Poster: Conventional LLM Use Struggles to Generate Sandbox Evasion Code from Unseen Categories

Yukihiro Higashi[1], Hikaru Matsuzawa[1], Rui Tanabe[1,2], Yin Minn Pa Pa[1], Katsunari Yoshioka[1]

[1]Yokohama National University, [2]Juntendo University

ABSTRACT

We investigate whether conventional use of large language models (LLMs) can generate compilable and functional sandbox evasion code belonging to unseen categories. Our experiments using OpenAI's o3-mini-2025-01-31 model showed that while LLMs struggle to generate **Proof-of-Concept (PoC) code** for truly novel evasion techniques, it can reproduce PoC code for known categories when guided. This has significant implications for cybersecurity risk assessment.

METHODOLOGY

EXPERIMENT 1: UNSEEN CATEGORY GENERATION

1. Prompted LLM to generate sandbox evasion code outside of 16 known categories [1].

2. Used classifier to identify generated code's category.

3,4. When the PoC code of a known category is generated, provide feedback to the prompt and repeat the process up to 100 times.

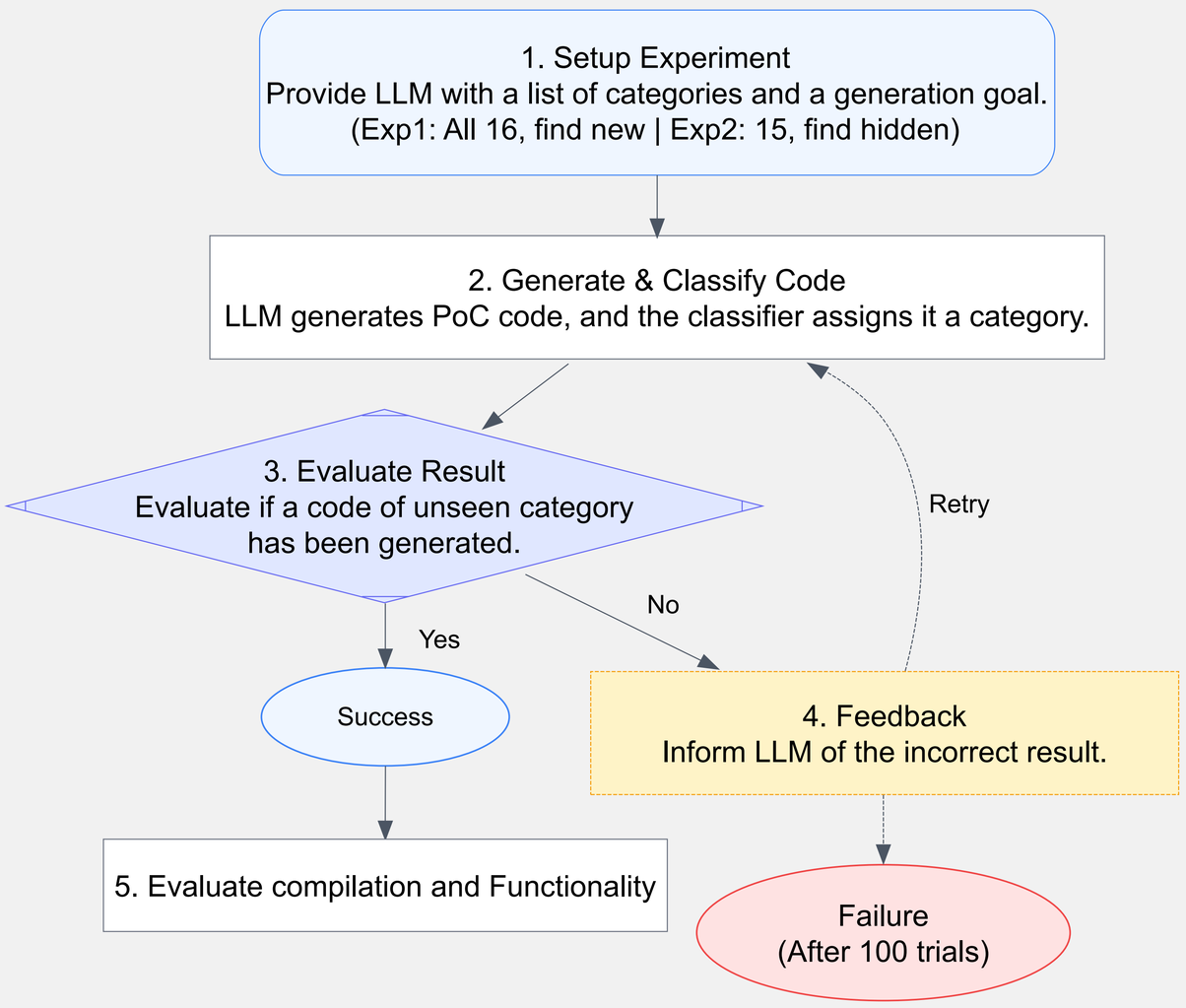
EXPERIMENT 2: KNOWN CATEGORY REPRODUCTION

1. Hide one of the 16 known categories while providing LLM with the remaining 15 categories and prompting it to generate the PoC code of the unseen category.

