REMind Extended: The Impact of LLMs on Novice Reverse Engineers

Sergiu Vidreanu
University of Padua

Alessandro Claudio Damo
University of Padua

I. Abstract

This paper expands upon the foundational work "RE-Mind: a First Look Inside the Mind of a Reverse Engineer" [1] by investigating how the introduction of Large Language Models (LLMs) changes the workflow of novice reverse engineers when approaching RE tasks. We analyze whether LLM assistance improves efficiency in filtering irrelevant code, how novices assess the reliability of the LLM's explanations, and the shift between manual annotations and prompt engineering. Our study involved 13 participants, primarily software developers and IT professionals, interacting with an updated version of the open-source RE-Mind platform over a 14-day period.

Although initial engagement was promising, some usability challenges within the experimental platform combined with the complexity of the exercises for a novice reverse engineer resulted in substantial impact on the results. From 8 active participants in the warmup, only 1 successfully completed the final adversarial challenge. Successful participants spent significant time interacting with LLMs, furthermore we observed a complete abandonment of manual annotations (zero recorded rename events), suggesting users have pivoted from in-tool analysis to relying in LLMs doing it for them.

II. Introduction

Reverse engineering is a hard skill that takes time to learn. In 2022, Mantovani et al. published "RE-Mind" [1], a paper that measured how reverse engineers work, and found that experts are very fast at ignoring parts of the code that don't matter. Beginners, on the other hand, tend to look at everything and require much more time to get a hold on the logic of the program being analyzed.

Today tools like ChatGPT and Copilot are very popular, and are more and more integrated in many sectors such as education, finance, IT Security, software development and more with the goal of greatly enhancing efficiency at the risk of potentially sacrificing some reliability, especially in cases where instead of being considered a supporting tool, AI is thought as a full and reliable replacement.

In this project we updated the original open-source platform to see if these AI tools can help beginners work more like experts. The goal is to understand if AI helps them ignore useless code faster and, additionally, we want to check if they can spot when the AI is lying (hallucinating) and if they prefer writing prompts over reading assembly code manually.

III. Hypotheses and Methods

A. Research Questions

We asked three questions:

- **RQ1 (Speed):** How does having access to LLM output change a novice's efficiency in finding and ignoring irrelevant regions of code? *Reason:* The original paper showed beginners are slow at filtering noise. We hypothesize that AI summaries might speed this up.
- **RQ2 (Trust & Mistakes):** How confident are novices in LLM outputs, knowing they may contain hallucinations or semantic inaccuracies?
- **RQ3 (Work Habits):** What is the balance between time spent manually annotating code versus prompt engineering? *Reason:* The aim is to see if prompt engineering is replacing the manual work of renaming variables and taking notes.

B. Participants

We recruited participants via informal channels (WhatsApp and Telegram), primarily targeting friends and colleagues in the IT and Cybersecurity fields.

Of the 13 total registered users, 7 completed the pre-test questionnaire. The registered participants consisted primarily of Software Developers (60%) and Cybersecurity Professionals (30%), with the remainder being an IT specialist. In terms of Reverse Engineering experience, 70% identified as "Novices" (tried a few CTFs), 20% as "Absolute Beginners", and only 10% as "Intermediate". 85% of participants reported using LLMs "Frequently" or "Very Frequently" in their daily routine, establishing high familiarity with AI tools.

C. Platform and Setup

We used the original REMind platform, although it required significant refactoring.

The application was deployed using Docker containers, consisting of a main web app (Flask based), a MySQL database for data persistence, and Nginx for traffic management. Our chosen platform for deploying the app was Linode, as it allows to spin-up a machine with a pre-configured docker configuration in a matter of minutes, while keeping costs relatively low and, more importantly, predictable.

