Decompiling the Synergy: An Empirical Study of Human–LLM Teaming in Software Reverse Engineering

Zion Leonahenahe Basque*, Samuele Doria†, Ananta Soneji*, Wil Gibbs*, Adam Doupé*,
Yan Shoshitaishvili*, Eleonora Losiouk†, Ruoyu Wang*, Simone Aonzo‡
*Arizona State University
†University of Padua
‡EURECOM

***Abstract*—Large Language Models (LLMs) are revolutionizing fields previously dominated by human effort. This work presents the first systematic investigation of how LLMs can team with analysts during software reverse engineering (SRE). To accomplish this, we first document the state of LLMs in SRE with an online survey of 153 practitioners, and then we design a fine-grained human study on two Capture-The-Flag-style binaries representative of real-world software.**

In our human study, we instrumented the SRE workflow of 48 participants (split between 24 novices and 24 experts), observing over 109 hours of SRE. Through 18 findings, we found various benefits and harms of LLMs in SRE. Remarkably, we found that LLM assistance narrows the expertise gap: novices' comprehension rate rises by approximately 98%, matching that of experts, whereas experts gain little; however, they also had harmful hallucinations, unhelpful suggestions, and ineffective results. Known-algorithm functions are triaged up to 2.4× faster, and artifact recovery (symbols, comments, types) increases by at least 66%. Overall, our findings identify powerful synergies of humans and LLMs in SRE, but also emphasize the significant shortcomings of LLMs in their current integration.

I. Introduction

The first step to securing software is understanding it; after all, complex tasks such as finding unintended program behavior first require comprehending its intended behavior. In the modern age, programs are developed at breakneck speeds, requiring an understanding of more software than ever. To understand this software, often foreign to its analyst, is known as Software Reverse Engineering (SRE).

Unfortunately, SRE is a complex and primarily human-driven process [1]–[3]. SRE practitioners use a variety of strategies to understand software, frequently carrying out multiple subtasks in the process [1]. These subtasks are typically performed iteratively, and the relationships among them are often complex [1], [2].

Network and Distributed System Security (NDSS) Symposium 2026
23 - 27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.240380
www.ndss-symposium.org

Recently, LLMs have shown promise for approaching largely human-driven tasks in security [4], [5]. As such, rapid work by both the research and practitioner communities explores integrating LLMs into SRE tasks. Researchers have approached isolated tasks, such as symbol recovery [6]–[9], type inference [6], [8], and vulnerability identification [10], [11]. In response, practitioners integrate these initial findings into binary decompilers, e.g., Hex-Rays Decompiler (ships with IDA Pro) and Ghidra, among other industry-standard SRE tools [12]–[15]. However, despite their known advancements in individual tasks, no work exists to show whether these LLMs benefit the SRE process from the human perspective.

Similar to human-AI teaming [16], the interaction between an SRE practitioner and an LLM impacts the performance of LLM-enhanced, human-driven SRE. We refer to this interaction and LLM results interpretation as the *dynamics* between SRE practitioners and LLMs. Existing studies, which focus on studying LLM improvements to SRE tasks in isolation [5]–[9], fail to consider the human-LLM dynamics involving iterative sub-tasks. These studies do not directly or adequately measure the impacts that LLMs have on the entire human-driven SRE process. This represents a lost opportunity to optimize LLM-enhanced, human-driven SRE.

In this paper, we present the first work studying the dynamics between humans and LLMs during SRE. To accomplish this systematic study, we present three distinct phases. First, to understand the current LLM-SRE community, we survey 153 SRE practitioners who have used LLMs in SRE. We discover that practitioners rely on six distinct features to accomplish SRE tasks. Using these results, with the help of 13 experts in SRE, we design a fine-grained human study involving two CTF-style challenges, a decompiler plugin to support the six discovered features, and an online platform for instrumentation. Finally, we conducted this study on 48 participants, relying on both quantitative and qualitative data found in observations, writeups, and post-study responses.

In our study, we observe over 6,586.0 minutes of SRE across 24 experts and 24 novices. Participants interacted with LLMs a total of 1,517 distinct times, across two challenges, on over 50 functions. Through an analysis of their behavior, we discover

18 different findings on the impacts and best strategies for LLM use in SRE. These findings enable us to answer *where*, *when*, and *how* LLMs can augment the SRE process. We find that they have both benefits and harms to SRE.

We find LLMs shorten the skill gap for novice SRE practitioners, improving their performance by 98%, helping them approach expert levels of SRE. On average, LLMs decrease the analysis time of standard algorithms by 238% and enable users to recover more artifacts lost to compilation (at least 66% more). However, these benefits were not without their harms. Experts showed nearly no change in performance when utilizing an LLM, and in some cases, were harmed by their hallucinations, namely on vulnerability identification. Artifacts recovered by LLMs, more frequently than not, were unhelpful, causing more noise in the SRE process. Additionally, novel and large code continues to degrade LLMs' quality, hurting reverse engineers who overrely on them.

We observe, as it stands, that LLMs play an assistant role in SRE. They are best utilized when they act as a quick and under-relied-upon filter for understanding, not a replacement. LLMs show promise in being the first line of analysis, but expertise is still required to interpret the results.

Contributions. This paper makes the following contributions:

- We create a snapshot of the state of LLM-SRE between 2024 and 2025, and establish six ways SRE practitioners rely upon and use LLMs.
- We illuminate 18 findings surrounding the integration of LLMs into the SRE process, including their effects, their best uses, and how they might be improved upon.
- We implement and open source a platform[1] and a decompiler plugin[2] for studying fine-grained human-driven SRE with LLMs in decompilers, including both behavior instrumentation and practitioner-utilized LLM features.

Our work takes the first steps toward understanding and optimizing how humans and LLMs collaborate for SRE. While the results are promising, they are constrained by some limitations: the scale and representativeness of our study challenges, the restricted tooling (limited to static analyses and excluding dynamic or advanced non-LLM tools), the absence of participant assessments of SRE environment satisfaction, and the reliance on self-reported expertise. These factors may limit generalizability (Section X).

However, even if our work is not definitive, it establishes a foundation for future research on human-AI collaboration in SRE. Our findings will help the community explore impactful ways to enhance human-LLM collaboration in SRE. It also offers insights for education, future research, and practical tool development for real-world reverse engineering.

II. Background and Scope of the Study

This paper explores the intersection of two domains: Software Reverse Engineering and Large Language Models.

[1]https://github.com/mahaloz/dec-synergy-study

[2]https://github.com/mahaloz/DAILA

Software Reverse Engineering (SRE) involves analyzing software to uncover its design and functionality. Applications range from malware analysis to vulnerability discovery and piracy prevention. The primary goal is to reconstruct program logic and identify conditions that trigger specific code behaviors, often linked to bugs or malicious actions. SRE typically unfolds in multiple phases and involves tools like IDA Pro and Ghidra, which integrate disassembly, decompilation, and debugging in a unified interface. Both static and dynamic analyses are employed: the former inspects code without execution (e.g., file structure, functions, assembly), while the latter monitors runtime interactions with memory and the OS.

This study focuses on *static binary analysis*, examining executables without execution. Our participants interact with disassembled and decompiled code via IDA Pro, aligning with traditional program comprehension research [1], [17] that emphasizes reading over debugging. We extend traditional research in this area by allowing access to decompilation, which is commonly used in SRE, as it simplifies binaries for understanding [18], [19].

Large Language Models (LLMs) are transformer-based neural networks trained on vast text corpora using self- and semi-supervised methods [20], [21]. Modern LLMs, typically using decoder-only architectures [22], excel at general-purpose language generation. Initially, task adaptation required fine-tuning. Today, *prompt engineering* guides model behavior through carefully designed input prompts, leveraging pre-trained knowledge without retraining [23]. This has proven efficient and often comparable to fine-tuning for large models.

Scope of the study. We aim to investigate how LLMs are being integrated into the SRE workflow, specifically for static code understanding, and assess whether their use enhances analysts' performance compared to traditional methods. Through this study, we intend to answer the following research questions:

RQ1: How do SRE practitioners integrate LLMs into the SRE process, and what are their perceptions? (Section IV)

RQ2: How does the inclusion of LLMs in the SRE process impact the performance of practitioners? (Section VI)

RQ3: How do practitioners interact with LLMs, and what factors influence their interactions? (Section VII)

III. Methodology

In this section, we describe the three phases of our methodology, discuss the recruitment process, and present the statistical methods employed throughout our study.

A. Three Study Phases

To answer our research questions, we systematically explore, design, and complete our study in three phases.

I) Formative research (Section IV). We conduct both a comprehensive literature review (including practitioner tools) and an online pre-study survey to understand the design requirements for our study. This phase informs our design on how practitioners interact with LLMs during the SRE process

and through what platforms. In total, we use 153 survey responses to create a snapshot of modern LLM for SRE use between 2024 and 2025.

II) Study design (Section V). Next, we design the human study and platform for measuring LLM use in SRE. We use the Phase I results for both the SRE platform (IDA Pro) and LLM features used in our study. We then design two binary executable CTF-style challenges for practitioners to complete, with a writeup required for explaining each solution. We design the challenges and platform iteratively with a team of 13 SRE experts to be representative of real-world difficulties that practitioners may face. Finally, we design a post-study survey to qualitatively confirm the quantitative results we find in the experiment.

III) Empirical Study and Analysis (Section VI and Section VII). Using the design from Phase II, we run our study on 48 participants (a subset of the 153 survey respondents in Phase I) and analyze the results. We analyze three sources of data in this phase: 1. fine-grained quantitative data, such as clicks and function renames actioned during the study, 2. solution writeups used to assess participants' understanding of the challenges, and 3. free responses in the post-study survey capturing general sentiments of LLM usefulness. We analyze the results with two goals. First, we analyze how LLMs impacted the SRE process, from the understanding level to the solving speed. Second, we explore how participants extracted useful responses from LLMs with different strategies.

B. Participant Recruitment

We recruited all participants online during 2024 and 2025. We primarily recruited participants through personal and professional connections to ensure that they have prior experience using LLMs for SRE. Recruits include students with relevant experience (e.g., taking malware analysis courses), researchers from eight universities, and experts from six renowned cybersecurity companies with headquarters located on different continents.

We do not offer compensation for the anonymous pre-study survey, which complies with prior work practices [1], [24]. Participants who expressed interest in Phase III provided their email addresses for contact. Contacted participants who completed the SRE session were compensated with a $50 gift card. This includes participants who *finished* both challenges, but may not have solved both correctly.

