers analyze the same binaries under a common understanding rubric, mapping their tool interactions into shared SRE abstractions to compare *what* they recover, *how* they navigate a binary, and whether the resulting strategies remain feasible as program size grows.

Our study is guided by four research questions that jointly assess the capability, limitations, behavioral strategies, and scalability of autonomous SRE agents compared with humans:

- **RQ1: Do agents perform reverse engineering at the same competency level as human experts?** We evaluate whether autonomous agents can reach a practically non-inferior level of understanding relative to a top-human reference cohort.
- **RQ2: What limits agent reverse-engineering performance?** We use ablation studies to examine which failure modes, representation effects, and interaction bottlenecks contribute to the gap between frontier and open-weight agents.
- **RQ3: How do agents’ and humans’ strategies differ?** We turn the workflow-related limitations identified in RQ2 into a strategy explanation by comparing how successful agents and humans move through binaries at the macro level (transitions between high-level analysis phases) and micro level (call-graph navigation).

- **RQ4: How does program size affect the feasibility of full-context analysis?** We evaluate the practical feasibility of observed agentic strategies by measuring how often complete decompiled or disassembled representations saturate available context windows across real-world software.

To answer these questions, we present, to our knowledge, the first controlled human-agent comparison in SRE that directly aligns challenge binaries, an understanding rubric, and behavioral traces, using an instrumented pipeline that repeatedly runs each agent through a standardized SRE framework interface (IDA Pro via MCP) and logs all tool interactions. By scoring agents' writeups on the identical challenges and the same understanding-point system used in a prior human study [4], we examine both *what* agents achieve and *how* their workflows differ from human analysts'.

Human-competitive understanding turns out to be strongly model-dependent: in a one-point non-inferiority test against the top human reference, GPT-5.5 Pro, Claude Opus 4.8, and Kimi K3 achieve top-human-level understanding on both challenges, whereas other open-weight models achieve it only on one challenge. Ablations show why: providing complete decompiled programs directly yields substantially larger gains than returning source code over the same MCP loop, especially for open-weight models, while disassembly-only inputs help less: representation quality, search burden, and workflow demands all contribute. Trace analysis shows this success does not stem from human-like navigation: agents cover similarly broad portions of the binary but jump nonlinearly and revisit far less often, a behavior we call *greedy fetching*. Finally, a context-window feasibility analysis over real-world binaries bounds this strategy: full-context analysis remains feasible for many binaries, especially with decompiled representations, but becomes unreliable as size grows or representation density worsens.

Contributions. This paper makes the following contributions:

- To our knowledge, we present the first controlled, end-to-end human-agent SRE comparison that aligns the binaries, understanding score rubric, and interaction traces, enabling direct comparison of both outcomes and analysis strategies. We build this comparison on top of the challenges, rubric, and human writeups of Basque et al. [4], which we use to drive the human-agent behavioral comparison itself.
- We show that agents can achieve human-level understanding, but this capability is model-dependent: GPT-5.5 Pro, Claude Opus 4.8, and Kimi K3 are non-inferior to the top human reference on both compact challenges within a one-point margin, while the remaining six evaluated agents perform substantially less consistently.
- We identify *greedy fetching*, a distinctive agent SRE strategy: rather than repeatedly revisiting and refining previously-inspected code, strong agents aggressively fetch new information across the binary. We show that this strategy achieves coverage comparable to humans.
- We quantify the scalability limits of full-context SRE using 4,000 real-world binaries. Even with a 1M-token context window, decompiled programs exceed the available context in 1.9%–39.1% of binaries, depending on the binary group; disassembly further increases the required context by roughly 2×.

Overall, our findings show that the strongest agents can achieve top-human outcomes on compact SRE tasks, but this capability is heterogeneous and stems from a navigation policy that differs substantially from that of unassisted human reverse engineering. Scaling that capability beyond favorable settings will require selective retrieval, persistent state, and more deliberate navigation rather than indiscriminate full-context acquisition.

2 Related Work

Human Reverse Engineering Workflows. Votipka et al. [45] characterize reverse engineering as iterative exploration, hypothesis formation, and validation, while Mantovani et al. [25] analyze humans' cognitive strategies and information needs. Aonzo et al. [2] compare human and machine decisions in malware classification. Closest to our human data source, Basque et al. [4] study human-LLM teaming in SRE using writeups and IDA logs from 48 participants. They define *understanding rate* as the score divided by solve time, show that it discriminates self-reported experts from novices, and use rate-ranked top- and bottom-ranked cohorts for strategy analysis. We reuse this performance construct for the RQ1 human reference, but evaluate autonomous agents rather than human-LLM teams.

LLMs for Software Reverse Engineering. Prior studies test LLMs for decompiled-code interpretation [31, 34, 37], decompiler refinement [17], function-name and symbol recovery [21, 46–48], and broader binary understanding [18, 38]. Representation quality (including pseudocode, intermediate representations, and recovered semantic information) strongly affects usefulness [17, 32]. Work on vulnerability analysis and assembly deobfuscation similarly shows that source-level competence does not transfer directly to binary settings [8, 15, 26, 42]. These studies establish useful SRE subtasks but provide less evidence about end-to-end tool-mediated workflows.

Agentic Reverse Engineering. BINREX [23] is a fully automated static binary-analysis agent combining semantic retrieval, hierarchical planning, IDAPython synthesis, and validation, evaluated across 72 BinREval tasks, additional benchmarks, a human study, and industrial malware deployment, substantially broader practical evidence than our two-task benchmark. We therefore do not claim the first autonomous binary-analysis agent or broadest evaluation; our contribution is an aligned human-agent experiment on the same binaries and behavioral measurements. ClearAgent [7] exposes analyzer-friendly tools for iterative vulnerability analysis; Nishizaka et al. [29] evaluate three agents on 24 CTF tasks and identify failures in observation, comprehension, and planning; ReCopilot [6] specializes a model for binary analysis using data-flow and call-graph context; and Radey et al. [35] discuss limitations including context, obfuscation, architectures, and guardrails. These works motivate our focus on tool-mediated behavior and failure modes.

Human-Agent Comparison and Analysis Behavior. Prior work has compared humans and autonomous agents in security tasks and examined the sequence of actions they take: CTF-Judge/CTFTiny [39] evaluates agent behavior step by step across 50 CTF challenges, while Tang et al. [41] compare autonomous agents with human teams in a live CTF setting. We therefore do not claim the first human-agent comparison in security; our contribution is specific to SRE: humans and agents analyze the same binaries, their IDA and MCP interactions are converted into common behavioral measures, and their final understanding is evaluated using the same rubric, capturing what they recover as well as how they navigate, inspect, and revisit a program. LLM assistance has also been studied for code understanding and higher-level software summaries [28, 36], with known difficulties in prompt construction and intent alignment [1, 49, 50], and security surveys likewise report useful but unreliable reasoning beyond narrow settings [14, 40, 44]. While prior MCP evaluations focus on general tool selection and argument-syntax failures [24], our work investigates domain-specific tool orchestration within reverse-engineering workflows and directly benchmarks agent behavioral traces against a performance-calibrated human reference on identical binaries.

3 Experimental Setup

We use an instrumented pipeline for autonomous SRE and reuse the challenges, writeups, IDA logs, and scoring criteria of Basque et al. [4], aligning agent and human measurements on the same tasks.

3.1 Statistical Methodology and Reporting

All tests use a significance threshold of $\alpha = 0.05$. Where we run many tests at once, we adjust raw p -values with the Benjamini–Hochberg (BH) correction [5] within each prespecified family of tests (adjusted values are denoted q). Effect sizes are Cohen’s d (Hedges’ g for $n < 20$) or Cliff’s δ [3, 9, 16]; outside RQ1 and the RQ2 ablations, tests are selected as in Basque et al. [4].

RQ1: non-inferiority. A difference test cannot show that two groups are the same. We therefore test whether an agent can be ruled out as being more than $\Delta = 1$ understanding point worse than the top-human reference: with $D = \mu_{\text{agent}} - \mu_{\text{human}}$, we test $H_0: D \leq -1$ against $H_1: D > -1$ using a one-sided Welch test (Welch–Satterthwaite degrees of freedom). If significant, the agent is *non-inferior*. For a tier of M agents, the estimand is the mean of agent means, with variance $\sum_m s_m^2 / (M^2 n_m)$ plus the human variance s_H^2 / n_H , so each agent’s own run variance is preserved. The 18 agent-level tests (9 agents $\times$ 2 challenges) form one BH family.

RQ2: exact small-sample tests. Each ablation compares five baselines with five ablation runs per model and challenge. With ten observations, we use an exact permutation test: we enumerate all $\binom{10}{5} = 252$ splits of the pooled scores into two groups of five and compute how often a split yields a mean difference at least as large as the observed one. We also report Cliff’s δ ($\delta = +1$: every ablation score exceeds every baseline score). The 18 source-code comparisons and the 90 static no-MCP comparisons (nine models $\times$ five representations $\times$ two challenges) form separate BH families. For claims about model groups (e.g., open-weight), we permute condition labels only within each model’s own runs and average the per-model differences, so each model has equal weight. Claims that an ablation reaches the human reference require both a corrected improvement over its baseline and corrected non-inferiority as in RQ1; claims that an ablation *maintains* performance use two one-sided tests (TOST) with a ± 1 -point equivalence margin.