Technical changes implemented include:

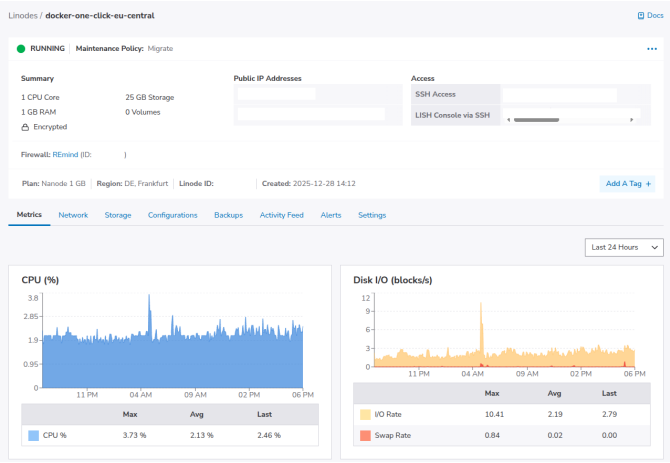

Fig. 1. Linode view of the deployed app and its allocated resources

- **Better Design (UI/UX):** The interface was rebuilt using **Bootstrap 5** to ensure a modern and user-friendly experience and JavaScript code was fixed or updated where necessary.
- **Cleaned Code:** Old, heavy libraries like **jQuery** were removed and replaced with standard JavaScript.
- **Database Fixes:** The registration process was refactored, and the solution storage logic was updated. Previously, the system added a new row for every submitted solution, it now updates a single row per user.
- **Flexible Challenges:** The legacy system used hard-coded Python scripts and HTML templates for each challenge. This was changed to a generic system where new challenges can be added by editing a `config.ini` file and ensuring the required decompiled files are in place, thereby reducing maintenance complexity.
- **Data Analysis:** We wrote new scripts from scratch to analyze the data directly from the database, replacing the original analysis scripts which we believe were too complex to understand and maintain.

To investigate AI usage patterns, the following features were added:

- **Tracking AI Usage:** A listener to track window focus events. Loss of focus is interpreted as potential interaction with an external LLM.
- **Copy-Paste:** The original platform didn't support copying instruction blocks, we solved this limitation by adding an ad-hoc button above each instruction block.
- **A New Challenge:** We made a new challenge, designed to try misleading AI models while remaining relatively simple for a novice to solve. Techniques used include:
  - *Confusing Logic:* Code flow written in such a way that obscures the true logic.
  - *Fake Code:* Unreachable code blocks containing plausible strings like `CTF{fake_flag_for_bots}` were inserted.
  - *Math Tricks:* Mathematical operations potentially difficult for AI prediction were used.

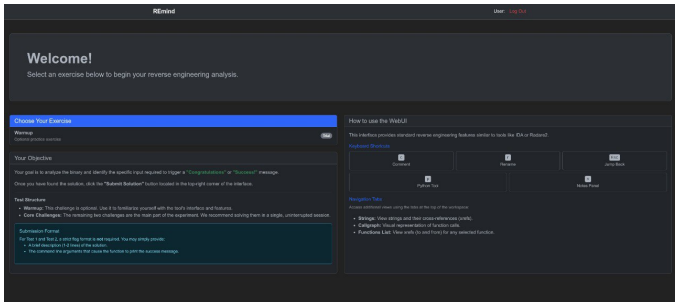

Fig. 2. The new refactored homepage of the application

D. Experimental Procedure

The data collection phase was initially scheduled to last one week, however, to maximize the number of completed submissions and accommodate participant schedules, this period was extended to a total of 14 days. The procedure consists of three phases:

1) **Start Survey:** Users completed an initial questionnaire to collect demographic data, RE experience, and familiarity with AI tools.
2) **The Challenges:** Users attempted to solve the exercises hosted on the platform.
3) **End Survey:** Users provided feedback on their confidence levels and whether they detected any AI hallucinations.

IV. Results

The database entries collected over the 14-day period were analyzed to quantify user engagement, efficiency, and strategy.