C. Statistical Analysis of Quantitative Data

We implement a rigorous methodology that integrates statistical testing and effect size estimation in our study. Because the field of human reverse engineering study is emerging, there are no established effect size thresholds specific to SRE tasks. Thus, we situate our methodology within the broader domain of software engineering, which also involves code comprehension, a key component of our participants' activities. We adopt the guidelines proposed by Kampenes et al. [25] and conduct all statistical tests using a significance threshold of $\alpha = 0.05$. In cases involving multiple comparisons, we apply the Benjamini-Hochberg correction [26] to control the false discovery rate (FDR).

Comparing two independent samples. For any given challenge, a participant is assigned to either the control or treatment group independently of their assignment on the other challenge. With each participant completing two challenges, we randomly assign their treatment group (LLM support) to one of the two challenges. This design, together with our decision to make two vastly different challenges, ensures that prior exposure to one challenge does not influence the performance of the other, which addresses the potential carryover effect.

Statistical methodology. I) We assess the normality of the data using the Shapiro-Wilk test. II) If the data is consistent with normality, we assess the equality of variances using Levene's test, with the mean as the center parameter for symmetric distributions [27]. III) If Levene's test fails to reject the null hypothesis, we assume equal variances and use the two-sample (pooled) t-test. Otherwise, if it indicates significant variance differences, we use Welch's t-test. IV) In case of statistically significant difference, we computed Cohen's *d* to quantify the magnitude; for samples with $n < 20$, we applied Hedges' g adjustment, which adds a small bias correction [28]–[30]. V) In order to estimate the effect size, we adopted the finer-grained classification of Funder et al. [31], who further refined Cohen's guidelines [32] that are frequently employed in social sciences and software engineering [33]. Accordingly, we adopted the following interpretation: effect size es is classified as small ($es \leq 0.1$), medium ($0.1 < es \leq 0.2$), large ($0.2 < es \leq 0.8$), and very large ($es > 0.8$).

However, if the assumption of normality is violated (point I) or the sample size is too small ($n < 11$), we follow the recommendations of Fay et al. [34] and Gibbons et al. [35]. Namely, we employ the Mann-Whitney U test as a robust nonparametric alternative to the independent samples t-test. Given that the literature suggests caution with very small samples [36], [37], we never test with fewer than five observations per group. We also compute Cliff's Delta δ as a robust nonparametric effect-size measure [38], [39]. It ranges from $-1 \leq \delta \leq 1$, and measures how often values in one distribution tend to be larger than values in another distribution. We adopt commonly used Cliff's δ thresholds in software engineering [33]: negligible ($\delta < 0.147$), small ($0.147 \leq \delta < 0.33$), medium ($0.33 \leq \delta < 0.474$), and large ($\delta \geq 0.474$).

Correlation. To assess the presence of a statistically significant relationship between two variables (i.e., equal size), we implement a flexible correlation testing procedure that selects the appropriate statistical test based on the distributional properties of the input data, enabling us to capture both linear and non-linear associations without violating statistical assumptions. If both datasets are found to be normally distributed (using the Shapiro-Wilk test), we compute the Pearson correlation coefficient, which reflects the strength and direction of a linear relationship. Conversely, if one or both groups deviate from normality, we use Spearman's rank-order correlation, which is more appropriate for ordinal data or

non-linear monotonic relationships. The effect size is again interpreted in accordance with Funder et al. [31].

IV. Formative Research

This section investigates **RQ1:** *How do SRE practitioners integrate LLMs into the SRE process, and what are their perceptions?*

We first investigate how practitioners use and integrate LLMs into the SRE process to guide our experiment design in Section V. Then, we design an online survey to provide us with a road map for accurately studying how practitioners use LLMs. All questions are motivated by a literature review of recent works in LLMs for SRE. We reviewed both academic work and popular plugins used by practitioners. We present aggregated answers to questions that are relevant to our study design. The survey and its answers can be found in Table IV in the Appendix.

In total, 153 participants responded to our online survey. The survey respondents were students (43.1%), followed by employees (27.5%), academic researchers (17.0%), and freelancers (9.2%). Many participants were seasoned reverse engineers: 41.2% had more than three years of experience, while 40.5% had between one and three years. Their most used SRE framework was IDA Pro (81.0%), with Ghidra (73.0%) closely following.

LLM Use. Participants also reported how often they used LLMS during SRE: 34.0% responded sometimes, whereas 32.0% responded often or always. Additionally, 67.8% of participants reported that LLMs were occasionally beneficial, 25.2% considered them highly advantageous, while 7.0% found them unhelpful. The most used LLM was GPT (OpenAI, 85.6%), followed by Claude (Anthropic, 11.6%). To interact with their LLMs, participants reported mainly using decompilation as input (59%), followed by machine code (28%), and intermediate languages (13%).

LLM Features. Participants reported using LLMs for six identified features known from prior work:

(1) *Function Summarization*. Comments the function with a description or summary based on its behavior or purpose [12], [14].
(2) *Function Identification*. Identifies well-known functions and algorithms. While testing this feature, we also noticed that if the LLM does not find a match, it tries to categorize the functions based on its potential role (e.g., cryptographic, network, file handling) w.r.t. similar well-known code patterns [40].
(3–4) *Function and Variable Renaming*. Renames the function or all of its variables with meaningful nomenclature to facilitate comprehension [6], [8], [9], [12]–[14], [40], [41].
(5) *Vulnerability Identification*. It identifies potential security vulnerabilities by detecting patterns and coding practices that may lead to security risks [10], [40].
(6) *Library Function Documentation*. It generates context-rich documentation of a library function so that the reverse engineer can understand its functionality without manually searching through reference materials [41].

The following percent of participants reported using these features in SRE: Function Summarization (63.7%), Function Identification (28.8%), Function Renaming (23.3%), Variable Renaming (28.1%), Vulnerability Identification (17.1%), and Library Function Documentation (2.1%). We note that participants wrote in the Library Function Documentation feature through an "Other" option in our survey. We use the results of these responses to design what LLM features will be available to participants in the study (Section V).

Perceptions on LLMs. In a free-response section, participants also shared their experiences using LLMs for SRE tasks. The reported experiences are mixed, acknowledging both the benefits and limitations of existing LLM-SRE tools. Participants report that LLMs excel at explaining the behaviors of decompiled code and improving code readability by renaming variables. They are also useful for understanding known algorithms and accelerating workflows by creating scripts for SRE frameworks. Regarding limitations, participants report that LLMs often produce incorrect or misleading responses, reducing trust and wasting time. LLMs often create generic or superficial explanations, and their effectiveness diminishes for large, obfuscated, or highly mathematical tasks.

V. Study Design

While our online survey in Section IV provides a broad perspective of LLM use in SRE, we next seek to understand how LLMs augment the process at a more technical level. Thus, we design a behavior-focused experiment that captures granular details on SRE performances, with and without LLM assistance on modern tools. Such a contrast allows us to measure not only raw performance differences but also how the presence of LLM assistance changes analyst behavior, tool usage patterns, and solution strategies.

We designed and piloted our experiment in collaboration with a Subject-Matter Experts (SME) team comprising 13 expert reverse engineers: three paper authors, six research group members, two industry professionals, and two academics with expertise in binary analysis and Capture The Flag (CTF) organization. Importantly, none of the SME team members participated in the study, ensuring unbiased data collection.

A. Study Overview

Motivated by recent work in human SRE [1], we design two CTF-style reverse engineering challenges and task participants with solving them. We design a fully browser-based study, where participants can access a virtual machine (VM), a virtual networked computer (VNC), that hosts the challenges and LLM tools, whose features were informed directly by the findings of our online survey (Section IV).

We provide each participant with a link to our online platform (Section V-B). Upon visiting the site, the participant is given access to two challenges (Section V-D), ordered randomly. Upon clicking start on a challenge, a VM is launched, giving access to that binary, an instrumented version of IDA

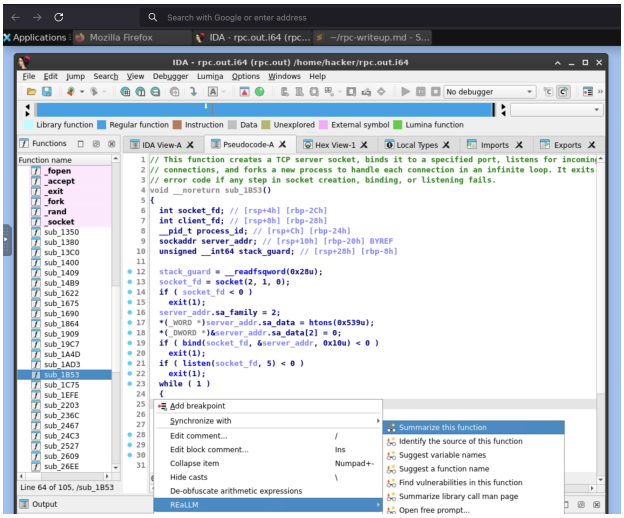

Figure 1: A screenshot of the online platform after a participant starts a challenge. A VNC is accessible in the browser that provides access to an instrumented decompiler (with an LLM plugin), browser, and text editor.

Pro (including the 6 LLM features derived from the pre-study, § IV), a text editor, and a web browser. On one challenge (random), the LLM features are disabled.

A participant reverses the challenge with the goal of *understanding* how to "read the flag." When a participant feels satisfied with their understanding, they end the study and complete the writeup. Upon finishing both challenges, they fill out a post-study survey. Accordingly, each participant in the study creates two writeups describing their solutions. We grade each writeup for understanding (§ V-F) and collect the technical data, such as clicks, generated while they reversed.

B. Online Platform

A key aspect of our study design is making it accessible, since expert reverse engineers are difficult to find and schedule [42]. To accomplish this, we design our online platform to host our challenges and tools that can allow participants to complete the study asynchronously.

Our online platform builds atop the cybersecurity education work DOJO [43], which helps provide participants with a virtual desktop, a virtual networked computer (VNC), that hosts both the study challenges and our tools. A screenshot of what participants would see in this environment can be seen in Figure 1.

Upon visiting the site for the first time, a participant is shown our two challenges in a random order. Before starting the experiment, each participant could optionally view a walkthrough video introducing the IDA Pro interface and a basic toy program to familiarize themselves with the platform. We clearly state in each challenge's description that the participants must not find and report a flag, but *must provide a natural language description of the flag retrieval procedure*. This is the same approach that Mantovani et al. adopt [1] to avoid unintended solutions in their SRE challenges.

When participants click "Start" on a challenge, the recording for that VNC is started as well as all other instrumentation, and a VNC tab is opened. That VNC has both the challenge open in our tools and a text editor for creating a writeup. When a participant feels they have completed the challenge, they click "Finish" and the final time is saved along with their instrumented data and writeup.

C. SRE & LLM Tooling

When a VNC is opened, a participant is given access to an instrumented version of IDA Pro 9 with the respective challenge open in it. Our instrumentation collects most trackable data in IDA, including functions visited, clicks, variable renames, etc. IDA is the only tool participants have access to, meaning they cannot use a debugger or run the challenge.