More details about our statistical methodology can be found in our repository (Section 12).

3.2 Dataset and Challenges

We evaluate two CTF-style challenges, RPC and Secrets, each with seven understanding points from prior work [4], where 81% of participants (39/48) moderately or strongly agreed the tasks were representative examples of real-world software. They exercise common SRE activities, including protocol recovery, cryptographic reasoning, path analysis, and vulnerability discovery.

RPC implements a custom protocol combining Base64, run-length encoding, and TEA encryption. Solving requires recovering the admin identity, exploiting time-seeded password generation, and issuing the sequence of operations needed to retrieve /flag. Secrets implements an AES-ECB encrypted file manager; path traversal exposes the admin hash, after which weak key derivation can be reversed to decrypt the admin flag file.

3.3 System Architecture

Agents access IDA Pro 9.3 through the same model-neutral MCP harness and its `idalib` headless interface. Every model receives the same 61-tool schema, task-prompt structure, logging layer, and reverse-engineering interface, ensuring that differences in performance reflect the models themselves rather than differences in the surrounding infrastructure. Our instrumented IDA Pro MCP plugin [19] logs each tool call, its arguments, outputs, and timestamps; each run yields an

action trace and a final writeup. The simple prompt template supplies the challenge description but no analysis strategy, keeping the setting close to the prior human study [4]. As in that human study, the toolset is static analysis only: decompilation, disassembly, cross-references, and data/structure inspection, with no debugger or dynamic/runtime execution path, keeping the agent and human conditions matched.

3.4 Candidate Agents

We used nine LLMs: *frontier* (leading state-of-the-art) and *open-weight* (trained parameters publicly available for download and local execution) models, organized by deployment classes. Frontier models form the class *Frontier* (OpenAI GPT-5.5 Pro, Claude Opus 4.8, Google Gemini 2.5 Pro), while open-weight models are organized in: *Single-GPU* (Qwen 3.5 9B, Gemma 4 31B), *Multi-GPU* (Llama 3.3 70B Turbo, GPT-OSS 120B), and *Large-Scale Accelerator* (DeepSeek V4 Pro, Kimi K3). Exact model API identifiers and access dates are in our repository (Section 12). All frontier models are invoked in their default generation mode (without extended-thinking or specialized reasoning tokens). Open-weight models are served through Together AI using tool-calling APIs; we note that some of these models may be subject to backend quantization by the provider [43]. All runs use a temperature of 0, making them as deterministic as possible.

3.5 Automated Writeup Evaluation

Every model–challenge configuration is repeated five times, making manual grading impractical at our scale: the 630 agent writeups (90 baseline MCP runs, 90 source-code-replacement runs, and 450 static no-MCP runs; nine models $\times$ two challenges $\times$ five repetitions per condition) contain $630 \times 7 = 4,410$ understanding points to check. We therefore use GPT-5.1 to evaluate each writeup, both the agent-generated writeups and the expert-labeled human writeups, against the seven boolean understanding questions for each challenge. For every evaluated answer, GPT-5.1 also returns a confidence score. We validate this grader against 96 expert-labeled human writeups [4] that were manually validated, as well as through a manual audit of 200 predictions from the full set of evaluated agent writeups, described next.

For the manual audit, we follow Zheng et al. [51] and sample 200 agent predictions. We stratify the samples by challenge, deployment class (frontier vs. open-weight), predicted label, and confidence stratum, where confidence is divided at a calibrated threshold of 0.93 obtained via Platt scaling [33]. On the held-out expert-labeled human writeups, which serve as ground truth for this calibration, predictions above this threshold achieve 96.45% precision. This stratification ensures that both reliable and uncertain predictions are represented. Samples are allocated proportionally, with at least two samples per cell, using a fixed random seed. Three independent annotators reached substantial agreement (Fleiss' $\kappa = 0.76$). Measured against their consensus, GPT-5.1 achieves 94.5% overall accuracy (F1= 0.94) and 96.6% accuracy on the high-confidence stratum. These results support the use of automated grading for the full evaluation.

4 RQ1: Do Agents Perform Reverse Engineering At The Same Competency Level As Human Experts?

RQ1 evaluates whether autonomous agents understand binary programs as well as the strongest human reverse engineers from the prior study.

Methods. We compare agents and humans using the same seven-point understanding rubric from prior work [4]. Each agent is run five times per challenge (RPC and Secrets) to capture run-to-run variability.

Our human reference is performance-based rather than solely on participants' self-reported expertise labels. Basque et al. define *understanding rate* as understanding score divided by solve

time and validate it as a meaningful SRE-performance measure: in their human study, self-reported experts have significantly higher understanding rates than novices ($p = 0.001$, reported effect size 0.57) [4]. Their strategy analysis likewise ranks participants by understanding rate and contrasts top- and bottom-performing cohorts. Using that validated construct, we rank the 48 participants separately for each challenge based on understanding rate. For each challenge, we use the 10 highest-rated participants as the *top-human reference* and the 10 lowest-rated participants as a lower descriptive reference. The actual comparison with agents, however, is based solely on the raw *understanding score*, the total number of rubric points achieved, regardless of time: the rate tells us *who* the best humans are, while the non-inferiority test asks whether an agent's *final understanding* is practically no worse than what those humans achieved.

We organize agents into four RQ1 analysis groups: *Frontier*, *Single-GPU*, *Multi-GPU*, and *Large-Scale Accelerator*. The primary analysis compares the Frontier agents group with the top-human reference using the non-inferiority test described in Section 3.1: we test whether the agent group's understanding score is within $\Delta = 1$ rubric point of the top-human reference. We choose this margin because one point corresponds to a single item in the understanding rubric and represents a realistic difference even among skilled human reverse engineers: in the prior human study [4], high-performing and expert participants did not always recover every understanding item, and one-point omissions occurred even among strong performances. We therefore treat an agent as practically non-inferior only when the data allow us to rule out a deficit greater than one rubric point relative to the top-human reference.

We aggregate scores for each agent tier by computing the unweighted arithmetic mean across constituent agents (i.e., assigning equal weight $1/M$ to each of the M agents in the tier). Because our design is balanced (five runs per agent per challenge), this equals the raw mean of all runs in the group. The uncertainty of this estimate, however, is not computed by pooling runs: the variance of the tier mean is the sum, over the M agents, of each agent's own variance contribution $s_m^2/(M^2 n_m)$ (where s_m^2 is the sample variance of agent m 's scores and n_m its number of runs on the challenge), which we combine with the variance of the human reference in a Welch–Satterthwaite test (see Section 3.1). This preserves inter-agent heterogeneity instead of treating all runs in a group as a single homogeneous source.

Results. Table 1 summarizes agents' average understanding scores and the understanding scores of the rate-selected human reference cohorts. The top-human reference scores 6.7 ± 0.5 on RPC and 5.5 ± 0.7 on Secrets. The remaining 28 humans score 4.2 ± 1.1 and 3.3 ± 0.8 , respectively, while the bottom-human reference scores 1.7 ± 0.5 and 0.7 ± 0.5 . Among agents, GPT-5.5 Pro, Claude Opus 4.8, and Kimi K3 achieve the highest scores, whereas the Single-GPU and Multi-GPU groups perform substantially lower under the MCP workflow.

We denote the mean score difference between agents and the top-human reference as $D = \mu_{\text{agent}} - \mu_{\text{human}}$; a negative D means agents scored lower on average. Non-inferiority is established when we can rule out that D is worse than -1 (i.e., that agents are more than one point behind). At the group level, Frontier agents average 5.20 on RPC versus 6.70 for the top-human reference ($D = -1.50$, i.e., agents are 1.5 points behind). The 95% one-sided lower confidence bound is -2.21 , which falls below the -1 boundary, meaning we cannot rule out that agents are more than one point worse. Non-inferiority is therefore not established ($p = 0.889$). On Secrets, Frontier agents average 5.07 versus 5.50 for the top-human reference ($D = -0.43$). The lower confidence bound is -0.92 , which remains above the -1 boundary, meaning we *can* rule out that agents are more than one point worse. Non-inferiority is therefore established ($p = 0.0288$). Thus, because the Frontier tier fails to establish non-inferiority on RPC, it does not demonstrate human-level understanding across both challenges.

Table 1. Average understanding scores and standard deviations on RPC and Secrets. Human cohorts are selected by understanding rate. Agent rows are organized by the RQ1 analysis groups.