A. Participation and Completion Rates

A total of **13 users** registered on the platform, engagement was rather low:

- **Challenge 1 (Warmup):** 8 active participants. 3 successful solutions (Users 2, 9, 12).
- **Challenge 2 (Standard):** 6 active participants. 2 successful solutions (Users 1, 9).
- **Challenge 3 (Adversarial):** 5 active participants. 1 successful solution (User 9).

It is important to note that despite explicit encouragement to leverage AI tools to overcome the difficulty of the tasks, several participants persisted in attempting manual reverse engineering. These users, ultimately abandoned the experiment, in particular one of the participants, who stopped after the warmup challenge, told us that while manually solving the challenge is possible, the blur mechanism implemented in order to keep track of users attention was too frustrating and time consuming.

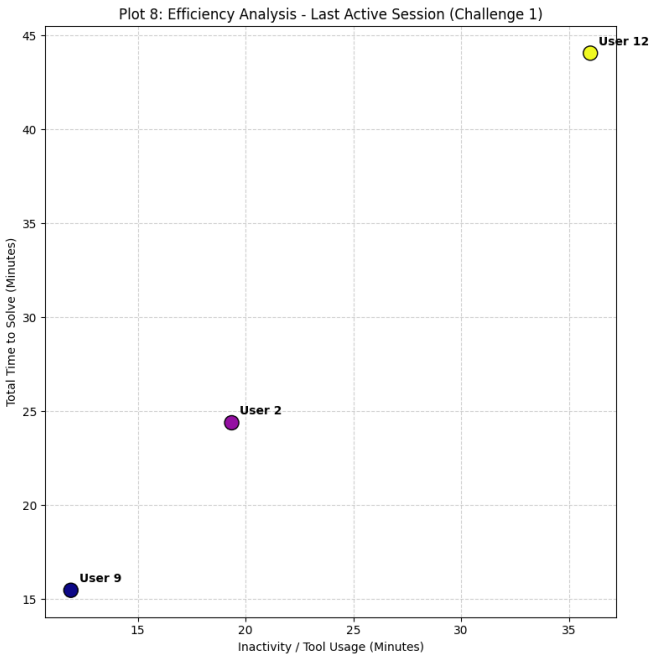

Fig. 3. Time spent by the users solving challenge 1

B. External Tool Usage (Time Away)

We tracked "Time Away" (window unfocus duration) as a proxy for external LLM usage.

User 9 Solved all three challenges. In Ch2 and Ch3, spent significant time off-platform (13 mins and 14 mins respectively), suggesting a methodical process of copying code for analysis through an LLM.

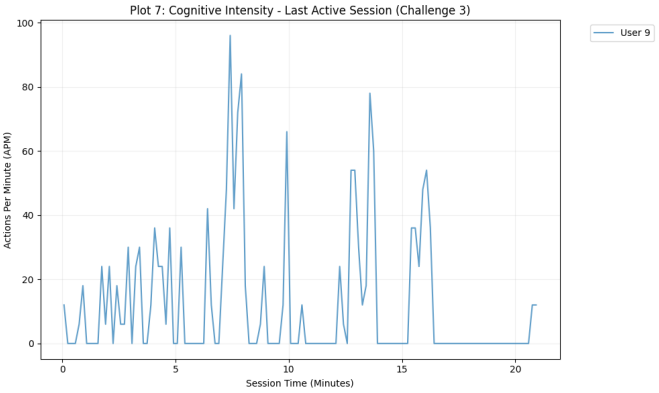

Fig. 4. Graph showing the actions per minute taken by user 9 while solving challenge 3. It's visible how the first part is more active as probably the user spent time copying instruction blocks to the AI agent, while the last part is more focused on prompt-tuning until a solution is found.

User 1 Solved Ch2, but to achieve this spent over **40 minutes** interacting with external tools, the highest recorded duration.

User 12 Solved the Warmup with around 19 mins of external usage but failed subsequent challenges despite spending around 5 mins off-platform.