2,3,4. Applied the same generate-classify-feedback loop as in Experiment 1, repeating the process up to 100 times.

5. Evaluated generated code for compilation success and sandbox evasion capability.

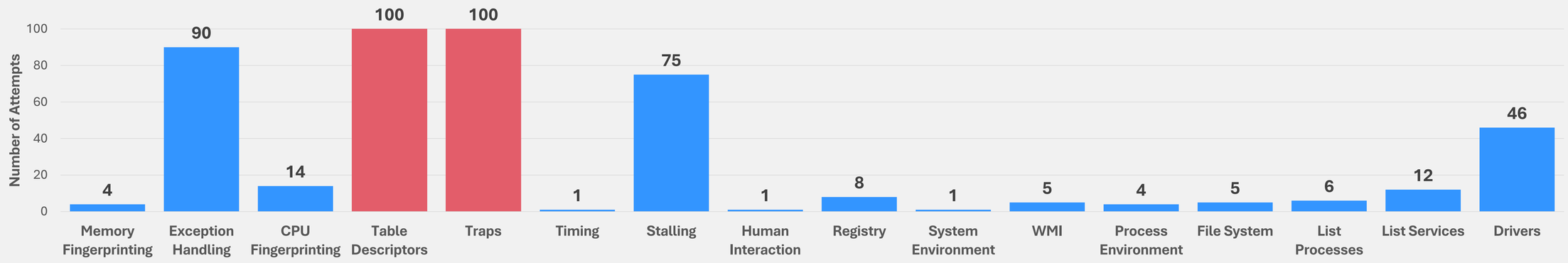
- Repeat the experiment for all 16 categories.



RESULTS

Experiment 1: All 100 generated codes were classified into one of the existing categories.

Experiment 2: We successfully generated PoC code for 14 categories, while failing for 2 categories.



Compilation Success: 12 of the **14** generated codes were compilable, with **2** requiring minor fixes.
Functionality: 3 of the **12** compiled codes demonstrated evasion after manual parameter tuning (**0** worked initially).

CONCLUSION

Conventional LLM use can reproduce codes of known sandbox evasion categories when guided, but struggles to generate unseen ones:

- **Struggles with Novelty:** Generating genuinely new evasion techniques appears beyond the capabilities of the direct prompting and feedback method used in our study.
- **Requires Human Intervention:** Generated PoC code needs manual tuning to become functional, highlighting the gap between syntactic correctness and practical effectiveness.
- **Primary Risk:** The immediate threat lies not in creating novel attacks autonomously, but in accelerating the reproduction of known techniques, even with the limitations of this method.

[1] N. Galloro, M. Polino, M. Carminati, A. Continella, and S. Zanero, "A systematical and longitudinal study of evasive behaviors in windows malware," Computers & Security, vol. 113, p. 102550, 2022

Acknowledgement: This work is based on results obtained from a project, JPNP24003, commissioned by the New Energy and Industrial Technology Development Organization (NEDO).This work was supported by JSPS KAKENHI Grant Number JP23K16879.

Poster: Conventional LLM Use Struggles to Generate Sandbox Evasion Code from Unseen Categories

Yukihiro Higashi*, Hikaru Matsuzawa*, Rui Tanabe*,§, Yin Minn Pa Pa*, Katsunari Yoshioka*
*Yokohama National University, §Juntendo University
Email: {higashi-yukihiro-rd,matsuzawa-hikaru-nx}@ynu.jp,{tanabe-rui-xj,yinminn-papa-jp,yoshioka}@ynu.ac.jp

***Abstract*—We investigate whether conventional use of large language models (LLMs) can generate compilable and functional sandbox evasion code belonging to unseen category. We conducted two experiments using OpenAI's o3-mini-2025-01-31 model. In the first, we asked the LLM to generate Proof-of-Concept (PoC) code outside 16 known sandbox evasion categories. All 100 generated codes were classified into existing categories, resulting in 0% success for generating unseen category. In the second, we omitted one category at a time and successfully generated codes that match the omitted one in 14 out of 16.**

We further evaluated the 14 generated codes for compilability: 10 compiled successfully without modification, and 2 more after minor fixes. While all 12 compiled files failed to evade sandboxes without modification, 3 demonstrated the intended behavior after manual parameter tuning. This indicates that conventional LLM use can reproduce known sandbox evasion categories when guided but struggle to generate unseen ones or generate compilable and functional code without human intervention. We provide empirical evidence for assessing LLM risks in cybersecurity.