To give participants access to LLM features, we develop a decompiler plugin that implements the six features discovered in our online survey (§ IV). We implement each feature based on prior work [6], [8]–[10], [12]–[14], [40], [41]. In particular, we closely studied the methodology for creating prompts from Hu et al. [6], which created prompts to improve decompiled functions. We created our prompts by first describing the decompilation improvement task (summarization, renaming, etc.). Next, we described the output format of the task (JSON). We then provided a minimal example input with the correct output format. Finally, we provided the decompilation text of the function in question (triggered by participant use on a function). We validated this design choice by studying other related open-source prompts [12]–[15], [40] and found that they had similar patterns.

Additionally, to account for any missed features that participants may use in real-world reversing, we integrate a chatbox into the decompiler where participants can interact freely with an LLM. Excluding the LLM chatbox, participants use the LLM features by right-clicking on an open function in IDA Pro and using the specified feature.

For all features, including the chatbox, participants could use any of the models reported in our online survey (§ IV): GPT-4o, GPT-4o-Mini, GPT-4-Turbo, Claude-3-5-Sonnet, and Gemini-Pro. When participants start the study, they are required to choose a model. If they are unsure which model to select, GPT-4o is used by default.

D. SRE Challenges Development

Here, we present our design heuristics for developing two challenges, RPC (Challenge #1) and Secrets (Challenge #2).

Representativeness. Challenges must represent real-world software. In collaboration with the SME team, we identify the most common software characteristics encountered during SRE and select a subset for manual static analysis. We exclude the ones that demand dynamic or automated analysis, e.g., self-modifying code. The list includes authentication/authorization, compression, encryption, hashing, data validation, encoding, file management, memory manipulation, networking, parsing, serialization, string operations, and sorting. We also include some known algorithms in each challenge because real-world

Table I: Software metrics of the SRE challenges.

Metric	RPC	Secrets
Functions	26	26
Lines of Code	427	461
McCabe Cyclomatic Complexity	29	31
Operands Count	957	1021
Distinct Operands	172	184
Operators Count	1933	2397
Distinct Operators	50	51
Halstead Volume	22526	26922
Halstead Difficulty	139	141
Halstead Effort	3.13×10^6	3.81×10^6

software uses them. We note that this list excludes traditional program obfuscation, such as control flow flattening, which was outside the scope of our study. Our challenges also only contained advanced mathematics in hashing and encryption.

In our post-survey, we ask participants to rate their agreement with the statement "the two challenges are good representative examples of real-world software." Out of 48 respondents, 81% ($n = 39$) strongly or moderately agree whereas no one strongly disagreed.

With the help of the SME team and some novice volunteers, we estimate that both challenges could be solved within one hour for experts and two hours for novices. Our participants could complete both challenges at different times to reduce cognitive burden.

Equal levels of difficulty. The two challenges must act as control and treatment. So, they must be of an equivalent level of difficulty, which means solving them should require similar efforts. However, perceived difficulty is extremely subjective. In addition to the supervision of the SME team, we elect to rely on the well-known software metrics, striving to ensure as much similarity as possible, as reported in Table I. Particularly, the Halstead Difficulty measures the difficulty in understanding the program, for example, when doing a code review.

In our post-survey, we ask participants to rate their agreement with the statement: "The two CTF challenges were of similar difficulty." Out of 48 responses, 87% ($n = 43$) selected strongly or moderately agree, whereas only one strongly disagreed.

E. SRE Challenges

Challenge#1 – RPC. It mimics a remote procedure call (RPC) server. The server uses a custom protocol to communicate with the client, which involves encoding (Base64), compression (Run-Length Encoding - RLE), and encryption (Tiny Encryption Algorithm - TEA). In this protocol, after authentication, the client can manipulate a 32-byte buffer by sending specific bytes that correspond to different operations. The flag is stored in the `/flag` file, but only the admin user can load files into the buffer. Therefore, to solve this challenge, the player has to find the (hardcoded) admin user name and then spot a vulnerability: the decryption key of the admin password is easily guessable as it is randomized from the `time` library function. Finally, after authenticating as an admin, they must send the correct bytes with a sequence of operations that load the flag file into the buffer, which is eventually returned.

Challenge#2 – Secrets. It is an encrypted file manager with a custom shell (and commands) for managing files. The file manager supports multiple user accounts for creating encrypted files. The encryption algorithm is Advanced Encryption Standard (AES) in ECB mode. On user registration, a username and password are stored. The system will create a folder with the same user name and derive the user's key using a custom key derivation function. Then, the key is hashed (DJB2), and its hash is stored in clear text in a file named `/<username>/.shadow`.The saved hash is from the key derived from the password. Since each file is encrypted, they are world-readable, implying that all other users can read everyone's hashes. In this system, the `admin` user has created an encrypted file named `flag` that contains the string to be retrieved. This challenge is solved by understanding that the function for showing the content of files is vulnerable to a path traversal attack. After creating a user account, the command `show ../admin/.shadow` will display the `admin`'s key hash. The custom key derivation function is then susceptible to brute-force attacks, as its output is contingent upon a mere two bytes and can be validated with the known DJB2 hash. In the event of a match, such a key decrypts the flag file.

Completion Criteria. Participants completed a challenge if they created a writeup and did not use outside tools other than IDA Pro and our LLM plugin. All writeups should contain text about solving the challenge. Writeups with only irrelevant text are not considered and would be an incomplete SRE session.

We note that this approach allows for participants who had incomplete or incorrect solutions to the challenges. We included all who made valid attempts on both and demonstrated some understanding of at least one. This variability makes direct comparisons difficult, as some participants spent less time and showed lower comprehension. However, it allows us to make a more holistic measurement of different SRE approaches and understanding levels. To account for this variability in solution completeness and participant understanding, we evaluated each challenge writeup using a structured scoring rubric, discussed next.

F. Challenge Writeup Evaluation

We scored each writeup from 0 to 8. Each challenge included 7 checkpoints, with a 8 point awarded for a fully correct solution. This accommodates solvers who demonstrate holistic understanding without explicitly identifying all technical details (e.g., recognizing an algorithm's function without naming it).

Challenge#1 - RPC - Checkpoints. First, writeups were awarded one point for identifying the numerical operation to read the flag. To read the flag, they must use the hardcoded name and password "admin" and "secret," which is worth one point. The second bug in RPC is that randomness is seeded to time, which awards another point for mentioning it in their writeup. All data must be decrypted to talk to the server, and mentioning "encrypted" or "hashed" data also awards one

point. Finally, a point is awarded for each of the three known algorithms (Base64, RLE, TEA) mentioned in the writeup.

Challenge#2 - Secrets - Checkpoints. Writeups are first awarded a point for mentioning that the program manages and creates files. An additional point is awarded if they mention that the files can be encrypted. A key part of the challenge is understanding that each password has a derived hash stored in a ".shadow" file, which awards another point. The hash is created from a key derived from the password, which is flawed. A point is awarded for mentioning that the derived key is flawed. To read an arbitrary user's hash, a user must use the path traversal in the "show" command. A point is awarded for mentioning "path traversal." Finally, a point is awarded for each of the two known algorithms (AES, DJB2) mentioned in the writeup. Note that Quicksort is not awarded a point since the algorithm is implemented in a function (ls) that is not required to solve the challenge.

Since the awarding of individual points could still be subjective, two researchers on the SME team scored the writeups independently. Across both challenges, a total of 96 writeups were scored. Both researchers scored a writeup the same, on the same checkpoints, in 82 cases (85%). They differed by assigning one point on 12 cases (13%) and two points on 2 cases (2%). In total, both researchers aligned on 753 out of 768 point assignments (98%). With this in mind, we concluded that our grading criteria and point assignment were likely sufficient.

VI. LLM Impacts on SRE

We ran our study, designed in Section V, between 2024 and 2025. In total, 51 respondents from our online survey participated in our experiment. After reviewing their results, we eliminated 3 participants' data from our analysis due to submission quality: one left an empty writeup, another wrote only irrelevant text in a writeup, and one used dynamic analysis to solve their challenge. As such, all analysis is performed on the 48 valid submissions in our study. Cumulatively, these 48 participants reverse engineered for 109 hours across two challenges, creating 96 writeups, and interacting with LLMs 1517 times. This group included 24 self-reported experts and 24 self-reported novices. We report the summary of their performance in Table II.

Therefore, the purpose of this section is to analyze these data to answer **RQ2:** *How does the inclusion of LLMs in the SRE process impact the performance of practitioners?*

We first focus on how LLMs affected the SRE process positively or negatively. Second, we analyze the strategies these participants took to get optimal performance from their LLMs. When comparing any two groups for independence or correlation, we use the methodology described in Section III-C. When we obtain a statistically significant result, we report it with triple: p-value (p), effect size (es), and effect size interpretation. Otherwise, we just report the p-value.

Finally, when available, we integrate responses from our post-study survey to give a more holistic view of our results.

Table II: Averaged performance metrics across all study participants and challenges, grouped by expertise.

Metric	Experts		Novices	
	RPC	Secrets	RPC	Secrets
Solve Times (m)	64.55	69.32	70.70	69.89
Time in Function (m)	2.39	2.56	2.62	2.61
Comments	2.00	1.96	1.38	0.83
LLM Interactions	19.96	10.42	19.17	13.67
Function Transitions	324.25	219.75	356.42	212.08
Clicks	512.04	361.17	636.12	417.21
Understanding Score	5.38	4.21	3.79	2.46

Each response is anonymized to their skill level with *N* or *E* for novice and expert, respectively.

A. Study Assumptions

Before further analysis, we validate two assumptions made in our study design.

Assumption 1: *Self-reported SRE expertise is trustworthy.* To evaluate this assumption, we first examined whether participants' self-reported years of SRE experience differed between the two groups. We found a statistically significant difference ($p = 0.0001$, $es = 0.65$, large effect size). On average, participants in the expert group reported six years of experience, compared to two years in the novice group. Although this comparison relies on self-reported measures as well, it provides evidence of internal consistency within participants' own assessments of their background.

We next evaluated whether the self-reported novice and expert groups differed on performance-based measures independent of self-report. Because we are interested in participants' underlying SRE skill, we compared their performance on the challenge completed without LLM assistance.

Experts statistically differed from novices in their understanding scores from their writeups ($p = 0.001$, $es = 0.56$, large effect size). On average, experts had an understanding score of 4.79, while novices had 3.12, both out of the maximum of 8. This indicates that the self-reported expert group often understood more than the novices across both challenges.