Type	Agent/Group	RPC	Secrets
Human	Top 10 reference	6.7 ± 0.5	5.5 ± 0.7
	Remaining 28	4.2 ± 1.1	3.3 ± 0.8
	Bottom 10 reference	1.7 ± 0.5	0.7 ± 0.5
Single-GPU	Qwen 3.5 9B	0.0 ± 0.0	2.4 ± 1.5
	Gemma 4 31B	3.2 ± 1.3	3.4 ± 1.5
Multi-GPU	Llama 3.3 70B Turbo	0.0 ± 0.0	2.0 ± 0.0
	GPT-OSS 120B	0.2 ± 0.4	2.0 ± 1.2
Large-Scale Accelerator	DeepSeek V4 Pro	3.6 ± 1.7	2.2 ± 1.5
	Kimi K3	6.6 ± 0.5	6.2 ± 0.8
Frontier	Gemini 2.5 Pro	2.2 ± 2.2	3.2 ± 0.8
	Claude Opus 4.8	6.6 ± 0.5	6.0 ± 0.7
	GPT-5.5 Pro	6.8 ± 0.4	6.0 ± 0.0

Looking at individual agents rather than tier averages reveals important differences that the group-level analysis obscures. After BH correction across the 18 per-agent comparisons, GPT-5.5 Pro is non-inferior on both RPC ($q = 0.0087$) and Secrets ($q < 0.001$) with respect to the top-human reference; Claude Opus 4.8 is also non-inferior on both ($q = 0.024$ and 0.013 , respectively). Kimi K3, although in the Large-Scale Accelerator group, likewise satisfies non-inferiority on both challenges ($q = 0.024$ and 0.013), while Gemini 2.5 Pro does not satisfy the criterion for either challenge. None of the Single-GPU, Multi-GPU, or Large-Scale Accelerator groups establishes non-inferiority as a tier, showing that deployment class alone is not a sufficient predictor of competency. Efficiency provides an additional perspective on these results. Across the baseline runs of the three models that establish non-inferiority on both challenges (GPT-5.5 Pro, Claude Opus 4.8, and Kimi K3) mean solve time is 628 s (10.5 min) on RPC and 666 s (11.1 min) on Secrets, versus 4,498 s (75.0 min) and 2,797 s (46.6 min) for the high-performing human cohort: successful agents complete the analysis approximately $7.2\times$ and $4.2\times$ faster, respectively.

RQ1: Do Agents Perform Reverse Engineering At The Same Competency Level As Human Experts?

Yes, but only for specific models. In this controlled two-task setting, GPT-5.5 Pro, Claude Opus 4.8, and Kimi K3 each achieve understanding scores that are statistically non-inferior to the top human reference on both challenges, meaning their performance is within one rubric point of the strongest human reverse engineers on these tasks. Importantly, this comparable understanding is achieved much faster: these agents complete the tasks approximately $4.2\times$ – $7.2\times$ faster than the high-performing humans.

5 RQ2: What Limits Agent Reverse Engineering Performance?

Although several high-performing agents reach the top-human reference, many agents, particularly those using small open-weight models, perform poorly. We investigate these differences through trace analysis and two complementary ablations: one changes the semantic quality of the code returned via the same MCP loop by providing the original source code, while the other supplies the entirety of the decompiled and disassembled code without MCP interaction.

5.1 Failure Modes and Tool Usage Patterns

Methods. We analyze *action traces*, i.e., chronological logs of tool names, arguments, plugin responses, and timestamps. We focus specifically on runs that scored zero points, separating cases where the agent fails to use the tools at all from cases where it uses them but still fails to derive useful understanding from the retrieved information.

Context saturation. One possible explanation for failure is context saturation, meaning that the accumulated session history exceeds the agent’s effective context window. We therefore measure both baseline saturation, caused by the fixed prompt and 61-tool schema, and final saturation at the end of each session, which is related to tool usage and data ingestion. Overall, saturation remains low across all window sizes. Models with the smallest windows (Llama 3.3 and GPT-OSS, 131k) end sessions at 5.3%–9.2%. The worst case is Gemma 4 (256k), which doubles from 7.2% to 14.1% on RPC, the highest average final saturation of all agents. Qwen3.5 9B barely moves (7.2% → 7.4%), consistent with sessions that end after very few tool calls. Mid-size windows (DeepSeek V4 Pro, 512k) end at about 7.5%, while agents with $\geq 1\text{M}$ -token windows, including the strongest ones (GPT-5.5 Pro, Kimi K3), never use past 6% of their context budget by session end. The full per-model table is available in our repository (Section 12). Since no average final saturation exceeds 14.1%, the raw context window size does not explain the baseline failures on these compact tasks.

Correlation between tool usage and understanding score. Frontier agents average 26.1 tool calls across the two challenges, while open-weight agents average 14.4. Spearman correlations show that higher tool use is associated with higher understanding scores for open-weight agents (RPC: $p = 1.9 \times 10^{-8}$, $\rho = 0.8$, very large effect size; Secrets: $p = 0.0002$, $\rho = 0.6$, large effect size) but not for frontier agents (RPC: $p = 0.0851$; Secrets: $p = 0.6457$). Therefore, tool volume alone is insufficient: productive exploration requires converting returned information into stable progress.

Tool-calling failures. Among the 17 zero-score runs, 70.6% make only minimal, non-investigative use of the available tools before terminating, issuing 2 or less actionable tool calls. In these runs, the agent nonetheless returns a full writeup, but its claims are hallucinated, given the little or no evidence it actually gathered, rather than grounded in the code it inspected. This failure mode appears exclusively in open-weight agents; frontier agents always engage substantively with the MCP interface, even in poor runs. The remaining zero-score traces are active but unproductive: they make more than 2 tool calls yet still fail to translate the returned information into progress. Thus, weak runs include both *tool-orchestration collapse* (minimal, superficial tool use paired with an unsupported, hallucinated writeup) and engaged sessions that either fail to exploit the information they retrieve or fail to submit a final answer. Frontier agents concentrate more of their tool budget on code reading (decompile covers 31.4% of their total tool calls), whereas failing agents overuse `get_string`, which accounts for 33.7% of their total tool calls. This trace evidence motivates the controlled representation and interface ablations below, but by itself does not identify whether the underlying difference is reasoning ability, planning, interface adaptation, or their interaction.

5.2 Binary Representation Effects

Methods. We first test whether the semantic quality of the returned code affects performance while retaining the interactive workflow. We patch the IDA Pro plugin so that `decompile` tool call returns the original function source code rather than decompiled pseudocode, while leaving the models, prompt, tool schema, and MCP interaction loop unchanged. We compare each model–challenge condition with its five baseline MCP runs using a two-sided permutation test on the mean score difference (as detailed in Section 3.1), testing both each model individually and population-level effects across all models or sub-groups (open-weight, Frontier). Because we test nine models across

two challenges, this yields 18 individual comparisons, corrected together as a single BH family, with q denoting the BH-corrected p -value. We use a model-stratified exact permutation test that gives each model equal weight (so that a single strong or weak model does not dominate the group result) and apply BH correction across the six reported group–challenge tests (All models $\times$ 2 challenges, Open-weight $\times$ 2, Frontier $\times$ 2).

Results. Source-code replacement produces a positive aggregate shift in understanding score. Across all nine models, mean performance increases by 0.80 points on RPC ($q = 0.0120$) and 0.58 on Secrets ($q = 0.0497$). The clearest sub-group effect is among open-weight models on RPC (+0.67, $q = 0.0208$); the corresponding Secrets estimate is +0.77 but does not remain significant after correction ($q = 0.0538$). The Frontier group effect is likewise not significant after correction.

When we look at each of the 18 individual model–challenge comparisons (nine models $\times$ two challenges) separately, none remains significant after BH correction, so we do not attribute the aggregate representation effect to any particular model. Because this experiment changes only the code returned by each tool call while keeping the interactive workflow the same, it more closely isolates the effect of code quality than the no-MCP ablation below, which changes multiple aspects of the analysis at once.

5.3 Interface and Full-Code Access

Methods. We next remove the MCP interaction and provide all the decompiled and disassembled code in the prompt. We evaluate five representations against the same five baseline MCP runs: decompiled pseudocode from IDA Pro, Ghidra, r2dec, and Angr, and IDA disassembly. This intervention changes several parts of the analysis at once: the model no longer needs to choose and recover from tool calls, it spends less effort searching, and the information is presented differently in context.

The no-MCP experiment tests nine models on five representations across two challenges, producing $9 \times 5 \times 2 = 90$ individual comparisons. Running these many tests increases the risk of false positives: at a 5% significance level, we would expect approximately four or five spurious results by chance alone. We therefore apply the BH correction across all 90 tests. For the aggregate results (All-model, Open-weight, Frontier groups), we use model-stratified permutation tests that give each model equal weight and apply BH correction across the group-level tests.

Results. Direct full-code access has a strong but representation- and challenge-dependent effect (Table 2). For all models equally weighted, IDA decompilation improves understanding by 2.09 points on RPC ($q = 1.99 \times 10^{-7}$) and 0.84 on Secrets ($q = 0.00371$). Ghidra (+1.93, $q = 2.92 \times 10^{-7}$), r2dec (+0.76, $q = 0.0416$), and Angr (+1.93, $q = 2.09 \times 10^{-8}$) representations also lead to improvements on RPC, but none shows a corrected all-model benefit on Secrets. IDA disassembly does not show a corrected aggregate improvement on either challenge (+0.51, $q = 0.147$ on RPC; -0.51 , $q = 0.0778$ on Secrets).