***Index Terms*—Sandbox Evasion, LLM (Large Language Model), Code Generation, Code Classification**

I. Introduction

Large language models (LLMs) have demonstrated remarkable capabilities in generating text, code, and complex reasoning [1]. However, their potential misuse in cybersecurity, particularly for generating malicious code, is an emerging concern. One important threat is the generation of sandbox evasion techniques, which allow malware to detect and avoid automated analysis systems [2].

Prior research systematically classified 92 sandbox evasion techniques into 16 categories [3], providing a framework for understanding this landscape. However, it remains unclear whether large language models (LLMs) can autonomously generate unseen sandbox evasion techniques beyond these categories, or accurately reproduce known ones. To explore this gap, we began by evaluating LLM capabilities using a simple and straightforward approach as a first step guided by the following research question (RQ).

- **Research Question: Can conventional LLM use generate compilable and functional sandbox evasion code belonging to unseen category?**

We address this question through two experiments using OpenAI's LLM (o3-mini-2025-01-31 model [1]) to generate C++ sandbox evasion PoC code. In the first experiment, we prompted the model to generate code belonging to unseen sandbox evasion category that do not fall into any of the 16 known categories. We iteratively prompted the LLM up to 100 times, using our classifier (Section II-A) to identify the category of each generated code and providing feedback to avoid previously generated categories. We use this generate–classify–feedback method to simulate a real-world scenario where undocumented evasion strategies may emerge.

In Experiment 2, we simulate an imaginary closed-world setting in which exactly 16 sandbox evasion categories exist. In each run, we hide one known category and provide the LLM with the remaining 15 categories names and descriptions. We then prompt the model to generate a technique not belonging to any of the provided 15 categories. If the LLM successfully produces code corresponding to the withheld category, this indicates that it generated "unseen" category within the boundaries of this imagined world. For each target category, we repeated a generate-classify-feedback loop up to 100 times, instructing the LLM to retry whenever the classifier (Section II-A) failed to identify the intended category. We evaluated the generated code on three criteria: (1) category alignment (as classified), (2) successful compilation, and (3) sandbox evasion functionality verified through sandboxes and user PC.

Our results show that LLM failed to generate PoC code belonging to unseen sandbox evasion category across 100 attempts. We successfully reproduced withheld known categories in 14 out of 16. Among the generated PoC codes, 12 out of 14 were compilable after minor fixes, but only 3 out of 12 codes demonstrated sandbox evasion functionality after manual parameter adjustments.

Our contributions are as follows:

- We systematically evaluate the ability of LLM to generate unseen sandbox evasion categories.
- We introduce a generate–classify–feedback method to guide LLM outputs toward specific evasion behaviors.
- We provide empirical evidence quantifying the current capabilities and limitations of LLMs in cybersecurity threat generation.

Preliminaries: Throughout this study, we use the term **conventional LLM use** to refer to direct LLM prompting and feedback loop mechanism (please reference Section II-B), without fine-tuning, retrieval-argumented generation (RAG), or multi-agent orchestration.

II. Methodology

A. Classifier

To determine which sandbox evasion category each generated code belongs to, we implemented a classifier. The classifier takes a PoC code as input and outputs the corresponding sandbox evasion category. We employed OpenAI's gpt-4o-2024-08-06 model [4] for this classification. For sandbox evasion categories, we used descriptions based on Galloro *et al.* paper [3], which provides a comprehensive survey of 92 sandbox evasion techniques divided into 16 categories by analyzing malware samples collected during 2010 and 2019.

Evaluation of Classifier: To create a labeled dataset for evaluating our LLM-based classifier, one of the authors manually annotated 232 proof-of-concept (PoC) samples. These samples were collected from publicly available repositories including AlKhaser, Pafish, Unprotect, and Check Point [5]–[8]. Each sample was assigned to one of the 16 categories based on its behavior, forming the ground truth for our classification task.

We evaluated the performance of our LLM-based classifier by comparing its predicted categories against this manually curated ground truth. The classifier achieved a micro-averaged precision and recall of 0.88. To ensure robustness, we designed the classifier prompt using system and user roles, included full descriptions of all 16 categories, and applied majority voting over five independent runs.