An *understanding rate* can be calculated by dividing a participant's understanding score by their solve time (minutes). Experts statistically differed from novices in their understanding rates ($p = 0.001$, $es = 0.57$, large effect size). Experts had an average rate of 0.08 points per minute, while novices had 0.06. This indicates that self-reported experts understood more in less time than novices, reproducing results found in previous work [1]. Although this data does not indicate if a self-reported expert is an expert relative to all reversers, it does suggest that they may be relative to our self-reported novices. Considering these observations, we assume self-reports of expertise levels are reflective of reality for our study.

Assumption 2: *Both challenges are of equal difficulty.* We validate this assumption by, again, comparing two groups when they are *not* utilizing an LLM. For this case, we compare participants across both expertise levels on the two challenges. There was no significant difference in understanding scores ($p = 0.68$) or rates ($p = 0.47$) on both challenges across

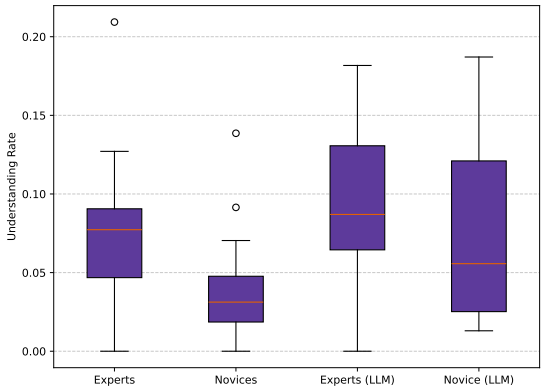

Figure 2: Understanding rates for participants with and without LLM assistance, grouped by SRE expertise.

all users, irrespective of expertise. Additionally, considering expertise, experts and novices had no significant difference in understanding scores or rates between the challenges ($p > 0.05$). This data indicates that challenges had similar rates of understanding and, therefore, difficulty.

B. Expertise Dependent LLM Improvements

SRE expertise plays a fundamental role in how fast and well reverse engineers understand a program [1]. SRE expertise also determined whether a participant greatly benefited from an LLM. In Figure 2, an overview of understanding rates can be seen across skill level and LLM use.

Finding 1. *Novices using LLMs exhibit expert levels of program understanding rates—regardless of whether experts use LLMs or not.* Comparing novice participants' performances across both challenges, we observed a statistical difference when they utilized an LLM ($p = 0.03$, $es = 0.38$, medium effect size). While utilizing an LLM, they had an average understanding rate of 0.07, compared to a 0.04 rate without (98.55% improvement). Additionally, when comparing novices with LLMs to experts (both with LLMs and without), we found no statistical difference in their understanding rates.

Various *novices* in our study also felt that they had improved holistically across challenges. As N15 noted, "I would not have understood the binary half as well without the LLM in that same time span." N13 remarked, "I was kinda mind blown by how the LLM analyzed some [complex] functions... and determined they perform base64 encoding and TEA decryption almost instantly. That really saved time during reversing by not having to focus on those parts and focus on more interesting parts to reverse engineer."

N14 summarized: "Using LLM made things much faster and easier with the risk of mistakes. However, these mistakes can be easily reversed. The time complexity spent on reversing might still be O(Time spent without LLM), but with LLM, the average time will be highly decreased." Meaning that in the worst case, fixing LLM mistakes could take as much time as doing the task manually from scratch (hence, same time complexity). However, this claim did not necessarily hold for all participants.

Finding 2. *LLM usage does not impact experts' program understanding rate.* Unlike novices, when comparing experts' performances across both challenges, we did not see a statistically significant difference in their understanding rates with LLMs. However, experts had marginal gains in their average understanding rate with LLM usage. This data may indicate that SRE expertise outweighs the effects of LLMs. An expert (E5), articulated why this may be the case, stating that LLMs were "a hit or miss on usefulness, but low effort to read the summary and see if it seemed to be useful. Really high value when it got it right."

C. Augmented Artifact Recovery

During the SRE process, participants recovered different artifacts lost in compilation. Specifically, symbol names (both functions and variables), types (including custom structs), and comments. Prior work had identified that these artifacts (function names and comments) were recovered more frequently by experts [1]. We explore these findings by expanding the number of artifacts users can create and receive help on from an LLM. Except for structs, LLMs could also recover these artifacts on demand, as a practitioner would normally.

Finding 3. *Manual artifact recovery is positively correlated with program understanding.* We find that experts and novices did not significantly differ in the number of artifacts they manually recovered ($p = 0.33$). On average, experts recovered 54 artifacts, while novices recovered 41. We speculate that this difference in results from prior work may be caused by the expanded set of artifacts we support in our study, such as structs and stack variables.

However, we found that participants who manually recovered more artifacts, regardless of expertise, tended to have higher understanding scores. A higher manual artifact recovery rate was positively correlated with a higher understanding score ($p = 0.01$, large effect size). Manually recovered artifacts do not include artifacts directly recovered by an LLM. In fact, LLM-recovered artifacts did not show a similar correlation with manually recovered artifacts.

Finding 4. *LLM artifact recovery is not correlated with improved program understanding.* We measured LLM recovered artifacts as those that an LLM directly created. This excludes artifacts that a human may have manually created after reading an LLM answer. These LLM changes were not correlated with improved understanding ($p = 0.34$). This finding may indicate that LLM recovered artifacts are not of the same quality as those created by humans. We note that this analysis ignores human changes resulting from LLM changes, which would increase understanding (explored in Section VII-B).

Finding 5. *LLM users recover more artifacts, including more false positives.* When combining the artifacts recovered by LLMs and humans, we found that the presence of an LLM led to more overall artifacts on a challenge. Both experts and novices significantly differed in their total recovered artifacts while using LLMs ($p = 0.01$, $es = 0.52$, large effect size, and

$p = 0.01$, $es = 0.43$, medium effect size, respectively). On average, experts went from 54 artifacts to 84, while novices went from 41 to 67. Although these findings seem intuitive, they indicate that LLMs may not be replacing human efforts to recover artifacts. Considering that LLM recovered artifacts had no understanding correlation, our data may indicate LLMs create more artifacts that do not contribute, meanwhile increasing noise, which may be harmful.

An artifact type of particular interest was function names. Generally speaking, (good) function names summarize the purpose of a group of code. This abstraction allows SRE practitioners to understand what a function does quickly [7]. As such, creating a function name can often give insights about a participant's understanding of that function.

Finding 6. *LLMs recover function names of known algorithms with a higher accuracy than humans.* A *known algorithm function* implements algorithms whose designs and specifications are standardized and documented online, such as base64 decoding. Across both challenges, there was a total of 13 known algorithm functions.

We manually compared the renames from LLMs to those of humans on all 13 functions. A function rename was counted as a match if it was semantically equivalent. For example, with the ground truth being `handle_request`, a rename of `handle_connection` would be counted as semantically equivalent. LLMs showed a statistically significant difference in average rename accuracy for these functions when compared to humans ($p = 0.4$, medium effect size). On average, LLMs had an 85% accuracy, while humans 66%.

We also found that LLMs showed no significant difference in rename accuracy on all other functions when compared to humans ($p = 0.82$). Humans had an average accuracy of 65%, while LLMs had an average of 57% across 37 functions. Considering earlier LLM artifact correlations, we speculate the impacts of these renames may not significantly contribute to improved understanding.

D. Function Speed Differences

While solving both challenges, participants had to analyze various functions. We recorded both the total time participants spent in each function and the number of visits. We counted a visit as a participant looking at a function (in IDA Pro) for more than one second. On average, experts revisited a function 9.46 times and spent a total of 296.72 seconds in a function. Novices revisited a function 9.93 times and spent a total of 313.98 seconds in a function. Interestingly, we did not observe any correlation, suggesting a contribution from LLMs when viewing the sessions as a whole. Therefore, we focused on individual functions.

Finding 7. *Experts using LLMs spend less time on known algorithms and more time on custom ones.* At the binary level, experts showed non-significant performance when using an LLM. However, when comparing experts' total view times of functions across themselves, with and without LLM, they showed a significant difference ($p < 0.05$) in total view time on six functions. On four of these functions, LLM users were at least 248% faster on average. All four functions implemented common algorithms: TEA, Heapsort, Base64 Decode, and RLE compression. On Base64 Decode, experts also had two fewer visits on average.

On the other two (out of six) functions, which implemented custom algorithms, LLM users experience a slowdown. They were a minimum of 314% slower on average, and had to revisit one of those functions, a request handler, up to 14 times more on average. Although these gains did save experts time, we conclude, from previous findings, that these did not play a pivotal role in understanding the program faster.

Finding 8. *Revisit frequency scales strongly with lines of code for everyone, and LLMs rarely change that pattern overall.* Looking at the binary as a whole, revisits are positively correlated ($p < 0.05$) with lines of code, in *every* case: for all participants, experts, and novices, and those with and without LLMs (six instances in total). While this is an expected result, it is surprising that Spearman's ρ we obtained is very similar in all the tested cases, i.e., $0.76 \leq \rho \leq 0.77$ (large effect size). We further investigated at function granularity, and in the majority of cases, both novices and experts did not show significant differences ($p \geq 0.05$) in their function visit times with and without LLMs. They did, however, show differences in function visit amounts on two functions. On the first, `load`, a function from RPC that contained 37 lines to read the flag, novices had 53% less revisits with LLM. On the second, `ls_cmd`, a function from Secrets that contained 107 lines to implement a Unix-style file lister, novices had 240% more revisits (with and without LLM). We note that the `ls_cmd` function was not essential to solve the Secrets challenge. Indeed, mentioning the function, or its purpose in the writeup, did not necessarily earn understanding points. This is a clear sign of how experts' skills have prevented them from wasting time on this function.

E. Misunderstandings

In some solution writeups, participants described program functionality that was incorrect or did not exist, which we classify as a *misunderstanding*. In 20 out of 96 writeups (20%), a user had at least one misunderstanding in their writeup. These misunderstanding cases were split evenly between 10 LLM users and 10 non-LLM users.

We analyzed these instances of misunderstandings and classified each occurrence into two distinct groups: *fabrications*, which were observed when a participant described a non-existent phenomenon, and *misclassifications*, which occurred when a description was incorrect. **Finding 9.** *LLM vulnerability hallucinations negatively impact subsequent human analysis.* We observed three fabrications only in the responses of participants who used an LLM. Specifically, all three participants reported a buffer overflow vulnerability in their writeup for the RPC challenge, which had been suggested by the LLM when asked to identify potential vulnerabilities. All three participants received these suggestions in different functions. On these functions, they spent a minimum of 231% more time than average.

Indeed, asking an LLM to identify vulnerabilities frequently resulted in worse user performance. The "find vulnerability" feature was negatively correlated with understanding score ($p = 0.01$, $es = -0.35$, large effect size).