The effect is concentrated among open-weight agents. IDA decompilation improves the aggregate open-weight tier score by 2.73 points on RPC ($q = 2.09 \times 10^{-8}$) and 1.37 on Secrets ($q = 1.10 \times 10^{-4}$). Ghidra (+2.60, $q = 3.93 \times 10^{-9}$), r2dec (+0.80, $q = 0.0287$), and Angr (+2.17, $q = 2.66 \times 10^{-7}$) also lead open-weight models to improve on RPC. Instead, no Frontier aggregate effect remains significant after the family-wide correction.

These results suggest that the benefit is not solely due to removing MCP: agents could, in principle, retrieve the same functions through tools, but doing so would require additional search, tool selection, and integration of fragmented outputs, whereas a contiguous decompilation reduces this orchestration burden.

Table 2. Equal-weight model-stratified RQ2 ablation effects: mean understanding-score difference (ablation minus baseline MCP) and BH-adjusted q per representation and model group. Bold entries remain significant at $q < 0.05$.

Representation	All models		Open-weight		Frontier	
	RPC	Secrets	RPC	Secrets	RPC	Secrets
Source code via MCP	+0.80 ($q = 0.0120$)	+0.58 ($q = 0.0497$)	+0.67 ($q = 0.0208$)	+0.77 ($q = 0.0538$)	+1.07 ($q = 0.148$)	+0.20 ($q = 0.610$)
IDA decompilation	+2.09 ($q = 2.0 \times 10^{-7}$)	+0.84 ($q = 0.0037$)	+2.73 ($q = 2.1 \times 10^{-8}$)	+1.37 ($q = 0.00011$)	+0.80 ($q = 0.425$)	-0.20 ($q = 0.757$)
Ghidra decompilation	+1.93 ($q = 2.9 \times 10^{-7}$)	+0.20 ($q = 0.439$)	+2.60 ($q = 3.9 \times 10^{-9}$)	+0.57 ($q = 0.0963$)	+0.60 ($q = 0.518$)	-0.53 ($q = 0.0833$)
r2dec decompilation	+0.76 ($q = 0.0416$)	-0.20 ($q = 0.518$)	+0.80 ($q = 0.0287$)	-0.03 ($q = 1.000$)	+0.67 ($q = 0.425$)	-0.53 ($q = 0.359$)
Angr decompilation	+1.93 ($q = 2.1 \times 10^{-8}$)	+0.40 ($q = 0.141$)	+2.17 ($q = 2.7 \times 10^{-7}$)	+0.50 ($q = 0.170$)	+1.47 ($q = 0.0649$)	+0.20 ($q = 0.695$)
IDA disassembly	+0.51 ($q = 0.147$)	-0.51 ($q = 0.0778$)	+0.43 ($q = 0.350$)	-0.33 ($q = 0.392$)	+0.67 ($q = 0.425$)	-0.87 ($q = 0.106$)

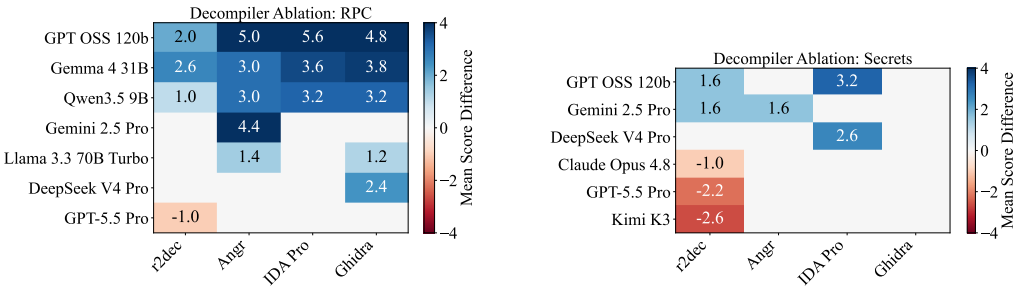


Fig. 1. Descriptive per-model understanding-score shifts for full decompiler inputs without MCP relative to baseline MCP runs (RPC left, Secrets right).

Figure 1 visualizes the per-model decompiler shifts. Fifteen of the 90 individual static no-MCP contrasts remain significant after BH correction, all with $|\delta| = 1$ (i.e., every ablation run outperformed every baseline run for that model) and adjusted $q = 0.0476$. The strongest positive model-level shifts are concentrated in weaker open-weight systems on RPC. Notably, some agents perform *worse* in the no-MCP condition: after correction, Kimi K3 and GPT-5.5 Pro regress under r2dec on Secrets, and Kimi K3 regresses under IDA disassembly on Secrets, suggesting that these strong models benefit from the interactive MCP workflow and/or different code representations on that challenge.

Human-reference analysis. We do not treat $p > 0.05$ from a difference test as evidence that an ablated configuration is comparable to humans. Instead, a configuration qualifies as reaching the human reference only if it meets both the same one-point non-inferiority (NI) criterion as RQ1 against the rate-selected top-human cohort (q_{NI}) and a corrected improvement over its own baseline (q_{shift}), both after BH correction. Only two configurations meet both conditions, both on RPC and using Gemma 4: with full IDA decompilation improves from 3.2 to 6.8 ($q_{shift} = 0.0476$) and is non-inferior to the top-human reference ($q_{NI} = 0.00582$); with full Ghidra decompilation it improves from 3.2 to 7.0 ($q_{shift} = 0.0476$) and is likewise non-inferior ($q_{NI} = 0.000202$). No Secrets configuration both significantly improves over baseline and establishes non-inferiority after correction.

RQ2: What Limits Agent Reverse Engineering Performance?

Semantically richer source responses yield a modest aggregate benefit, whereas direct full-decompiler inputs yield substantially larger gains, especially for open-weight agents. These gains are not reproduced by disassembly-only inputs and are not uniform across challenges, models, or representations. The results point to a combination of representation quality, search burden, and tool-mediated workflow demands rather than to any single factor. This suggests that future evaluations of SRE agents should consider model capability and interface design separately, since a weak workflow can hide capabilities that become visible under a better representation.

6 RQ3: How Do Agents' and Humans' Strategies Differ?

Several high-performing agents reach the human reference level, but similar outcomes do not necessarily imply similar analytical processes. RQ3, therefore, asks how successful agent workflows differ from recorded human reverse-engineering behavior on the same binaries.

In this context, previous work [39] has evaluated offensive-security agent trajectories against human-crafted step-level references. Our focus is instead a matched human-agent comparison in which both groups analyze the same binaries, and their behavior can be mapped onto common reverse-engineering abstractions.

6.1 Macro-Strategies Comparison

Methods. At the macro level, we map both human and agent traces into five categories. Agent traces consist of MCP tool calls logged during each run; human traces consist of manual IDA Pro interactions recorded during the prior study [4]. Despite the difference in interaction medium, both expose the same broad analytical activities:

- *Recon*: entry-point and binary-overview discovery;
- *Data*: inspection of variables, memory, and structures;
- *Relationship*: cross-reference and call-graph traversal;
- *Read*: direct reading of decompiled or disassembled code;
- *Edit*: renaming and annotation.

Our categories refine the three-phase model of Votipka et al. [45]: Recon corresponds to overview; Data and Relationship decompose sub-component scanning; Read captures focused code inspection; and Edit makes state externalization explicit.

For the descriptive macro comparison, we use high-performing human and agent traces to highlight the clearest contrast in strategies. Each session is represented as a sequence over the five categories, from which we compute a first-order Markov transition matrix that shows, for each category, the probability of transitioning to the next. We report $\Delta = P_{\text{agent}} - P_{\text{human}}$, where positive entries indicate transitions that are more common in agent traces, and negative entries indicate transitions more common in human traces. This comparison is *descriptive only*: because we deliberately selected the best-performing traces, any patterns we observe might partly reflect the selection rather than a general behavioral law. We therefore use it only to generate hypotheses about differences in navigation; the formal statistical claims below are based on session-level comparisons that do not condition on the outcome.

Results. Figure 2 shows that high-performing agent traces are more read-centric, whereas human traces allocate more activity to iterative editing and repeated structural exploration.

The strongest agent-skewed transition is *Recon*→*Read* ($\Delta=0.81$), accompanied by increases in *Data*→*Read* ($\Delta=0.66$), *Data*→*Relationship* ($\Delta=0.17$), and *Read* self-loops ($\Delta=0.16$). Human-skewed behavior includes *Data* self-loops ($\Delta= -0.81$), *Edit*→*Read* ($\Delta= -0.69$), *Edit* self-loops ($\Delta= -0.21$), and *Recon* self-loops ($\Delta= -0.58$). The *Edit* row deserves separate mention: its Δ entries sum to -1

because the high-performing agent traces contain no *Edit* transitions at all. This is not a limitation of the toolset: the MCP interface exposes renaming, retyping, and commenting tools, yet they were never used. Humans externalize what they learn into the database, whereas agents may instead rely on their conversation context to carry analysis state. This descriptive contrast motivates a more explicit analysis of navigation.

