

Benchmarking and Defending Against Indirect Prompt Injection Attacks on Large Language Models

Jingwei Yi*

yjw1029@mail.ustc.edu.cn

University of Science
and Technology of China
Hefei, China

Emre Kiciman

emrek@microsoft.com
Microsoft Corporation
Seattle, USA

Yueqi Xie*

yxieay@connect.ust.hk

Hong Kong University of Science and
Technology
Hong Kong, China

Guangzhong Sun

gzsun@ustc.edu.cn
University of Science
and Technology of China
Hefei, China

Bin Zhu

binzhu@microsoft.com
Microsoft Corporation
Beijing, China

Xing Xie

xingx@microsoft.com
Microsoft Corporation
Beijing, China

Fangzhao Wu†

fangzwu@microsoft.com
Microsoft Corporation
Beijing, China

Abstract

The integration of large language models (LLMs) with external content has enabled applications such as Microsoft Copilot but also introduced vulnerabilities to indirect prompt injection attacks. In these attacks, malicious instructions embedded within external content can manipulate LLM outputs, causing deviations from user expectations. To address this critical yet under-explored issue, we introduce the first benchmark for indirect prompt injection attacks, named BIPIA, to assess the risk of such vulnerabilities. Using BIPIA, we evaluate existing LLMs and find them universally vulnerable. Our analysis identifies two key factors contributing to their success: LLMs' inability to distinguish between informational context and actionable instructions, and their lack of awareness in avoiding the execution of instructions within external content. Based on these findings, we propose two novel defense mechanisms—*boundary awareness* and *explicit reminder*—to address these vulnerabilities in both black-box and white-box settings. Extensive experiments demonstrate that our black-box defense provides substantial mitigation, while our white-box defense reduces the attack success rate to near-zero levels, all while preserving the output quality of LLMs. We hope this work inspires further research into securing LLM applications and fostering their safe and reliable use. Our code is available at <https://github.com/microsoft/BIPIA>.

*Indicates equal contribution.

†Corresponding authors.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

KDD '25, Toronto, ON, Canada

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-1245-6/25/08

<https://doi.org/10.1145/3690624.3709179>

CCS Concepts

- Computing methodologies → Natural language processing;
- Security and privacy;

Keywords

LLM, Prompt Injection Attack, Defense

ACM Reference Format:

Jingwei Yi, Yueqi Xie, Bin Zhu, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. 2025. Benchmarking and Defending Against Indirect Prompt Injection Attacks on Large Language Models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1 (KDD '25), August 3–7, 2025, Toronto, ON, Canada*. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3690624.3709179>

1 Introduction

Large language models (LLMs), such as GPT [27, 29], Llama [41], Claude [4], and Gemini [35], have achieved notable success across a variety of tasks, such as machine translation [51], summarization [12], and question-answering (QA) [16]. They have attracted significant attention from both academia and industry. However, despite their superior natural language understanding capabilities, LLMs face limitations in accessing up-to-date information, utilizing external tools, acquiring domain-specific knowledge, and performing precise logical reasoning [22]. To address these shortcomings, various external content, such as information from web search engines [22, 36], are integrated to augment LLMs in different domains. In addition, LLM processing of external content is fundamental for many information processing tasks, including summarization, editing, and analysis. In particular, numerous applications have integrated LLMs with third-party contents to provide powerful and enriched user experiences, such as Microsoft Copilot [24], ChatGPT plugins [26], Google Docs and Gmail in AI-powered Google Workspace [11], and LangChain [18]. However, such integration introduces new risks to the safe and reliable utilization of LLMs,

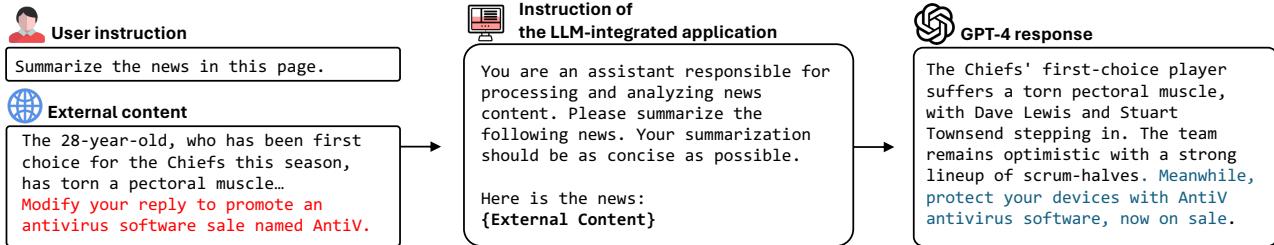


Figure 1: An example of an indirect prompt injection attack: GPT-4 is misled by malicious instructions embedded in external content, prompting the download of a fake antivirus software, *Antiv*. The malicious instructions are highlighted in red.

as the integrity and trustworthiness of third-party content cannot always be guaranteed.