Additionally, the LLM feature to identify vulnerabilities drew the most skepticism ($n = 5$) of all LLM features in our study. It was widely seen as unreliable, prone to false positives, and lacking actionable specificity. E22 wrote, "It said there was a UAF, but it's not." N15 commented, "After using it, I felt at that point I was better off searching for the vulnerability myself." E10 similarly shared: "ChatGPT is a game changer for reversing [...] really good for code understanding, not good for finding complex vulnerabilities."

VII. LLM Strategies and Factors

In Section VI, we answered RQ2 showing how LLMs affect both novices and experts alike during the SRE process. A part of that analysis showed that novices approach a similar understanding rate to experts while using an LLM. In this section, we look at this phenomenon from a different angle by answering **RQ3:** *How do practitioners interact with LLMs, and what factors influence their interactions?*

Using a methodology inspired by previous work [1], we select the top ten and bottom ten participants by their understanding rate, on their LLM-enabled challenge, and we examine strategies and factors that contributed to improved or worsened performances. The top ten performers included five novices and five experts, whereas the bottom ten comprised nine novices and one expert.

A. LLM Expertise

Before joining our study, all participants reported both their prior experience in SRE and LLM-assisted SRE. For LLM-assisted SRE experience, participants reported using LLMs rarely, sometimes, often, or always for SRE. We considered those who reported often or always to be experienced LLM users and all others to be inexperienced. On average, a participant reported sometimes using an LLM, indicating they are inexperienced LLM users.

Finding 10. *Prior experience in using LLMs does not make them more useful for SRE.* When comparing the top and bottom performers, we found no significant difference between their reported LLM-SRE expertise ($p = 0.78$). When expanding this to the entire dataset of participants, we found no correlation between LLM experience and understanding rate or solve time ($p = 0.25$, $p = 0.41$). Both comparisons suggest that having prior experience in using LLMs may not increase their usefulness significantly over that of new LLM users. This may also suggest that LLMs are intuitive enough to use, that new users derive the same benefits from them as experienced ones.

Finding 11. *Experienced LLM users are more cautious about using LLMs.* Although experienced LLM users were not significantly better at SRE with LLMs, they did show differences in how much they relied on them. We found that LLM expertise (higher being more experienced) was negatively correlated with LLM usage amount ($p = 0.04$, $es = -0.3$, large effect size). On average, an experienced LLM user queried an LLM 20 times during a challenge, while an inexperienced user queried it 35 times. We note that this difference did not correlate with improved understanding, suggesting that this caution may have been inconsequential.

Combining these two findings, we conclude that LLMs may not require special training for usage in SRE and can be used effectively by novices. However, it may be harder to convince prior LLM users to keep using them widely, since distrust may be contributing to their caution, as explored in Section VI-E.

B. LLM Features

In our study, participants had access to the six LLM features (presented in Section IV) and a free-prompt LLM chat. Across all participants, the most frequently used task was largely `Func Summary` (625), followed by `Func Rename` (328), `Var Rename` (287), `Lib Func Docs` (84), `Func Identify` (84), `LLM Chat` (75), and `Func Vulns` (34).

Finding 12. *Understanding does not hinge on the quantity of LLM queries.* At a high level, no feature was utilized more across the top and bottom performers. When comparing the amounts of queries both for each feature and also all together, we found no significant difference between the two groups ($p > 0.05$). On average, top performers queried 29 times, while the bottom performers queried 37 times. However, these two groups still performed significantly different ($p = 0.01$, $es = 1.0$, large effect), with a 0.17 and 0.02 understanding rate on average respectively. The data show that sheer volume of LLM queries has no predictive power: the decisive factor is the placement of queries, i.e., knowing which function to ask about and when to ask, rather than how often any given feature is invoked.

Finding 13. *Asking LLMs clarifying questions is still an important LLM feature.* The `LLM Chat` feature is different from other features because it allows participants to freely interact with the LLM, where users can provide arbitrary context. We analyzed participants' use of the `LLM Chat`, which was used a total of 75 times, and categorized the users' interactions by their purpose with respect to the six features. However, the largest share of queries (40.6%) was context-related, and we were unable to associate them with a feature (e.g., hash-breaking strategies and explanations of regular expressions). This indicates a firm reliance on the LLM for problem-solving support and help with articulating technical concepts. Then, the most common category/feature was `Lib Func Docs`, accounting for 25.0% of the queries, suggesting that participants frequently relied on the LLM as a quick-reference tool for detailed questions about specific functions. This was followed by `Func Summary` (15.6%), `Func Identify` (12.5%), and `Func Vulns` (3.1%).

During reversing, participants switched between LLM actions (queries) and human actions (artifact recovery). Some actions used in succession led to increased understanding rates. We investigated these action sequences in pairs. The heatmap

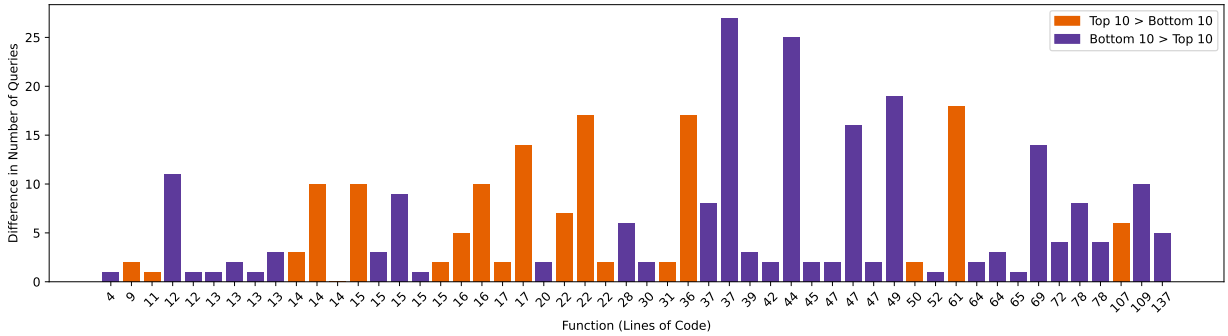

Figure 3: The difference in query amount by the top 10 and bottom 10 LLM users, sorted by function lines of code.

in Figure 6 visualizes these pairs, focusing on which led to human actions.

Finding 14. *LLM Func Summary and Var Rename lead to better program understanding.* Across all participants, three LLM features were correlated with an increase in understanding rate when accompanied by a human change: (`Func Summary` $\rightarrow$ function rename), (`Func Summary` $\rightarrow$ function retyping), and (`Var Rename` $\rightarrow$ function rename). They had the following statistical results: ($p = 0.01$, $es = 0.40$, large effect), ($p = 0.02$, $es = 0.32$, large effect), and ($p = 0.01$, $es = 0.42$, large effect), respectively. Notably, an LLM summarization followed by a manual function rename had the highest occurrence of 154, which dwarfed the next largest of just 19 with an LLM variable and a function rename.

E1 appreciated "the ability of LLM to effortlessly summarize the function, renaming local variable names, and renaming function name." N13 found Var Rename feature useful to "filter out decompilation statements." Similarly E7 commented, "Variable names were a lot more useful than I expected; a quick way to increase readability." According to N21, "summarize function combined with rename variables is a GOATED combo that made everything easier."

Of all action tuples involving LLMs, only these three tuples had positive effect on understanding rates. The common occurrence of all of them is the ending human action. This finding aligns with earlier findings in Section VI-C that manually recovered artifacts were correlated with improved understanding. From this, we conclude that LLM responses which led to human actions were likely the most helpful towards increasing understanding.

Finding 15. *Experts overestimate the usefulness of some LLM features.* In the post-study survey, participants ranked their perceived usefulness of features. We asked for both an explicit score (on a five-point Likert scale) and a free response if they felt anything was notable. All reported post-study scores can be found in Figure 5 in the Appendix.

In only two cases, Func Summaryand Func Vulns, user opinions aligned strongly with our observed results: summarization was useful and vulnerability identification was harmful. After the Func Summary feature ($n = 24$, § VI-B), Lib Func Docs ($n = 8$), Var Rename ($n = 8$), and Func Rename ($n = 5$) features were found helpful by our participants for improving code readability in the free response.

E11 mentioned "the [document] library call man page [feature] was useful in speed[ing] up the reversing process." Novices had contrasting view about Lib Func Docs feature. N12 appreciated the feature :"the ability to summarize functions and write man page descriptions directly in the decompiled view as comments was an excellent feature." N19, on the contrary said, "summarize library call man page injected big blocks of text as comments in the decompiled view which made it hard to read, preferred the color highlighted man pages on the web instead." We note that across all experts, there was no notable change in performance–as we have previously demonstrated, yet, many quotes and Likert-scale responses from experts contradict our observed results. Additionally, features such as manual page recovery played a lesser role than many participants thought.

C. Query Frequency

Although the best and worst performers had similar overall LLM usage, they differed when analyzed on a per-function basis. One specific aspect was the number of times users were prompted on a single function.

Finding 16. *LLMs have degrading benefits when utilized repeatedly on individual functions.* Top performers had fewer repeated query uses across all functions than bottom performers. Their average query use per function significantly differed ($p = 0.0004$, $es = -0.94000$, large effect size), with the top users using 1.71 queries and the bottom users using 3.43. This data may extend the findings in Section VII-C, furthering that less query use on individual functions may allow better understanding rates among participants.

Some participants reflected this fear of over-querying and over-reliance on LLMs: as E8 admitted, "I was prone to try to go faster and be less careful, relying a bit too much on the LLM support... sometimes it caused me to miss important details." This issue was particularly found concerning for less experienced users, as cautioned by E9: "I wouldn't want a novice using it, as I'm afraid they would reverse a lot of programs without actually understanding what's going on."

The most frequently queried functions were `dispatch` and `decompress` from the RPC challenge, each queried

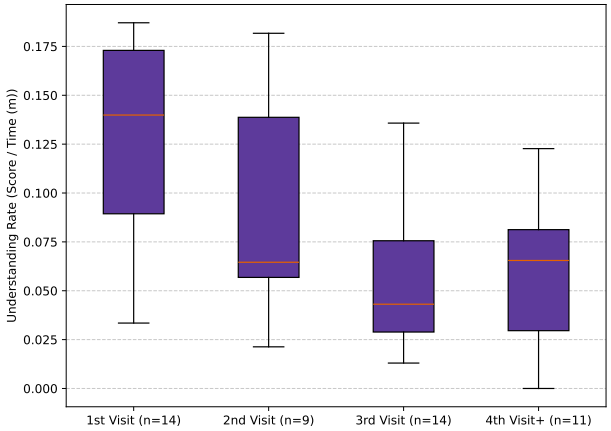

Figure 4: Understanding rates for participants that complete all queries on their first function visit vs on multiple visits.