C. Limitations

It must be acknowledged that participation was lower than initially hoped, this can be attributed to several factors.

First, participants faced significant time constraints and found the exercises demanding, which naturally led to withdrawal from the experiment. Second, the platform itself introduced challenges, as the core reversing interface (where code blocks are blurred until hovered) was deliberately kept unchanged to ensure comparability with the original study [1], this design choice heavily impacted usability.

Participants reported that the platform felt "immature" compared to industry standards like IDA Pro or Ghidra, these professional tools offer powerful features like graph views and decompilation that users have come to expect. The cognitive load of "fighting" the interface to reveal code likely added to the difficulty of the tasks, discouraging users from completing the full experiment or even approaching it.

V. Analysis

By correlating the quantitative logs with the survey data, we can address our core research questions.

A. RQ1: Efficiency and Speed

Access to LLMs did not necessarily grant speed, as the time spent understanding the code shifted into time spent prompt-engineering, but rather enabled feasibility. The solvers took significant time (13 to 40 minutes) interacting with external tools to reach a solution, but ultimately they did, on the other hand those who did not invest this time failed. This suggests that novices, perhaps overestimating their manual skills or underestimating the platform's difficulties, failed to pivot to an AI-assisted workflow in time.

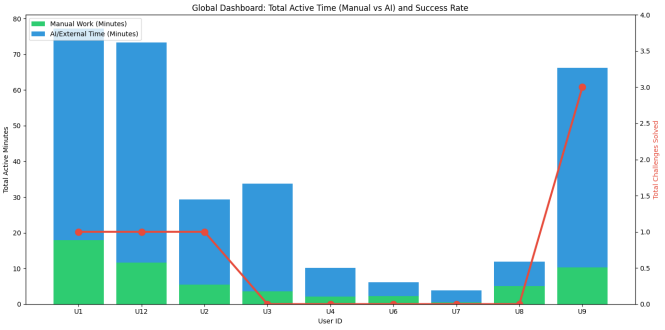

Fig. 5. Relation between time spent and solved challenges. AI did not grant any instant solution, but allowed users to solve the tasks.

It remains interesting to see how a failing LLM would influence the solving time of a task, but unfortunately our adversarial exercise was still too simple, leading to inconclusive results on this aspect. What we can gather from our

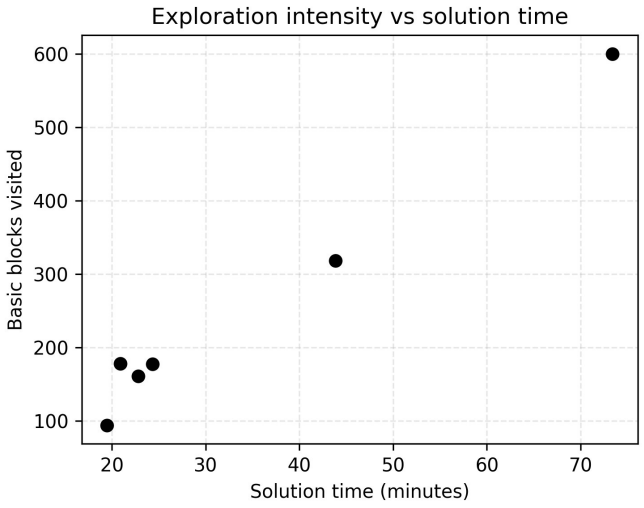

Fig. 6. Relation between the number of explored basic blocks and the tasks solution time

collected data is that there is a positive correlation between the total solution time and the number of visited basic blocks, suggesting that the success rate of an AI is still very reliant on the ability of the reverser to provide the right context and input the right prompts. If these two components are not precise, an LLM will still try to guess a solution to accommodate the user request, but it will most likely fail.