6.2 Micro-Strategies Comparison

Methods. We construct a call graph for each binary and map both agent and human movements between functions. Function depth is the shortest call-graph distance from `start`. We classify each movement as:

- *Down*: from a function to one of its callees;
- *Up*: from a function to one of its callers;
- *Lateral*: between functions at the same depth without a direct call-graph edge; or
- *Jump*: between functions at different depths without a direct call-graph edge.

These categories refine the navigation behaviors identified by Mantovani et al. [25] by explicitly separating hierarchical traversal from non-structural navigation. We further stratify movements by *visit layer* k : within a session, a movement out of function f belongs to layer k if it is the k -th time the session leaves f . Layer $k=1$ thus captures how each function is first left, and deeper layers capture how the analyst behaves when returning to functions already visited.

Results. The layer-based descriptive analysis already suggests a strong navigation asymmetry. At the first visit ($k=1$), agents allocate 81.6% of movements to jumps, compared with 19.0% for humans; at deeper layers, agent movement remains jump-heavy. This jump-heavy pattern suggests a strategy we term *greedy fetching*: instead of following call-graph structure step by step, strong agents typically jump to non-adjacent functions, fetch new code, and continue reasoning over newly retrieved information, revisiting relatively little. Greedy fetching describes *how* information is acquired, not simply how much of the binary is eventually covered. On compact binaries, this can be effective because much of the relevant program can be gathered quickly; as programs grow, however, aggressive retrieval may increase tool overhead and consume the available context before the agent can consolidate what it has seen. We treat this as a hypothesis at this stage: the next subsection, RQ4, examines and measures its scalability.

6.3 Measuring Greedy Fetching

Methods. We formalize *greedy fetching* as a four-dimensional session-level behavioral signature based on the following metrics:

$$J = \frac{N_{\text{jump}}}{N_{\text{function transitions}}}, \quad C = \frac{N_{\text{unique viewed functions}}}{N_{\text{callgraph functions}}}, \quad (1)$$

$$Q = \frac{N_{\text{code read events}}}{N_{\text{analysis events}}}, \quad R = \frac{N_{\text{repeated function views}}}{N_{\text{function views}}}. \quad (2)$$

Here, J is the non-call-graph jump rate, C is the function coverage, Q is the read-event share, and R is the revisit rate. J , C , and R are computed over the same recovered call graph for each binary, so they are defined identically for humans and agents; Q instead relies on mapping events to the paper's Read category (MCP operations for agents, manual IDA events for humans).

To see how and where greedy fetching manifests, we measure these four metrics on the strong-performing cohorts established in RQ1: the top-performing human participants and all five baseline MCP runs from the three non-inferior models (GPT-5.5 Pro, Claude Opus 4.8, Kimi K3). Because

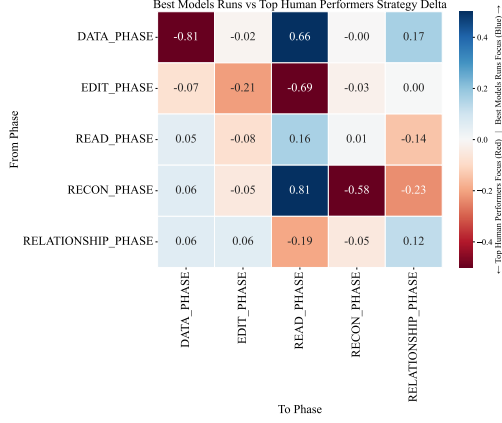


Fig. 2. Delta matrix comparing high-performing agent traces with top human traces. Positive values are agent-skewed, and negative values are human-skewed.

Table 3. Greedy-fetching metrics: strong-agent cohort (GPT-5.5 Pro, Claude Opus 4.8, Kimi K3) vs. top-performing humans. Columns report session-level median [IQR]; Δ is the equal-weight agent-mean minus human-mean (percentage points, 95% hierarchical-bootstrap CI); q is BH-adjusted.

Challenge	Metric	Strong agents: median [IQR]	Humans: median [IQR]	Δ pp [95% CI]	q
RPC	Jump rate J	82.1 [69.3, 84.9]	6.7 [4.5, 11.4]	+66.5 [+52.5, +78.4]	0.0070
RPC	Function coverage C	96.3 [87.0, 96.3]	92.6 [85.2, 95.4]	+2.2 [-6.0, +9.8]	0.7952
RPC	Read-event share Q	87.9 [83.1, 88.9]	72.9 [67.7, 76.2]	+18.5 [+9.0, +29.4]	0.0727
RPC	Revisit rate R	8.0 [4.2, 10.3]	85.6 [84.1, 87.1]	-76.3 [-80.1, -72.2]	0.0070
Secrets	Jump rate J	73.9 [58.7, 87.5]	31.4 [27.1, 42.8]	+37.9 [+18.0, +55.4]	0.0070
Secrets	Function coverage C	88.2 [82.4, 94.1]	94.1 [88.2, 98.5]	-0.4 [-12.2, +13.9]	0.9895
Secrets	Read-event share Q	76.5 [71.8, 79.2]	68.9 [37.0, 76.1]	+17.7 [+2.5, +34.4]	0.3730
Secrets	Revisit rate R	5.9 [0.0, 24.6]	76.8 [73.6, 81.1]	-63.1 [-76.1, -45.7]	0.0070

the 15 agent runs (three models $\times$ five runs for each challenge) share underlying models, they are not truly independent. To avoid artificially inflating statistical significance, we average the five runs within each model to yield three independent model-level values, which we compare against the ten human participants. For each metric $M \in \{J, C, Q, R\}$ and each challenge, we estimate the equal-weight agent mean minus the human mean, $\hat{\Delta}_M$.

We quantify the precision of this estimate with a 95% CI from a *hierarchical bootstrap*: a two-stage resampling in which some agent models are included and some of their five runs contribute, reflecting both model and run variability, with human participants resampled independently.

We assess statistical significance with an exact permutation test: pooling all 13 independent clusters (three agent-model means and ten human participants), we calculate how often all $\binom{13}{3} = 286$ possible assignments to the “agent” group produce a difference at least as large as observed, then apply BH correction across the eight planned comparisons. Table 3 additionally reports session-level medians and inter-quartile ranges.

Results. The strongest human–agent differences are in *navigation* and *revisitation*, not in how much of these compact binaries is eventually inspected.

On RPC, the strong-agent cohort has a median jump rate of 82.1% versus 6.7% for humans, an equal-weight difference of +66.5 percentage points. The 95% CI for this difference is [+52.5, +78.4]

(well above zero), confirming agents jump far more often than humans ($q = 0.0070$). Instead, the agent revisit rate is 8.0% versus 85.6% for humans, a -76.3 point difference (95% CI $[-80.1, -72.2]$; $q = 0.0070$), where the entirely negative interval indicates that agents revisit far less.

The pattern repeats on Secrets: jump rate is 73.9% versus 31.4% (+37.9 points, 95% CI $[+18.0, +55.4]$, $q = 0.0070$), while revisit rate is 5.9% versus 76.8% (-63.1 points, 95% CI $[-76.1, -45.7]$, $q = 0.0070$).

In contrast, function coverage does *not* differ significantly on either task (RPC: $q = 0.795$; Secrets: $q = 0.990$), and read-event share differences are not significant after correction (RPC: $q = 0.073$; Secrets: $q = 0.373$), even though it is higher for agents in the raw numbers. We obtain the same overall pattern when we include all unassisted human sessions, not just the top ten performers, suggesting the difference reflects a broader contrast between human and agent analysis behavior rather than an artifact of the top-performing cohort selection.

Greedy fetching depends on the analysis environment. We next test whether agents use the same greedy-fetching strategy when the code representation changes. The source-code replacement experiment (Section 5.2) keeps the models and MCP interaction loop unchanged while returning semantically richer original source for code-reading calls. For the same three strong models, all greedy-fetching metrics shift when the code quality improves. Function coverage falls from 89.0% to 65.0% on RPC and from 73.1% to 33.3% on Secrets; read share falls from 81.1% to 70.2% and from 70.5% to 58.3%, respectively. Jump rate also falls from 77.6% to 65.9% on RPC and from 79.3% to 73.9% on Secrets. In other words, when each tool call returns higher-quality code, the agents need to fetch less of it.

Crucially, this reduced fetching does not hurt understanding: the equal-weight score difference between source-code and baseline runs is $+0.267$ on RPC (90% CI $[0.006, 0.528]$) and $+0.067$ on Secrets (90% CI $[-0.318, 0.451]$), both falling entirely within the prespecified $[-1, +1]$ practical-equivalence region: the models understand the binary just as well despite fetching less code. To test whether these behavioral shifts are statistically significant, we apply the same model-stratified permutation tests used in the human-agent comparison above. After BH correction, function coverage and read share both show significant reductions on both challenges, confirming that agents genuinely fetch less code when given better representations. The jump-rate reduction is significant on RPC but not on Secrets, suggesting that the navigation change is challenge-dependent.

RQ3: How Do Agents' and Humans' Strategies Differ?