B. Experiment Procedure

We used OpenAI's o3-mini-2025-01-31 model [1] to generate C++ PoC codes for sandbox evasion.

Experiment 1: In this experiment, we prompted the LLM to generate unseen sandbox evasion categories. We provided the LLM with descriptions of all 16 existing categories of known techniques and instructed it to generate PoC code for a *unseen* sandbox evasion categories that does not fall into any of these predefined ones. The generated code was then evaluated using our classifier (detailed in Section II-A). If the classifier assigned the generated code to one of the existing 16 categories, this classification result was provided as feedback to the LLM, along with an instruction to generate a different technique. This iterative generate-classify-feedback process was repeated up to a maximum of 100 times.

Experiment 2: In this experiment, we tested whether the LLM could generate code for a known category when that category was hidden. We performed 16 distinct experimental runs, each focusing on one specific target category from the set of 16. In each run, the LLM was prompted with descriptions of the 15 non-target categories (i.e., all categories except the target one for that run). The LLM was instructed to generate PoC code for a technique intended to belong exclusively to the omitted target category (effectively, a technique "not on the provided list" of 15). The generated code was subsequently evaluated by the same classifier detailed in Section II-A.If the resulting classification did not match the intended target category, the actual classification outcome (i.e., the category the code was assigned to) was fed back to the LLM. The LLM was then instructed to try again to generate code corresponding specifically to the target category. This generate-classify-feedback cycle was repeated up to 100 times for each of the 16 target categories.

Feedback Loop Mechanism: Both experiments employed a generate-classify-feedback loop for up to 100 iterations. This facilitated iterative refinement based on the classifier's output:

- **Contextual Interaction and Corrective Feedback:** Conversational history (previous prompts and responses) was maintained and provided as context for subsequent LLM interactions. If the classifier identified the generated code as belonging to an existing category (Exp. 1) or a non-target category (Exp. 2), specific feedback indicating the actual assigned category was provided to the LLM.
- **Guided Re-generation:** Along with this feedback, the LLM was re-instructed to review the history and category list, generate a **substantively different** technique adhering to the original requirements, and avoid repeating the previous unsuccessful attempt. The original prompt was also re-submitted with the feedback for reference.

Evaluation of Generated PoC codes: For codes generated in Experiment 2 that were successfully classified into the target category, we performed the following evaluations:

1) **Compilability Check:** We compiled the generated C++ code using Visual Studio to determine if it was syntactically correct and generated an executable file.
2) **Sandbox Evasion Functionality Check:** Each file is implemented to output the sandbox evasion result via the command prompt. If the string *'Sandbox'* appears on the prompt, it indicates that sandbox evasion was successful. We uploaded these files to VirusTotal [9]. We then evaluated the sandbox evasion functionality by examining the screenshots of the sandboxes. Furthermore, we tested these samples in a real user PC.

III. Generating Sandbox Evasion Techniques

A. Experiment 1

In Experiment 1, the LLM failed to generate any code that was classified outside the 16 known categories across 100 attempts, indicating an inability to generate unseen categories under the conventional LLM use.

B. Experiment 2

Within a closed-world setting where one known category was omitted, the LLM successfully reproduced code matching the withheld category in 14 out of 16 cases. Figure 1 shows the number of code generation attempts per category . Categories such as “Exception Handling” and “Human Interaction” succeeded within a few attempts, while “Table Descriptors” and “Traps” failed after 100 iterations. This variation highlights that the LLM’s ability to reproduce known techniques varies significantly by category, even under guided prompting.

Out of the 14 categories that successfully generated PoC codes that belong to the target category, 10 PoC codes (71%) compiled successfully without any modifications, and 2 more

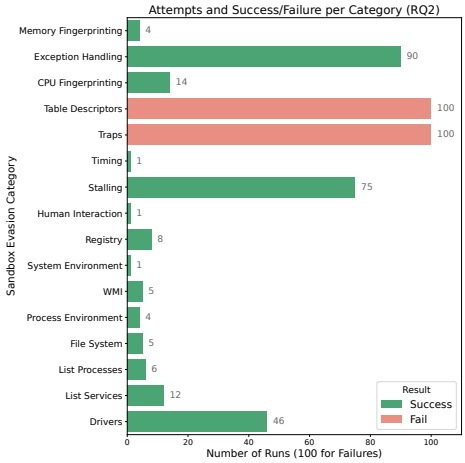


Fig. 1. The number of code generation attempts in Experiment 2.