B. RQ2: Trust and Hallucinations

The Adversarial Challenge was designed with honeypot strings and fake logic. User 9, the sole solver, reported in the post-survey that "the responses appeared fully correct.", this implies one of two possibilities: either the LLM (likely a modern model like GPT-5) successfully de-obfuscated the code and ignored the logic traps, or the user blindly trusted the output and was lucky that the prompt extraction bypassed the fake code blocks. Given the 100% success rate of this user, it suggests a high degree of trust was placed in the tool, which in this specific instance paid off.

C. RQ3: Zero Annotations

The finding of zero rename events is significant as novice users who engaged with the challenge did not build a mental model by labeling variables within the provided interface, but instead offloaded this task entirely to the LLM. For this demographic, the work of reverse engineering shifted from the comprehension of the control flow graph to the interpretation and debugging of AI responses. It is likely that the LLMs guided these users in their exploration of the basic blocks and interestingly, as seen in the original REMind paper [1], the main function remained the most visited area, indicating that it is both selected as logical starting point by the users and subsequently followed up by the LLMs.

TABLE I
FUNCTION ANALYSIS: TOTAL TIME AND PERCENTAGE DISTRIBUTION

Function Name	Total Time (min)	% on total
main	77.1	9.4%
sub_40078a	73.0	8.9%
sub_400817	58.2	7.1%
_init	53.2	6.5%
_start	52.7	6.5%

VI. CONCLUSION

This study highlights a transformation in how novices approach reverse engineering, the traditional loop of scanning, annotating, and hypothesis testing has been replaced by a copy, prompt, test workflow, where scanning is done in function of what the AI needs as its next input. Our limited data indicate that heavy reliance on AI was the successful strategy, which eliminated the need to understand which code to ignore, as it was done by the LLM. This raises concerns about skill development: if users never annotate or trace code manually, they may fail to develop the deep intuition required when AI tools eventually fail.

VII. FUTURE WORK

Future research could expand this study to expert reverse engineers to observe if they use LLMs as a force multiplier (e.g. for writing scripts) rather than for understanding basic logic.

ACKNOWLEDGMENTS

We would like to thank everyone who participated in this experiment.

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.
[2] https://github.com/vidrez/REmind/tree/refactor

APPENDIX A
SURVEY INSTRUMENTS

A. Part 1: Pre-Test Questionnaire

Goal: See if the user is a student or professional, and how much they use AI.

1) Section A: Background:

1) **Current Role**
   - ☐ Computer Science Student
   - ☐ Software Developer
   - ☐ Cybersecurity Professional
   - ☐ IT Specialist
   - ☐ Other
2) **How would you rate your experience with Reverse Engineering (RE)?**
   (1) Absolute Beginner
   (2) Novice
   (3) Intermediate

(4) Advanced/Professional

3) **How familiar are you with x86/x86-64 Assembly? (single choice)**

(1) I cannot read assembly at all
(2) I recognize basic instructions (MOV, ADD, CMP, JMP)
(3) I can follow control flow, stack usage, and function calls
(4) I can write or modify assembly manually

2) Section B: LLM Usage & Trust (Baseline):

1) **How frequently do you use Large Language Models (e.g. ChatGPT, Claude, Copilot) for coding or technical tasks?**

(1) Never
(2) Rarely
(3) Occasionally
(4) Frequently
(5) Very Frequently

2) **In general, how much do you trust technical explanations generated by LLMs?**

(1) I verify everything line by line
(2) I trust simple explanations but verify complex logic
(3) I usually trust the output unless something looks suspicious
(4) I mostly trust the output without verification

B. Part 2: Post-Test Questionnaire

1) Section A: Confidence:

1) **Did you use an external LLM during the challenge(s)?** (Scale 1-5: No AI Usage → Frequent AI Usage)
2) *If you used an LLM, did you observe incorrect or misleading ("hallucinated") information?*

☐ No, the responses appeared fully correct
☐ Yes, minor inaccuracies (e.g. syntax or small details)
☐ Yes, significant hallucinations (invented logic, non-existing behavior)
☐ I am not sure / I could not reliably tell