Strong agents do not reverse engineer like humans. They achieve similarly broad coverage by jumping aggressively across the binary and rarely revisiting previously inspected code, a strategy we call *greedy fetching*. This behavior is not fixed: richer code representations reduce the need for broad retrieval without reducing understanding. For future SRE agents, the implication is clear: progress may come not from fetching more code, but from better navigation, selective retrieval, and persistent analysis state that helps agents retain and refine what they have already learned.

7 RQ4: How Does Program Size Affect The Feasibility Of Full-Context Analysis?

Greedy fetching depends on a critical assumption: that enough of the binary can be retained within the agent's available context. RQ4 asks how often complete decompiled or disassembled representations fit within current model context windows when moving beyond the two compact benchmark binaries. We evaluate this scalability question directly across a large sample of real-world software.

Methods. We construct four groups of real-world binaries (Windows/Linux and system/application). To avoid choosing an arbitrary application workload, we ground application selection in public popularity rankings. On Windows, we randomly select and install 50 applications from the Microsoft

Store's *Most popular apps for PC* listing available at the time of collection [27]. On Linux, we follow the same approach on a clean Ubuntu 24.04 system, randomly selecting and installing 50 popular application packages using the Debian Popularity Contest (PopCon) as the popularity source [11]. On Windows, we distinguish system and application binaries based on standard directory locations. On Linux, we isolate application binaries by hashing directories before and after installing the selected software, removing any binaries already present in the baseline. Following state-of-the-art practice [10], we use Detect-It-Easy [13] to filter framework and packer code (e.g., .NET, PyInstaller, Electron) that hides the real app's code. We remove duplicates by considering their SHA-256 values, randomly select 1,000 binaries per group, and extract complete decompiled and disassembled representations using IDA Pro. We then tokenize each representation using the appropriate tokenizer for each evaluated model and compute context-window saturation as the ratio of the token count to the model's context window size. A saturation of 100% or higher indicates that the representation fills the entire context window.

Results. We measure, for each model, the proportion of binaries that saturate the context window (excluding samples for which IDA failed to analyze); the full per-model, per-OS, per-group breakdown is in our repository (Section 12).

Three patterns are consistent: first, decompiled representations are substantially smaller than disassembled representations. Statistical analysis indicates that disassembly is significantly larger than decompiled code ($p = 1 \times 10^{-5}$, effect size $es = 0.87$, classified as a large effect), approximately twice as large, confirming that decompiled representations use the available context more efficiently and are therefore more compatible with greedy fetching than assembly-level views.

Second, Linux binaries exhibit lower saturation rates than Windows binaries: decompiled code size is significantly larger on Windows ($p = 1.1 \times 10^{-163}$, $es = 0.55$, large effect size), which explains the higher saturation rates and makes a randomly selected Windows binary substantially more likely to exceed the practical operating range of greedy fetching.

Third, although application binaries tend to have higher saturation rates than system binaries, the practical difference is small despite being statistically significant (mean difference of 7.96 percentage points, $p = 2.57 \times 10^{-15}$, $|es| = 0.28$, small effect size).

Taken together, these patterns show that greedy fetching through complete binary retrieval is only conditionally feasible in practice. Even for models with context windows of at least 1M tokens (e.g., GPT-5.5 Pro, Kimi K3), the share of binaries that saturate the window on decompiled code ranges from 1.9% for Linux system binaries to 39.1% for Windows applications, so complete decompiled retrieval occurs in 60.9-98.1% of cases depending on the binary group. For Single-GPU models, the share of saturating binaries reaches 66.6% on Windows applications and falls to 20.5% on Linux applications.

The practical implication is that context saturation is a scalability boundary rather than the primary explanation for failures on the compact benchmark tasks. An agent that retrieves large amounts of decompiled code can fit the entire binary into context in many real-world cases, but in the remaining cases, more selective retrieval and state management are required than greedy fetching alone provides.

RQ4: How Does Program Size Affect The Feasibility Of Full-Context Analysis?

Full-context analysis works surprisingly often, but it does not scale indefinitely. Large context windows can hold the complete decompilation of many real-world binaries, yet larger applications, especially on Windows, still exceed this limit, and disassembly further exacerbates it. This marks a practical boundary for greedy fetching: future SRE agents cannot rely on simply loading more code. They will need to decide *what* to retrieve, *what* to keep, and *what* to revisit, combining selective retrieval with persistent analysis state as programs grow.

8 Takeaways

Taken together, our results do not show that autonomous SRE has been solved. They do, however, fit a broader trend reported in prior work: autonomous agents are making inroads into a domain that traditionally depended on skilled human reverse engineers [7, 23, 29, 35].

1. Human-competitive SRE is possible, but not yet general. RQ1 shows that, on these two compact static-analysis SRE tasks, human-level performance is strongly model-dependent. GPT-5.5 Pro, Claude Opus 4.8, and Kimi K3 satisfy the non-inferiority criterion on both challenges, but the large differences among agents make it clear this capability should not yet be generalized to smaller, workstation-scale LLMs. While Kimi K3 shows open-weight models can achieve this, its size still requires accelerator infrastructure; bringing this capability to ordinary local hardware remains an important future direction for analyzing sensitive software.

2. Agent capability depends on tool and analysis fidelity. RQ2 shows that model quality alone does not explain performance: weaker agents either never establish a productive tool-use loop or retrieve information but fail to turn it into understanding, so autonomous SRE depends on both reasoning capability and the ability to operate the analysis workflow. Our ablations clarify this: providing the complete decompiled program directly yields substantially larger improvements than returning the original source code via MCP, particularly for open-weight models. Removing MCP alone is not sufficient; representation quality, search burden, tool orchestration, and reasoning ability all contribute. Future agents must therefore treat the interface as an integral part of the system.

3. Strong agents reach similar coverage through a very different strategy. Humans and strong agents eventually inspect similarly broad portions of the binaries, but they reach that coverage in very different ways: agents exhibit substantially more non-call-graph jumping and substantially less revisitation, a behavior we call *greedy fetching*, rather than repeatedly traversing structure and returning to inspected code; agents aggressively move toward new functions and reason over what is accumulated in context. While prior work has found that sequential exploration is characteristic of human novices [25], agents bypass structured exploration entirely: when sufficient code fits within the context, broad exposure can substitute for the structured memory-management strategies humans need.

4. Greedy fetching is an adaptation, not a fixed property of agents. The source-code intervention reveals greedy-fetching changes when the information environment changes: when strong models receive semantically richer source responses via MCP, function coverage and read activity decrease significantly, while understanding remains the same. Agents do not fetch aggressively out of an immutable preference for reading code; rather, greedy fetching is useful when information is fragmented, and retrieval is cheap, so richer information means agents need less of it. Interfaces that expose better abstractions may reduce unnecessary exploration.

5. Context capacity changes the agent's SRE strategy. Across real-world binaries, complete decompiled representations often fit within large context windows, but saturation becomes substantially more common for large Windows apps. This bounds the greedy fetching strategy: when

programs exceed the context budget, agents must decide what information to enter into context, discard, or keep for later reasoning. Larger context windows can move this boundary, but do not remove it. Next-generation SRE systems will therefore need mechanisms that current agents use weakly: *selective retrieval*, *persistent analysis state*, and *evidence-driven revisitation*.

9 Threats to Validity

Task scope and external validity. We measure SRE performance using two compact, static-analysis CTF-style challenges. This provides a controlled, reproducible setting, but does not cover all of real-world SRE. Our results do not imply that agents perform robustly on all security tasks. However, this restricted setting enables our matched human-agent comparison using the same understanding rubric. While broader benchmarks exist [23, 39, 41], they lack the matched human writeups and fine-grained IDA traces required for our behavioral analysis. RQ4 partially addresses this by evaluating whether complete representations of real-world binaries fit in model contexts, though it measures scalability rather than reasoning. Furthermore, the evaluation interface itself is static-only, with no debugger or dynamic/runtime execution path (Section 3.3), because it intentionally matches the static IDA-based setting used in the human reference study. This does not imply that dynamic analysis is unimportant for real-world SRE; extending this comparison to dynamic tooling is left to future work.

Training-data contamination. Because the challenge binaries and their source code were released publicly before some models' training cutoffs, we cannot rule out the possibility that models were exposed to the underlying implementation during training. The structured 7-point scoring rubrics, ground-truth solution writeups, and fine-grained human interaction logs, by contrast, were never made public, which reduces the risk that models memorized the specific evaluation criteria or reference solutions. This does not, however, eliminate the possibility of prior exposure to the source code itself, and we treat training-data contamination as an unresolved threat to validity rather than a risk we can rule out.

Human reference and non-inferiority margin. RQ1 defines the strongest human reference using the understanding-rate measure [4], which determines *who enters the reference cohort*, while the RQ1 outcome itself is the raw understanding score. Our $\Delta = 1$ margin represents a single understanding concept, as even strong humans sometimes miss a rubric item, so non-inferiority indicates practical comparability under this explicit tolerance, not equivalence to all professional reverse engineers. Relatedly, the RQ2 ablations change the overall analysis environment rather than isolating single causal factors: providing the original source restores names, types, and structure (an upper-bound representation experiment rather than a pure decompiler test), while the no-MCP experiment removes tool-selection costs, search effort, and interaction failures while changing context organization. We therefore interpret these ablations as environmental changes rather than clean causal decompositions.