22 times. Specifically, `dispatch` handles the user's operation command, while `decompress` performs RLE decoding. Similarly, `base64_decode` and `tea_decrypt`, also from RPC, were queried 16 and 18 times, respectively. These functions all play a central role in the challenge logic, and their high query count may reflect areas of conceptual complexity or critical decision points for participants.

Finding 17. *LLMs perform worse on larger functions (greater than 40 lines of code).* The size of functions, which often relates to complexity [44], was also a factor in query reuse. For both challenges, the median, mean, and max lines of code for functions were 31, 38, and 137, respectively. The top performers significantly differed from the bottom performers in their use of LLMs on functions larger than the median size ($p = 0.0003$, $es = -0.57988$, large effect size). In Figure 3, the *difference* in query amount can be seen on each function across both challenges, sorted by lines of code. Interestingly, the top performers used LLMs *less* on larger functions than the bottom performers, with an average of 4.73 vs 9.69.

D. Query Timing

One factor of query use was the time at which a participant decided to utilize an LLM while reverse engineering a function. To better quantify the time spent reversing on a function, we also tracked the number of visits (leaving a function and then re-entering it) participants had on functions. We observed that many top participants used all their LLM queries on their first visit to a function. To further investigate, we measured the number of visits each participant made to a function before using their last LLM query, and we categorized the strategy accordingly. For example, if a participant completed all of their LLM queries in three visits to a function, they would implement a three-visit strategy for most functions. The aggregated understanding rates of each visit strategy can also be seen in the Figure 4.

Finding 18. *LLMs provide greater benefit when utilized at the beginning of function understanding, not the end.* Out of the top ten performers of all LLM users, 9 are first-visit strategy users (out of a total of 11 first-visit strategy users). Moreover, first-visit strategy users significantly differed in their understanding rates from all other participants ($p = 0.01$, $es = 0.45$, medium effect size). On average, they had an understanding rate of 0.13, while all users had 0.08. This data indicates that LLM use at first glance of a function may produce better overall results.

Users who adopted the first-visit strategy primarily relied on the `Func Summary` prompt, which accounted for 41.20% of their queries. This was followed by `Var Rename` (18.92%) and `Func Rename` (21.62%). In contrast, the use of `LLM Chat` was relatively limited (4.94%), with `Lib Func Docs` (5.54%), `Func Vulns` (2.24%), and `Func Identify` (5.54%) used even less frequently.

E. Other Strategies

Participants also described other LLM strategies that they felt had a significant effect on their SRE process. E7 noted, "When I renamed a few things first, I think the LLM suggestions got better." E12 noted that they observed improved results when they added manual context, such as function comments, saying 'I often had to add a bit of context in a hack-y way." However, we found no correlation between any human-action-to-LLM chain and understanding score, suggesting that this may be a perception rather than a reality.

E22 shared their strategy of working from the inside out, naming variables and functions in the deepest functions first, then moving outward, also known as a depth-first search (DFS) strategy, and added that "naming them from inner function to its caller provided better results." Previous work has shown DFS to be an effective strategy for humans in SRE [1], and we found indications that this may also be true for LLM use. When comparing visit time on individual functions, we found that functions queried in a DFS strategy versus a linear strategy had a statistical difference ($p = 0.001$, $es = -0.24702$, small effect size). The averages for each were 25.82 seconds and 46.83 seconds, respectively, though a small effect indicates that there may not be a substantial difference in results.

VIII. Key Takeaways

We conducted the first empirical study comparing humans, both experts and novices, with and without LLM support during software reverse engineering. In this section, we synthesize the major themes emerging from our findings and discuss their broader implications.

I) LLMs primarily shape the first moments of understanding, not deeper refinements. Our findings indicate that the SRE process is often disproportionately influenced by the first encounter with a function. Early impressions frequently determine the pace and quality of subsequent analysis, affecting both overall solve time and understanding rate. LLMs are well-suited to this stage, as they enable practitioners to obtain a first-pass overview at minimal cognitive costs. This effect is most evident among novices, whose understanding rates approach those of experts when assisted by LLMs (Findings 1, and 18).

However, this advantage does not extend to deeper or iterative understanding. When practitioners revisit functions or attempt to refine mental models across functions, LLM assistance yields diminishing returns and can even hinder progress (Finding 16). These results suggest that current LLMs act primarily as summarization tools rather than collaborators capable of supporting sustained reasoning across functions or binaries. As a result, LLMs accelerate early semantic grounding but do not meaningfully assist with the deeper analytical processes characteristic of expert-level SRE (Findings 2, and 7).

II) The current interaction model of LLMS in SRE does not support expert knowledge refinement. Despite increasing adoption, the prevailing interaction model for LLM-assisted SRE does not effectively support experts. Experts exhibited little improvement in understanding rate or overall performance when using LLMs, suggesting that the integration model itself is insufficient (Finding 2). Experts selectively offload routine pattern recognition tasks, such as identifying known algorithms, while reallocating effort toward bespoke or complex logic. This results in effort redistribution rather than net acceleration (Finding 7).

These observations highlight a limitation of current LLM integrations, which primarily treat LLMs as first-pass summarizers rather than tools for iterative refinement or collaborative hypothesis development. Without advances that enable meaningful knowledge refinement, LLMs are likely to remain assistants for novices rather than true teammates for experts.

III) Even rare hallucinations can severely derail SRE workflows. Although hallucinations occurred infrequently across thousands of LLM interactions, their impact was disproportionately harmful. In particular, hallucinated vulnerability reports significantly disrupted participants, often leading analysts to pursue nonexistent flaws for extended periods (Finding 9). This behavior is especially concerning in SRE, where practitioners frequently explore competing hypotheses and cannot easily invalidate theories without substantial investigative effort.

IV) Semantic recovery often depends on the act of naming, not merely the presence of names. Artifact recovery, including function names, variable names, types, and comments, plays a central role in SRE. While LLMs substantially increase the total number of recovered artifacts, these automatically generated artifacts are not correlated with improved program understanding (Findings 4, and 5). In contrast, artifacts created manually by practitioners show a strong positive relationship with understanding (Finding 3).

We hypothesize that this discrepancy arises because artifact creation is itself an understanding process. The act of naming requires the analyst to commit to a hypothesis about a function's role, even when imperfect. When LLMs generate artifacts automatically, this process is partially bypassed, potentially reducing opportunities for deeper comprehension. While LLM-generated names are effective for identifying known algorithms (Finding 6), reliance on automatically generated artifacts in more complex contexts may hinder understanding.

V) Only certain forms of cognition can be effectively offloaded to LLMs. Our findings reveal clear limits on which aspects of SRE can be effectively offloaded to LLMs. Tasks that admit a concise, single-purpose abstraction, such as summarizing well-scoped functions or identifying standard algorithms, are well suited to LLM assistance and consistently beneficial (Findings 6, and 18). In contrast, functions that serve multiple roles, have been heavily optimized, or incorporate intertwined logic, remain challenging for LLMs to characterize accurately (Finding 17).

These results suggest that LLMs are most effective for compressible cognition, where behavior can be meaningfully captured in a single abstract description. For more complex code, successful SRE still requires iterative human analysis, potentially complemented by selective and early LLM use.

IX. Related Work

Human Cognitive Processes during SRE. Mantovani et al. [1] investigated how human experts approach and solve SRE tasks. Although their study primarily focused on analyzing cognitive strategies, it provided a foundational reference for the design of our user study. Additional related work on characterizing mistakes in the SRE process [45] and on the automated analysis of instrumented disassembler SRE data via screen recordings [46] further informed our effort to understand human behavior in SRE.

In contrast, our work differs significantly in scope and methodology: we introduce more realistic and semantically rich challenges with higher code complexity, and our participants interact with a broader set of tools – most notably, decompilation and LLM integration – which fundamentally alter the nature of the SRE process and the cognitive dynamics involved.

Other researchers [24] evaluated whether the adoption of external machine learning-based tools can improve human performance in discriminating a malicious program from a benign one. The authors find that machine learning algorithms can be a valuable tool for malware classification but should not be used to replace human analysts entirely. While Aonzo et al. [24] compare the independent decision-making processes of humans and machines, we delve into how LLMs can assist human analysts throughout the RE process.

LLMs for SRE. Since the recent advent of LLMs, we have seen an increasing evaluation of the capabilities of LLM-based tools, some of which have also considered the SRE scenario. Concerning end-to-end SRE, Pearce et al. [4] explored the ability of LLMs to identify program purposes, capabilities, and variables from decompiled code. Their findings indicate that, while LLMs show promise, their performance is insufficient for reliable zero-shot reverse engineering. Their work establishes that LLMs can perform some initial SRE tasks, but they do not study how their results may be used or perceived by human SRE practitioners. Our work explores this interaction and the more human-driven use of LLMs in SRE.

Amodei et al. [9] propose a novel technique to recover variable names from binaries, leveraging the strengths of both

generative models and program analysis. They evaluated their technique on a dataset of 940k binary functions and showed that it outperforms state-of-the-art techniques. The recent paper by Peiwei et al. [6] directly addresses the synergy between decompilers and LLMs for SRE. Their focus on optimizing decompiler output with LLMs aligns with our goal of improving code readability during the SRE process. Similarly, Shang et al. [5] evaluated LLM's ability to understand stripped binary code, focusing on tasks such as function name recovery and code summarization. Their study revealed the challenges posed by complex and obfuscated code, emphasizing the need for further research to improve LLM performance in this domain. Rukmono et al. [47] introduce an approach to summarizing software systems at higher levels of abstraction. They combine the capabilities of LLMs with static code analysis to generate summaries of software components.

LLMs are also used to find vulnerabilities in code [48]. Researchers have mostly focused on C/C++, Java, and Solidity, in particular on memory-related, framework-specific, and smart contract vulnerabilities, respectively. However, most attempts target function-level and file-level vulnerabilities, while there is a lack of repository-level datasets, thus limiting the adoption in real-world scenarios.

Our work builds upon the existing state of the art and offers a deeper understanding of the practical applications of LLMs during the whole SRE process.

User-Study on LLMs. Researchers have explored how users interact with LLM-based solutions for software development and analysis. LLMs have been used to improve developer productivity. GILT [49] introduced a prototype tool that integrates information retrieval with LLMs to improve information search during software development. The authors demonstrated the effectiveness of the tool in assisting developers with various tasks. Nguyen et al. [50] investigated how students interact with LLMs for code generation, highlighting challenges in prompt engineering and the misalignment between human and machine understanding. Furthermore, Zamfirescu-Pereir et al. [51] explored the difficulties non-AI experts encounter when designing prompts.

Previous user studies have focused on measuring user interaction with LLM-enhanced tools at individual stages of the SRE process or software development. This paper focuses on the entire SRE process workflow.

X. Limitations

While designing our study, we had to make several choices to balance the feasibility of the study with the amount of data we could collect. These choices may have introduced bias into our results.