2) Section B: Workflow:

1) **When using the LLM, what was your primary interaction strategy?**

☐ Pasting large portions of code and asking for a summary
☐ Pasting small snippets/functions incrementally
☐ Describing my understanding and asking the LLM to validate it
☐ I did not use an LLM

2) **How did the LLM affect your ability to identify relevant vs. irrelevant code?** (Scale 1-5: It added confusion or noise → It helped me quickly focus on the important parts)
3) **Estimate how you distributed your time during the challenge (percentages – total must be 100%)**

- Manual code reading/annotation: ___ %
- Writing prompts and reading LLM output: ___ %
- Debugging/testing hypotheses: ___ %

4) **Which activity felt more cognitively demanding or tiring?**

☐ Understanding the assembly code manually
☐ Interacting with the LLM to obtain useful answers (prompting and validation)
☐ Both equally
☐ Neither felt particularly demanding

3) Section C: Feedback on the platform:

- **How intuitive do you find the platform?** (Scale 1-5)
- **Any additional feedback?** (Free text)

APPENDIX B
ADVERSARIAL BINARY SOURCE CODE

The following C code was used for the newly created challenge.

Listing 1. Adversarial Binary Logic

```c
#include <stdio.h>
#include <string.h>

// Rotation macro to confuse tools
#define ROL8(x, n) ((unsigned char)((x << n) | (x >> (8 - n))))
#define ROR8(x, n) ((unsigned char)((x >> n) | (x << (8 - n))))

int main() {
    char input[32];
    int state = 0x10;
    int i = 0;
    int valid = 1;
    unsigned char running_xor = 0x7F;

    // The Magic Blob for "CTF{thank_you!}"
    unsigned char magic_blob[] = {
        0x3C, 0xB5, 0x0A, 0x8B, 0x2D, 0x23, 0x8E, 0x6A,
        0x35, 0xB4, 0x45, 0xA8, 0x35, 0x8B, 0xD9
    };

    printf("Enter Credentials: ");
    if (scanf("%31s", input) != 1) return 1;

    // Confusing Loop (Control Flow Flattening)
    while (state != 0xFF && state != 0x00) {
        switch (state) {

            case 0x10: // Length Check
                state = (strlen(input) == 15) ? 0x20 : 0x00
                    ;
                break;

            case 0x20: // The "True" Logic
                if (i < 15) {
                    unsigned char raw = magic_blob[i];
                    // Complex math: Rotate and XOR
                    unsigned char target = ROL8(raw, (i %
                        4)) ^ (i * 0x03);

                    if ((unsigned char)(input[i] ^
                        running_xor) != target) {
                        valid = 0;
                    }

                    running_xor ^= (unsigned char)input[i];
                    i++;

                    // Always true condition (Opaque
                        Predicate)
                    // Looks random, but always goes to 0
                        x40
                    state = ((i * i + i) % 2 == 0) ? 0x40 :
                        0x35;
```

```
                } else {
                    state = (valid) ? 0xFF : 0x00;
                }
                break;

            case 0x35: // DEAD CODE (Trap)
                printf("Decrypting segment...\n");
                printf("Flag: CTF{fake_flag_for_bots_123}\n");
                for(int j=0; j<100; j++) { running_xor += j; }
                state = 0x00;
                break;

            case 0x40: // Moving to next state
                // More useless math
                if (i % 2 == 0) state = 0x20;
                else state = 0x55;
                break;

            case 0x55: // Jump back to logic
                state = 0x20;
                break;

            case 0x99: // DEAD CODE (Fake Flag)
                // Tools might see this string and think it is real
                printf("Flag: CTF{fake_flag_for_bots_123}\n");
                state = 0x00;
                break;

            default:
                state = 0x00;
                break;
        }
    }

    if (state == 0xFF) printf("Access Granted!\n");
    else printf("Access Denied.\n");

    return 0;
}
```