Comparing human and agent behavior. RQ3 compares manual IDA interactions with agent MCP interactions; the raw events are not directly equivalent, so we address this by mapping behaviors to shared abstractions (J, C, R) based on the recovered call graph. Read share (Q) remains the least directly comparable due to differences in granularity, so we do not rely on it alone to claim strategy differences. Our behavioral reference is the ten strongest unassisted participants per challenge [4], who should not be conflated with the general professional population; to ensure statistical validity, we perform inference at the model-cluster level to avoid pseudo-replication, use hierarchical bootstrap intervals, and apply BH correction across the eight planned comparisons. On interpretation: strong agents jump between structurally unrelated functions more often than humans and revisit functions less often, without achieving significantly broader function coverage, so greedy fetching is an alternative method of acquiring information; RQ3 shows this behavior

diminishes with richer program information, indicating it is an environmental adaptation rather than an intrinsic agentic property.

Model reproducibility. Evaluating closed-frontier models affects long-term reproducibility as providers update their systems; we mitigate this by reporting experimental configurations and evaluating open-weight models, but exact reproduction of closed-model behavior may become impossible.

10 Conclusion

Autonomous SRE is no longer only a question of whether an agent is capable enough. Our results show that, under this controlled two-task setting, some agents can already reach human-competitive understanding on compact, static-analysis SRE tasks, and do so much faster, but their success depends strongly on the surrounding analysis environment.

We identify *greedy fetching* as a distinct agent strategy: strong agents jump aggressively across a binary, retrieve new information quickly, and revisit far less than human analysts. This behavior changes with richer representations and eventually runs into context limits as programs grow, so future progress will require better interfaces, selective retrieval, persistent analysis state, and more deliberate revisitation, not simply larger context windows.

If humans and machines can reach the same understanding through fundamentally different paths, then the question is no longer only whether machines can reverse engineer, but what *understanding* means when the path to it is no longer uniquely human.

11 Ethical Considerations

Our study constitutes human subject research (HSR), so we follow our institution’s HSR guidelines and received approval from the Institutional Review Board (IRB). In accordance with our IRB’s requirements and our own ethical considerations regarding the proper handling of our participants, all participants were fully informed of the study’s purpose, data handling procedures, and their right to withdraw at any time without penalty. They provided explicit consent for recording and data storage, and all recordings were stored on access-controlled servers in accordance with the best security practices.

12 Data Availability

To support transparency and reproducibility, we release an anonymized repository¹ containing all code, scripts, and agent results required to reproduce our experiments and analyses. The repository includes the evaluation pipeline, data processing and statistical analysis code, and instructions for running the full workflow. It also provides the prompts, configurations, and experimental settings used for all agents. The repository also contains additional details about the statistical methodology, along with the prompts used, the tool list, and additional figures and tables².

References

- [1] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. 2019. Guidelines for Human-AI Interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 1–13. doi:10.1145/3290605.3300233
- [2] Simone Aonzo, Yufei Han, Alessandro Mantovani, and Davide Balzarotti. 2023. Humans vs. Machines in Malware Classification. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 1145–1162. <https://www.usenix.org/conference/usenixsecurity23/presentation/aonzo>

¹https://anonymous.4open.science/r/humans_vs_machines/

²https://anonymous.4open.science/r/humans_vs_machines/supplementary/README.md

- [3] Andrea Arcuri and Lionel Briand. 2014. A Hitchhiker’s Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering. *Software Testing, Verification and Reliability* 24, 3 (2014), 219–250. doi:10.1002/stvr.1486
- [4] Zion Leonahenahe Basque, Samuele Doria, Ananta Soneji, Wil Gibbs, Adam Doupé, Yan Shoshitaishvili, Eleonora Losiouk, Ruoyu Wang, and Simone Aonzo. 2026. Decompiling the Synergy: An Empirical Study of Human–LLM Teaming in Software Reverse Engineering. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, USA.
- [5] Yoav Benjamini and Yoel Hochberg. 1995. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society: Series B (Methodological)* 57, 1 (1995), 289–300. doi:10.1111/j.2517-6161.1995.tb02031.x
- [6] Guoqiang Chen, Huiqi Sun, Daguang Liu, Zhiqi Wang, Qiang Wang, Bin Yin, Lu Liu, and Lingyun Ying. 2025. ReCopilot: Reverse Engineering Copilot in Binary Analysis. arXiv:2505.16366 [cs.CR] <https://arxiv.org/abs/2505.16366>
- [7] Xiang Chen, Anshunkang Zhou, Chengfeng Ye, and Charles Zhang. 2025. ClearAgent: Agentic Binary Analysis for Effective Vulnerability Detection. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Language Models and Programming Languages (LMPL)*. Association for Computing Machinery, 130–137. doi:10.1145/3759425.3763397
- [8] Byunggeon Choi, Hongjoo Jin, Dong Hoon Lee, and Wonsuk Choi. 2026. Toward Efficient Deobfuscation via Large Language Models. *IEEE Access* 14 (2026), 30844–30859. doi:10.1109/ACCESS.2026.3660477
- [9] Jacob Cohen. 1988. *Statistical Power Analysis for the Behavioral Sciences* (2 ed.). Routledge.
- [10] Savino Dambra, Yufei Han, Simone Aonzo, Platon Kotzias, Antonino Vitale, Juan Caballero, Davide Balzarotti, and Leyla Bilge. 2023. Decoding the Secrets of Machine Learning in Malware Classification: A Deep Dive into Datasets, Feature Extraction, and Model Performance. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 60–74. doi:10.1145/3576915.3616589
- [11] Debian Project. [n. d.]. Debian Popularity Contest. <https://popcon.debian.org/>. Accessed August 21, 2026.
- [12] Gelei Deng, Yi Liu, Victor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. 2024. PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 847–864. <https://www.usenix.org/conference/usenixsecurity24/presentation/deng>
- [13] Detect-It-Easy 2025. <https://github.com/horsicq/detect-it-easy>.
- [14] Yangruibo Ding, Yanjun Fu, Omriyyah Ibrahim, Chawin Sitawarin, Xinyun Chen, Basel Alomair, David Wagner, Baishakhi Ray, and Yizheng Chen. 2025. Vulnerability Detection with Code Language Models: How Far Are We?. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering*. IEEE, 1729–1741. doi:10.1109/ICSE55347.2025.00038
- [15] Chongzhou Fang, Ning Miao, Shaurya Srivastav, Jialin Liu, Ruoyu Zhang, Ruijie Fang, Asmita, Ryan Tsang, Najmeh Nazari, Han Wang, and Houman Homayoun. 2024. Large Language Models for Code Analysis: Do LLMs Really Do Their Job?. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 829–846. <https://www.usenix.org/conference/usenixsecurity24/presentation/fang>
- [16] David C. Funder and Daniel J. Ozer. 2019. Evaluating Effect Size in Psychological Research: Sense and Nonsense. *Advances in Methods and Practices in Psychological Science* 2, 2 (2019), 156–168. doi:10.1177/2515245919847202
- [17] Peiwei Hu, Ruigang Liang, and Kai Chen. 2024. DeGPT: Optimizing Decompiler Output with LLM. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. doi:10.14722/ndss.2024.24401
- [18] Xinyu Hu, Zhiwei Fu, Shaocong Xie, Steven H. H. Ding, and Philippe Charland. 2025. SoK: Potentials and Challenges of Large Language Models for Reverse Engineering. arXiv:2509.21821 <https://arxiv.org/abs/2509.21821>
- [19] IDA Pro MCP 2025. <https://github.com/mrexodia/ida-pro-mcp>.
- [20] Zimo Ji, Daoyuan Wu, Wenyuan Jiang, Pingchuan Ma, Zongjie Li, and Shuai Wang. 2025. Measuring and Augmenting Large Language Models for Solving Capture-the-Flag Challenges. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 603–617. doi:10.1145/3719027.3744855
- [21] Linxi Jiang, Xin Jin, and Zhiqiang Lin. 2025. Beyond Classification: Inferring Function Names in Stripped Binaries via Domain-Adapted LLMs. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. doi:10.14722/ndss.2025.240797
- [22] Ziyang Li, Saikat Dutta, and Mayur Naik. 2025. IRIS: LLM-Assisted Static Analysis for Detecting Security Vulnerabilities. In *The Thirteenth International Conference on Learning Representations*. <https://openreview.net/forum?id=9LdJDU7E91>
- [23] Yuzhe Liu, Zhijie Liu, Zhengmin Yu, Shu Wang, Ling Jiang, Sen Nie, Shi Wu, Zhanyong Tang, and Yuan Zhang. 2026. Towards Generality: Task-Adaptive Binary Analysis via Semantic Retrieval and Verifiable Reasoning. In *35th USENIX Security Symposium (USENIX Security 26)*. USENIX Association, Baltimore, MD. <https://www.usenix.org/conference/usenixsecurity26/presentation/liu-yuzhe>
- [24] Ziyang Luo, Zhiqi Shen, Wenzhuo Yang, Zirui Zhao, Prathyusha Jwalapuram, Amrita Saha, Doyen Sahoo, Silvio Savarese, Caiming Xiong, and Junnan Li. 2025. MCP-Universe: Benchmarking Large Language Models with Real-World Model Context Protocol Servers. arXiv:2508.14704 [cs.AI] <https://arxiv.org/abs/2508.14704>