Restricted Tooling. In the SRE sessions of our study, we limited all participants' analyses of challenges to static analysis. In real-world SRE, practitioners often have access to dynamic analysis tools, such as debuggers, which enable them to confirm assumptions through direct observation. They may also have access to other, more advanced tools, like symbolic execution frameworks. Since we did not allow participants to use these types of tools, they may have solved challenges in alternative ways to their normal workflow. This limitation can contribute to longer solve times and understanding rates.

Additionally, we did not survey our participants to determine if they were satisfied with a static-only approach to reverse engineering. We speculate that many would prefer more tools when reverse-engineering. However, to limit confounding variables in our study, we limited what tools participants could use, similar to prior work [1].

Challenge Representativeness. We designed two CTF-style challenges for our study that approximate tasks SRE practitioners may encounter in real-world programs. Although the majority of participants reported that they are representative of real programs (Section V), our challenges can still lack characteristics of real-world software. Our challenges contain no obfuscation, have limited complex mathematics, and are smaller than modern statically compiled programs.

Self-Reported Expertise. To determine SRE expertise, we relied on self-reporting and partially confirmed results by statistically testing the difference between the two groups. However, the difference between these two groups may not necessarily indicate that one is a novice and the other an expert. Similar to previous works [1], this could have led to the incorrect classification of experts and novices in SRE. Future work should explore more effective experiments for benchmarking and determining expertise levels in SRE.

LLM Prompts. As with any research applying LLMs, the style and wording of prompts may influence the results. During the SRE sessions, participants primarily relied on prompts written by the research team. These prompts were adapted from previous research, but they pose the risk of being more or less effective than the prompts practitioners currently use. As such, some scenarios of LLM use in our study may yield better or worse performance, which could bias the results.

XI. Conclusion

Our study takes the first in-depth empirical dive into how LLMs impact SRE and demonstrates that they *augment* analysts rather than *replace* them. The benefits are clear: LLMs support SRE practitioners in many tasks and can even close the gap in some aspects between experts and novices. However, our study shows that many areas of LLM integration still require research and development to be helpful for the most challenging SRE tasks. We present our findings, hopeful that the SRE community will utilize our work to make LLMs an even more helpful tool for security. All in all, LLMs are neither oracles nor impostors, but mirrors held to human insight: their reflections sharpen only in the steady gaze of critical thought and domain expertise.

XII. Open Science

To protect the privacy of our participants, individual observed actions from our empirical study are not available to the public, nor upon request. This includes all data generated by individuals in our SRE environment, all write-ups, and individual responses to our pre- and post-study surveys. However,

we make aggregated data, such as the total number of clicks in the study, available upon request. Additionally, our entire SRE environment infrastructure, including all of our tools, prompts, and challenges in the study, is made open source and available at https://github.com/mahaloz/dec-synergy-study. An actively maintained version of the LLM decompiler plugin is also available at https://github.com/mahaloz/DAILA.

XIII. Ethics 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). Our study is approved under "Continuing Review," which requires yearly re-approval to ensure continued ethical and safety compliance until all research activities (including reporting of findings) are complete.

In accordance with our IRB's requirements and our own ethical considerations of the proper handling of our participants, all of them were fully informed of the study 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 best security practices. No sensitive or personally identifying information was collected beyond contact details necessary for compensation.

The study tasks were designed to pose minimal psychological or professional risk, with no deception or stress-inducing elements. Participants were not required to complete the two challenges in sequence, as they could take a break of any duration between them. Finally, all results were analyzed and reported in aggregate form to prevent any possibility of reidentification. Even when participant perceptions (qualitative insights from the open responses) are reported about their experiences in the experiment, we use random participant identifiers.

Acknowledgments

This research was supported in part by the Advanced Research Projects Agency for Health (ARPA-H) under contract SP4701-23-C-0074; the National Science Foundation (NSF) under grants 2232915 and 2146568; the U.S. Department of the Interior under Grant No. D22AP00145-00; the U.S. Department of the Navy under grant N00014-23-1-2563; and by the Defense Advanced Research Projects Agency (DARPA) in collaboration with the Naval Information Warfare Center Pacific (NIWC Pacific) under contract N66001-22-C-4026. Additional support was provided by the U.S. Department of Defense and Google academic research funding.

This work also received funding from the French National Research Agency (ANR) under the France 2030 program through grants ANR-22-PECY-0007 (DefMal) and ANR-23-IAS4-0001 (CKRISP). Further support was provided by the European Union NextGenerationEU through the project SEcurity and RIghts In the CyberSpace (SERICS) (PE00000014–CUP H73C2200089001), as part of the National Recovery and Resilience Plan (NRRP), and by the 2023 STARS Grants UniPD programme through the PatchThemAll project.

Finally, we would like to thank all the participants who took part in our user study and Riccardo Bonavigo for helping with the challenges' design.

References

[1] A. Mantovani, S. Aonzo, Y. Fratantonio, and D. Balzarotti, "RE-Mind: a first look inside the mind of a reverse engineer," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 2727–2745. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/mantovani

[2] D. Votipka, S. Rabin, K. Micinski, J. S. Foster, and M. L. Mazurek, "An observational investigation of reverse Engineers' processes," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1875–1892. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/votipka-observational

[3] T. Nosco, J. Ziegler, Z. Clark, D. Marrero, T. Finkler, A. Barbarello, and W. M. Petullo, "The industrial age of hacking," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1129–1146.

[4] H. Pearce, B. Tan, P. Krishnamurthy, F. Khorrami, R. Karri, and B. Dolan-Gavitt, "Pop quiz! can a large language model help with reverse engineering?" 2022. [Online]. Available: https://arxiv.org/abs/2202.01142

[5] X. Shang, S. Cheng, G. Chen, Y. Zhang, L. Hu, X. Yu, G. Li, W. Zhang, and N. Yu, "How far have we gone in binary code understanding using large language models," in *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2024, pp. 1–12.

[6] P. Hu, R. Liang, and K. Chen, "Degpt: Optimizing decompiler output with llm," in *Proceedings 2024 Network and Distributed System Security Symposium (2024). https://api. semanticscholar. org/CorpusID*, vol. 267622140, 2024.

[7] L. Jiang, X. Jin, and Z. Lin, "Beyond classification: Inferring function names in stripped binaries via domain adapted llms," in *Proceedings 2025 Network and Distributed System Security Symposium (2025). https://api. semanticscholar. org/CorpusID*, 2025.

[8] D. Xie, Z. Zhang, N. Jiang, X. Xu, L. Tan, and X. Zhang, "Resym: Harnessing llms to recover variable and data structure symbols from stripped binaries," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 4554–4568.

[9] X. Xu, Z. Zhang, Z. Su, Z. Huang, S. Feng, Y. Ye, N. Jiang, D. Xie, S. Cheng, L. Tan *et al.*, "Unleashing the power of generative model in recovering variable names from stripped binary," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2025.

[10] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "PentestGPT: Evaluating and harnessing large language models for automated penetration testing," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 847–864. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/deng

[11] Z. Li, S. Dutta, and M. Naik, "Llm-assisted static analysis for detecting security vulnerabilities," *arXiv preprint arXiv:2405.17238*, 2024.

[12] Ivan Kwiatkowski, "Gepetto," https://github.com/JusticeRage/Gepetto, Accessed December 17, 2025.

[13] Tim Blazytko, "ReverserAI," https://github.com/mrphrazer/reverser_ai, Accessed December 17, 2025.

[14] Atredis Partners, "aiDAPal," https://github.com/atredispartners/aidapal, Accessed December 17, 2025.

[15] Evyatar E., "GptHidra," https://github.com/evyatar9/GptHidra, Accessed December 17, 2025.

[16] R. Zhang, N. J. McNeese, G. Freeman, and G. Musick, ""an ideal human" expectations of ai teammates in human-ai teaming," *Proceedings of the ACM on Human-Computer Interaction*, vol. 4, no. CSCW3, pp. 1–25, 2021.

[17] J. Börstler and B. Paech, "The role of method chains and comments in software readability and comprehension—an experiment," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 886–898, 2016.

[18] K. Yakdan, S. Dechand, E. Gerhards-Padilla, and M. Smith, "Helping johnny to analyze malware: A usability-optimized decompiler and malware analysis user study," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 158–177.