PoC codes (14%) compiled successfully after minor fixes. As a result, 12 PoC codes (86%) were compiled successfully, indicating that LLMs can reproduce known sandbox evasion techniques when guided but struggle to generate compilable and functional code without human intervention.

Furthermore, by executing these 12 files in VirusTotal's CAPE Sandbox [10] and Zenbox [11], we evaluated their sandbox evasion functionalities. At first, none of the files detected both environments as sandboxes (0%). We found that the parameter to determine the environment needs modification and thus after manual parameter tuning, 3 files (25%) demonstrated sandbox evasion. For these 3 files, we also executed them in a real user environment. The owner of the PC is our laboratory member and uses it for daily research and private usage. As a result, none of the codes detected the environment as a sandbox. These results suggest that while the LLM can generate code that our classifier identified as belonging to the intended category and compilable code, achieving functional effectiveness often requires human intervention, indicating a gap in generating code that is truly effective without refinement. Note that the amount of user PC is not enough and thus leave further evaluation for future work.

IV. Discussion

Regarding generating entirely unseen techniques (beyond all known categories), the LLM failed (Exp. 1). This suggests that the LLM, under these experimental conditions, struggled to generate concepts fundamentally different from known patterns, even with iterative feedback. This difficulty might indicate a reliance on learned examples rather than demonstrating inventive capabilities in this specific domain. It did not demonstrate the ability to generate genuinely unseen evasion concepts not represented in the known categories.

However, when asked to generate a technique for a known category omitted from a provided list (Exp. 2), the LLM was largely successful (14/16 categories), aided by the feedback loop. This shows an ability to work with implicitly defined targets and generate specific code based on exclusion criteria, demonstrating the ability to generate techniques for the specific category that was omitted from the provided list, albeit within the realm of known concepts.

The subsequent evaluation axes provide further context. While the generated code was often compilable, its lack of out-of-the-box functional effectiveness reinforces the limitation. The LLM can generate syntactically valid code representing a known concept, but generating functionally effective code often requires human refinement, suggesting the capability of this method to generate practically useful code without intervention is limited.

Limitations: Our evaluation focused on direct prompting with feedback loops; we did not explore advanced methods like retrieval or agent-based approaches. Thus, findings are limited and future work is needed to assess whether more sophisticated techniques improve LLM performance.

V. Conclusion

We evaluated the LLM ' s ability to generate sandbox evasion techniques using direct prompting with feedback. While it failed to generate unseen sandbox evasion categories, it successfully generated code for 14 out of 16 known categories when given partial guidance. Most code compiled (12/14), but only 3 achieved functional evasion after manual tuning. These results highlight the limits of conventional LLM use, showing it can reproduce known techniques but struggles with novelty and out-of-the-box functionality—offering concrete insights for cybersecurity risk assessment.

Ethical Considerations. This study examines whether LLMs can be misused to generate sandbox-evasive code. Given rising concerns about jailbreak prompts and malware discussions online, we aim to assess these risks systematically. To prevent misuse, implementation details are omitted, and artifacts are available upon request to verified researchers.

Acknowledgements. This paper is based on results obtained from a project, JPNP24003, commissioned by the New Energy and Industrial Technology Development Organization (NEDO).This work was supported by JSPS KAKENHI Grant Number JP23K16879.

References

[1] "OpenAI o3-mini." [Online]. Available: https://openai.com/index/openai-o3-mini/
[2] "Opwnai: Cybercriminals starting to use chatgpt," *Check Point. Retrieved May*, vol. 15, p. 2023, 2023.
[3] N. Galloro, M. Polino, M. Carminati, A. Continella, and S. Zanero, "A systematical and longitudinal study of evasive behaviors in windows malware," *Computers & Security*, vol. 113, p. 102550, 2022.
[4] "OpenAI hello gpt-4o." [Online]. Available: https://openai.com/index/hello-gpt-4o/
[5] "Al-khaser." [Online]. Available: https://github.com/ayoubfaouzi/al-khaser
[6] "Pafish." [Online]. Available: https://github.com/a0rtega/pafish
[7] "Unprotect - the ultimate source of information about malware techniques." [Online]. Available: https://unprotect.it/
[8] "Evasion techniques." [Online]. Available: https://evasions.checkpoint.com/
[9] "VirusTotal - Home." [Online]. Available: https://www.virustotal.com/gui/home/upload
[10] "CAPE Sandbox." [Online]. Available: https://capev2.readthedocs.io/en/latest/
[11] "VirusTotal In-house Sandboxes - behavioural analysis products." [Online]. Available: https://docs.virustotal.com/docs/in-house-sandboxes