- [25] Alessandro Mantovani, Simone Aonzo, Yanick Fratanonio, and Davide Balzarotti. 2022. RE-Mind: a First Look Inside the Mind of a Reverse Engineer. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 2727–2745. <https://www.usenix.org/conference/usenixsecurity22/presentation/mantovani>
- [26] Dylan Manuel, Nafis Tanveer Islam, Joseph Khoury, Ana Nunez, Elias Bou-Harb, and Peyman Najafirad. 2024. Enhancing Reverse Engineering: Investigating and Benchmarking Large Language Models for Vulnerability Analysis in Decompiled Binaries. arXiv:2411.04981 [cs.CR] <https://arxiv.org/abs/2411.04981>
- [27] Microsoft. [n. d.]. Most popular Apps on PC. <https://www.microsoft.com/en-us/store/most-popular/apps/pc>. Accessed August 21, 2026.
- [28] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to Help With Code Understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (Lisbon, Portugal) (ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 97, 13 pages. doi:10.1145/3597503.3639187
- [29] Ryutaro Nishizaka, Yudai Fujiwara, Takuya Shimizu, Kazushi Kato, and Yuichi Sugiyama. 2026. Towards LLM-Resistant Software Protection: Agent Failure Patterns in CTF Reverse Engineering. In *Workshop on Binary Analysis Research (BAR)*. San Diego, CA, USA. doi:10.14722/bar.2026.23058
- [30] Timothy Nosco, Jared Ziegler, Zechariah Clark, Davy Marrero, Todd Finkler, Andrew Barbarello, and W Michael Petullo. 2020. The industrial age of hacking. In *29th USENIX Security Symposium (USENIX Security 20)*. 1129–1146.
- [31] Hammond Pearce, Benjamin Tan, Prashanth Krishnamurthy, Farshad Khorrami, Ramesh Karri, and Brendan Dolan-Gavitt. 2022. Pop Quiz! Can a Large Language Model Help With Reverse Engineering? arXiv:2202.01142 [cs.SE] <https://arxiv.org/abs/2202.01142>
- [32] Tomás Pelayo-Benedet, Kevin Borgolte, and Ricardo J. Rodríguez. 2026. Does Representation Matter? Evaluating IRs for LLM-Based Binary Decompilation. In *Proceedings of the 9th Workshop on Binary Analysis Research (BAR)*. Internet Society, San Diego, CA, USA. doi:10.14722/bar.2026.23077
- [33] John C. Platt. 2000. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. In *Advances in Large-Margin Classifiers*, Alexander J. Smola, Peter L. Bartlett, Bernhard Schölkopf, and Dale Schuurmans (Eds.). MIT Press, 61–74.
- [34] Saman Pordanesh and Benjamin Tan. 2024. Exploring the Efficacy of Large Language Models (GPT-4) in Binary Reverse Engineering. arXiv:2406.06637 [cs.SE] <https://arxiv.org/abs/2406.06637>
- [35] Salem Radey, Jack West, and Kassem Fawaz. 2026. Challenges and Future Directions in Agentic Reverse Engineering Systems. arXiv:2604.14317 [cs.CR] <https://arxiv.org/abs/2604.14317>
- [36] Satrio Adi Rukmono, Lina Ochoa, and Michel R. V. Chaudron. 2023. Achieving High-Level Software Component Summarization via Hierarchical Chain-of-Thought Prompting and Static Code Analysis. In *2023 IEEE International Conference on Data and Software Engineering (ICoDSE)*. IEEE, 7–12. doi:10.1109/ICoDSE59534.2023.10292037
- [37] Xiuwei Shang, Shaoyin Cheng, Guoqiang Chen, Yanming Zhang, Li Hu, Xiao Yu, Gangyang Li, Weiming Zhang, and Nenghai Yu. 2024. How Far Have We Gone in Binary Code Understanding Using Large Language Models. In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 1–12. doi:10.1109/ICSME58944.2024.00012
- [38] Xiuwei Shang, Zhenkan Fu, Shaoyin Cheng, Guoqiang Chen, Gangyang Li, Li Hu, Weiming Zhang, and Nenghai Yu. 2026. An Empirical Study on the Effectiveness of Large Language Models for Binary Code Understanding. *Empirical Software Engineering* 31, 1, Article 14 (2026). doi:10.1007/s10664-025-10748-5
- [39] Minghao Shao, Nanda Rani, Kimberly Milner, Haoran Xi, Meet Udeshi, Saksham Aggarwal, Venkata Sai Charan Putrevu, Sandeep K. Shukla, Prashanth Krishnamurthy, Farshad Khorrami, Ramesh Karri, and Muhammad Shafique. 2026. Towards Effective Offensive Security LLM Agents: Hyperparameter Tuning, LLM as a Judge, and a Lightweight CTF Benchmark. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 40. 29660–29668. doi:10.1609/aaai.v40i35.40210
- [40] Ze Sheng, Zhicheng Chen, Shuning Gu, Heqing Huang, Guofei Gu, and Jeff Huang. 2025. LLMs in Software Security: A Survey of Vulnerability Detection Techniques and Insights. *Comput. Surveys* 58, 5, Article 134 (2025), 35 pages. doi:10.1145/3769082
- [41] Tingxuan Tang, Nicolas Janis, Kalyn Asher Montague, Kevin Eykholt, Dhilung Kirat, Youngja Park, Jiyong Jang, Adwait Nadkarni, and Yue Xiao. 2026. From Assistance to Autonomy: An Empirical Study of AI Use in a Live Capture-the-Flag (CTF) Competition. In *35th USENIX Security Symposium (USENIX Security 26)*. USENIX Association, Baltimore, MD. <https://www.usenix.org/conference/usenixsecurity26/presentation/tang-tingxuan>
- [42] Anton Tkachenko, Dmitriy Suskevich, and Benjamin Adolphi. 2025. Deconstructing Obfuscation: A Four-Dimensional Framework for Evaluating Large Language Models Assembly Code Deobfuscation Capabilities. arXiv:2505.19887 <https://arxiv.org/abs/2505.19887>
- [43] Together AI. [n. d.]. Serverless models. <https://docs.together.ai/docs/serverless/models>. Accessed August 21, 2026.
- [44] Saad Ullah, Mingji Han, Saurabh Pujar, Hammond Pearce, Ayse Coskun, and Gianluca Stringhini. 2024. LLMs Cannot Reliably Identify and Reason About Security Vulnerabilities (Yet?): A Comprehensive Evaluation, Framework, and

- Benchmarks. In *2024 IEEE Symposium on Security and Privacy (SP)*. 862–880. doi:10.1109/SP54263.2024.00210
- [45] Daniel Votipka, Seth Rabin, Kristopher Micinski, Jeffrey S. Foster, and Michelle L. Mazurek. 2020. An Observational Investigation of Reverse Engineers’ Processes. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 1875–1892. <https://www.usenix.org/conference/usenixsecurity20/presentation/votipka-observational>
 - [46] Danning Xie, Zhuo Zhang, Nan Jiang, Xiangzhe Xu, Lin Tan, and Xiangyu Zhang. 2024. ReSym: Harnessing LLMs to Recover Variable and Data Structure Symbols from Stripped Binaries. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 4554–4568. doi:10.1145/3658644.3670340
 - [47] Xiangzhe Xu, Zhuo Zhang, Zian Su, Ziyang Huang, Shiwei Feng, Yapeng Ye, Nan Jiang, Danning Xie, Siyuan Cheng, Lin Tan, and Xiangyu Zhang. 2025. Unleashing the Power of Generative Model in Recovering Variable Names from Stripped Binary. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. doi:10.14722/ndss.2025.240276
 - [48] Muzhi Yu, Zhengran Zeng, Wei Ye, Jinan Sun, Xiaolong Bai, and Shikun Zhang. 2025. R3-Bench: Reproducible Real-World Reverse Engineering Dataset for Symbol Recovery. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 217–227. doi:10.1109/ASE63991.2025.00026
 - [49] J.D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can’t Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) (*CHI ’23*). Association for Computing Machinery, New York, NY, USA, Article 437, 21 pages. doi:10.1145/3544548.3581388
 - [50] Rui Zhang, Nathan J. McNeese, Guo Freeman, and Geoff Musick. 2021. "An Ideal Human": Expectations of AI Teammates in Human-AI Teaming. *Proc. ACM Hum.-Comput. Interact.* 4, CSCW3, Article 246 (Jan. 2021), 25 pages. doi:10.1145/3432945
 - [51] Xinran Zheng, Xingzhi Qian, Yiling He, Shuo Yang, and Lorenzo Cavallaro. 2025. Beyond Classification: Evaluating LLMs for Fine-Grained Automatic Malware Behavior Auditing. arXiv:2509.14335 [cs.CR] <https://arxiv.org/abs/2509.14335>