[19] Z. L. Basque, A. P. Bajaj, W. Gibbs, J. O'Kain, D. Miao, T. Bao, A. Doupé, Y. Shoshitaishvili, and R. Wang, "Ahoy sailr! there is no need to dream of c: A compiler-aware structuring algorithm for binary decompilation," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 361–378.
[20] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
[22] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du *et al.*, "Lamda: Language models for dialog applications," *arXiv preprint arXiv:2201.08239*, 2022.
[23] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
[24] S. Aonzo, Y. Han, A. Mantovani, and D. Balzarotti, "Humans vs. machines in malware classification," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 1145–1162. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/aonzo
[25] V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. Sjøberg, "A systematic review of effect size in software engineering experiments," *Information and Software Technology*, vol. 49, no. 11-12, pp. 1073–1086, 2007.
[26] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: a practical and powerful approach to multiple testing," *Journal of the Royal statistical society: series B (Methodological)*, vol. 57, no. 1, pp. 289–300, 1995.
[27] J. L. Gastwirth, Y. R. Gel, and W. Miao, "The impact of levene's test of equality of variances on statistical theory and practice," *Statistical Science*, vol. 24, no. 3, pp. 343–360, 2009.
[28] J. Cohen, *Statistical power analysis for the behavioral sciences*. routledge, 2013.
[29] L. V. Hedges, "Distribution theory for glass's estimator of effect size and related estimators," *journal of Educational Statistics*, vol. 6, no. 2, pp. 107–128, 1981.
[30] P. Marfo and G. Okyere, "The accuracy of effect-size estimates under normals and contaminated normals in meta-analysis," *Heliyon*, vol. 5, no. 6, 2019.
[31] D. C. Funder and D. J. Ozer, "Evaluating effect size in psychological research: Sense and nonsense," *Advances in methods and practices in psychological science*, vol. 2, no. 2, pp. 156–168, 2019.
[32] J. Cohen, "A power primer," *Methodological Issues and Strategies in Clinical Research*, pp. 279–284, 2016.
[33] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014.
[34] M. P. Fay and M. A. Proschan, "Wilcoxon-mann-whitney or t-test? on assumptions for hypothesis tests and multiple interpretations of decision rules," *Statistics surveys*, vol. 4, p. 1, 2010.
[35] J. D. Gibbons and S. Chakraborti, *Nonparametric statistical inference: revised and expanded*. CRC press, 2014.
[36] M. W. Fagerland and L. Sandvik, "Performance of five two-sample location tests for skewed distributions with unequal variances," *Contemporary clinical trials*, vol. 30, no. 5, pp. 490–496, 2009.
[37] A. Hart, "Mann-whitney test is not just a test of medians: differences in spread can be important," *Bmj*, vol. 323, no. 7309, pp. 391–393, 2001.
[38] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions." *Psychological bulletin*, vol. 114, no. 3, p. 494, 1993.
[39] R. J. Grissom and J. J. Kim, *Effect sizes for research: Univariate and multivariate applications*. Routledge, 2012.
[40] Mattias Karlsson, "GhidraChatGPT," https://github.com/likvidera/GhidraChatGPT, Accessed December 17, 2025.
[41] Binary Ninja, "Binary Ninja Sidekick," https://sidekick.binary.ninja/, Accessed December 17, 2025.
[42] A. Ellard-Gray, N. K. Jeffrey, M. Choubak, and S. E. Crann, "Finding the hidden participant: Solutions for recruiting hidden, hard-to-reach, and vulnerable populations," *International journal of qualitative methods*, vol. 14, no. 5, p. 1609406915621420, 2015.
[43] C. Nelson and Y. Shoshitaishvili, "Dojo: Applied cybersecurity education in the browser," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 2024, pp. 930–936.
[44] M. H. Halstead, *Machine-independent computer programming*. Spartan Books, 1962.
[45] I. Ford, A. Soneji, F. B. Kokulu, J. Vadayath, Z. L. Basque, G. Vipat, A. Doupé, R. Wang, G.-J. Ahn, T. Bao *et al.*, ""watching over the shoulder of a professional": Why hackers make mistakes and how they fix them," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 350–368.
[46] T. T. Zhang, C. Taylor, B. Coppens, W. Mebane, C. Collberg, and B. De Sutter, "reanalyst: Scalable annotation of reverse engineering activities," *Journal of Systems and Software*, p. 112492, 2025.
[47] S. A. Rukmono, L. Ochoa, and M. R. Chaudron, "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, 2023, pp. 7–12.
[48] Z. Sheng, Z. Chen, S. Gu, H. Huang, G. Gu, and J. Huang, "Llms in software security: A survey of vulnerability detection techniques and insights," 2025. [Online]. Available: https://arxiv.org/abs/2502.07049
[49] D. Nam, A. Macvean, V. Hellendoorn, B. Vasilescu, and B. Myers, "Using an llm to help with code understanding," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3639187
[50] S. Nguyen, H. M. Babe, Y. Zi, A. Guha, C. J. Anderson, and M. Q. Feldman, "How beginning programmers and code llms (mis)read each other," in *Proceedings of the CHI Conference on Human Factors in Computing Systems*, ser. CHI '24, vol. 1. ACM, May 2024, p. 1–26. [Online]. Available: http://dx.doi.org/10.1145/3613904.3642706
[51] J. Zamfirescu-Pereira, R. Y. Wong, B. Hartmann, and Q. Yang, "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*, ser. CHI '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3544548.3581388

APPENDIX

A. Online Post-Study Survey Questionnaire

CTF Challenges: Participants were asked to indicate their level of agreement using a 5-point Likert scale ranging from *Strongly Disagree* to *Strongly Agree*.

- The two CTF challenges are good representative examples of real-world software.
- The security vulnerabilities in the two CTF challenges are good representative examples of real-world ones.
- The two CTF challenges were of similar difficulty.

Usability: Participants were asked to indicate their level of agreement using a 5-point Likert scale ranging from *Strongly Disagree* to *Strongly Agree*.

- Learning to use IDA Pro with LLM support was easy for me.
- My interaction with IDA Pro with LLM support was clear and understandable.

Contribution of the LLM: Participants were asked to indicate their level of agreement using a 5-point Likert scale ranging from *Strongly Disagree* to *Strongly Agree*.

- Using IDA Pro with LLM support allowed me to understand the functions of the program faster than when I did NOT have GPT support.
- I was more confident of the solution when I used IDA Pro with LLM.
- Different prompt engineering techniques have an impact on the quality of the suggestions.
- I felt that the LLM can be used instead of searching using the web browser.

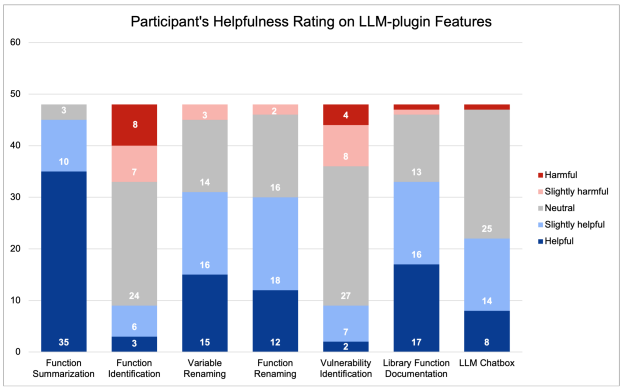

Figure 5: Participants' responses to a 5-point Likert scale from Helpful to Harmful for LLM-plugin features

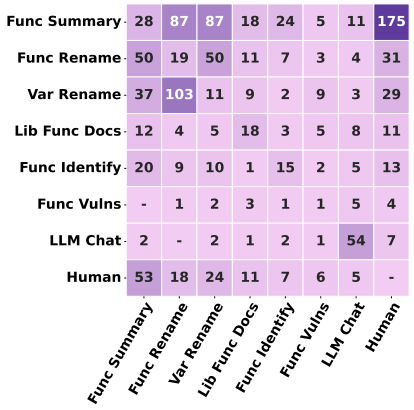

Figure 6: Heatmap of LLM action transitions, with the y-axis representing the initial action and the x-axis representing the subsequent action. Each cell indicates the number of transition occurrences across all participants. *(human to human action transitions were not included)

Usefulness of LLM-Enhanced Features: Participants were asked to rate the following features on a 5-point Likert scale ranging from *Very Harmful* to *Very Helpful*, based on how each supported their understanding of the target program.

- Summarize this function.
- Identify the source of this function.
- Suggest variable names.
- Suggest function name.
- Find vulnerabilities in this function.
- Summarize library call man page.
- Free prompt with GPT.

Open-Ended Reflection:

- What stood out to you about the experience of using IDA Pro with LLM support compared to the standard IDA Pro? For example, was anything good, bad, surprising, or notable?

Participant Information for Compensation Disbursement:

- What email should we send the gift card to?
- Are you a US citizen?

Table IV: A summary of key findings from the survey conducted on the study participants. Categories marked with an asterisk (*) are part of multiple-choice questions.

	Survey	
Total	153	
Age Range	**No.**	**%**
21-25	66	43.1%
26-30	34	22.2%
18-20	17	11.1%
36-40	13	8.5%
31-35	13	8.5%
40+	9	5.9%
50+	1	0.7%
Occupation*	**No.**	**%**
Academic/Student	66	43.1%
Industry/Employee	42	27.5%
Academic/Researcher	26	17.0%
Industry/Freelance	14	9.2%
Other	5	3.3%
SRE Experience (years)	**No.**	**%**
[1-3]	62	40.5%
[4-6]	28	18.3%
< 1	28	18.3%
[7-9]	17	11.1%
> 10	18	11.8%
SRE Frequency	**No.**	**%**
Sometimes - Once a week	56	36.6%

Table III: Participant Agreement Ratings on 5-point Likert Scale.
SA = Strongly Agree, MA = Moderately Agree, N = Neutral, MD = Moderately Disagree, SD = Strongly Disagree

Statement	**SA**	**MA**	**N**	**MD**	**SD**
1. Challenges represent real-world software	13	18	12	5	0
2. Security vulnerabilities in challenges are realistic	6	31	10	1	0
3. Challenges are similarly difficult	8	23	5	11	1
4. Learning to use IDA Pro with LLM support was easy.	32	13	1	1	1
5. Interacting with IDA Pro with LLM support was clear and understandable.	27	17	2	1	1
6. LLM helped me understand functions faster	33	11	3	1	0
7. More confident of the solution using LLM	15	19	9	4	1
8. Prompt engineering has an impact on the quality of the suggestions	14	10	22	1	1
9. LLM can be used instead of searching on the web browser	21	18	3	5	1

Often - Multiple days in a week	49	32.0%
Rarely - Once a month	23	15.0%
Always - Every day	19	12.4%
Very Rarely - Once a year	6	3.9%
Novices and Experts	**No.**	**%**
Novice	85	55.6%
Expert	68	44.4%
SRE Context*	**No.**	**%**
For hobby and/or passion	110	32.9%
Competitive (e.g., CTF)	87	26.0%
For work and/or research	79	23.7%
For academic studies	55	16.5%
Other	3	0.9%
Use of LLMs outside SRE	**No.**	**%**
Often - Multiple days in a week	61	39.9%
Always - Every day	35	22.9%
Sometimes - Once a week	26	17.0%
Rarely - Once a month	21	13.7%
Never - I do not use them at all	10	6.5%
LLMs Preference*	**No.**	**%**
GPT by OpenAI	140	57.1%
Claude by Anthropic	36	14.7%
LLaMA by Meta	27	11.0%
Other	24	9.8%
PaLM by Google	13	5.3%
Mistral by Mistral	5	2.0%
Program Format Popularity*	**No.**	**%**
ELF	124	53.4%
Portable Executable (PE)	69	29.7%
Other	27	11.6%
Mach Object (Mach-O)	12	5.2%
SRE Frameworks*	**No.**	**%**
IDA Pro	110	81.0%
Ghidra	107	73.0%
Binary Ninja	48	13.6%
angr	29	8.2%
Radare2	27	7.6%
Other	11	3.1%
JEB	10	2.8%
Rizin/Cutter	8	2.3%
Hopper	3	0.8%
Use of LLMs during SRE	**No.**	**%**
Sometimes	52	34.0%
Rarely	49	32.0%
Often	43	28.1%
Always	9	5.9%
LLMs used during SRE*	**No.**	**%**
GPT by OpenAI	131	64.2%
Other	27	13.2%
Claude by Anthropic	20	9.8%
LLaMA by Meta	15	7.4%
PaLM by Google	8	3.9%
Mistral by Mistral	3	1.5%
Most frequent queries to LLMs*	**No.**	**%**
Summarize a function	100	25.9%
Improve the readability of the code	76	19.7%
Identify the original source code	43	11.1%
Rename variables	41	10.6%
Rename functions	36	9.3%
Summarize a program	36	9.3%
Other	25	6.5%
Find vulnerabilities	29	7.5%
Type of code input*	**No.**	**%**
Source Code (decompiled)	132	60.0%
Machine Code (disassembled)	61	27.7%
Intermediate Language (lifted)	27	12.3%
Usefulness of using LLMs during SRE	**No.**	**%**
Sometimes yes, sometimes no	104	68.0%
Yes	36	23.5%
No	13	8.5%