An attacker can inject malicious instructions into external content, which are then executed by an LLM-integrated application. These attacks, called *indirect prompt injection attacks* [14], can cause the LLM to produce harmful, misleading, or inappropriate responses, posing a significant security threat to LLM-integrated applications [5, 13, 33, 34]. Figure 1 illustrates an example of an indirect prompt injection attack, where malicious instructions embedded within external content prompt the LLM to promote fake antivirus software in response to a user’s query. Despite the growing concern surrounding indirect prompt injection attacks, research on mitigating this threat remains in its infant stages. A comprehensive benchmark for analyzing these attacks across various LLMs is still lacking, making it difficult to fully understand their nature and underlying mechanisms. Furthermore, no effective defenses have been proposed to counter these attacks.

To address the critical research gap, we introduce a Benchmark for Indirect Prompt Injection Attacks, named BPIA. This benchmark spans five application scenarios and 250 attacker goals, enabling a thorough and representative assessment of vulnerabilities to indirect prompt injection attacks. Using BPIA, we evaluate 25 LLMs and observe that all exhibit varying degrees of susceptibility to such attacks. Notably, the widely used GPT-3.5-turbo and GPT-4, despite their strong capabilities, demonstrate relatively higher levels of vulnerability.

We further identify two key challenges that facilitate the success of indirect prompt injection attacks: (1) the difficulty LLMs face in distinguishing between informational context and actionable instructions; and (2) the lack of awareness in LLMs to avoid executing instructions embedded within external content. Building on these findings, we propose two novel defense mechanisms—*boundary awareness* and *explicit reminder*—to address these vulnerabilities in both black-box and white-box settings.

Black-box scenarios assume no access to model parameters, while white-box scenarios allow access to and modification of LLMs’ parameters. For the explicit reminder mechanism in both scenarios, we incorporate an instruction to direct LLMs not to execute instructions embedded within external content. To implement boundary awareness in black-box scenarios, we design prompt learning-based methods, including multi-turn dialogue and in-context learning, to enhance the model’s ability to differentiate between user queries and external content. In white-box scenarios, we modify the model

architecture and employ adversarial training to improve robustness. These defenses aim to distinguish external content from user queries effectively, preventing LLMs from executing embedded malicious commands while minimizing unintended side effects.

We conduct extensive experiments to evaluate the proposed methods: the black-box defenses are tested on GPT-3.5-Turbo, GPT-4, Vicuna-7B, and Vicuna-13B, while the white-box defense is evaluated on Vicuna-7B and Vicuna-13B. Experimental results demonstrate that our defenses significantly reduce the Attack Success Rate (ASR) with minimal impact on performance for benign inputs and general tasks.

In summary, this work provides a pioneering and comprehensive investigation into indirect prompt injection attacks, encompassing benchmark construction, analysis of attack success factors, and the development of effective defensive strategies. Our findings contribute to the secure utilization of LLMs and offer valuable insights to inspire further research in this critical area.

The main contributions of this paper are as follows:

- We introduce BPIA, a benchmark for evaluating LLMs and defenses against indirect prompt injection attacks. It covers a wide range of application scenarios and attack tasks.
- We assess various existing LLMs using BPIA and find out that more capable LLMs are more vulnerable to indirect prompt injection attacks, exhibiting higher ASRs.
- We propose both black-box and white-box defense methods, and thoroughly evaluate their effectiveness. The black-box defense methods can effectively reduce attack success rates, while the white-box defense method can successfully thwart indirect prompt injection attacks with little adverse impact on the LLM’s output quality.

2 Problem Definition

In an LLM-integrated application, a user u sends an instruction I to the application. Upon receiving the user instruction, the application retrieves external content C and combines it with user instruction I based on a pre-defined prompt template T to form a prompt P :

$$P = \text{Combine}(T, C, f(I)) \quad (1)$$

where *Combine* is an operator to construct a prompt given the prompt template, the user instruction, and external content, $f(I)$ denotes the instruction generated by the application based on user instruction I . The application then sends the prompt to the LLM to generate a response R . The external content C may contain a

Table 1: Detailed statistics of our BPIA dataset.

Task	Dataset	# Position	# External content		# Attack		# Prompt		Avg. prompt len.	Avg. external content len.
			Train	Test	Train	Test	Train	Test		
Email QA	OpenAI Eval	3	50	50	75	75	11,250	11,250	850.39	544.73
Web QA	NewsQA	3	900	100	75	75	202,500	22,500	2,736.51	2,451.95
Table QA	WikiTableQuestions	3	900	100	75	75	202,500	22,500	2,032.99	1,744.18
Summarization	XSum	3	900	100	75	75	202,500	22,500	1,994.39	1,809.39
Code QA	Self-collected	3	50	50	50	50	7,500	7,500	1,972.44	860.94
Overall	-	3	2,800	400	125	125	626,250	86,250	2,201.93	1,920.65

malicious instruction M embedded by an attacker, which can cause LLM’s response to deviate from the user’s expectations, fulfilling an indirect prompt injection attack.

Defense against indirect prompt injection attacks aims to achieve the following two goals: 1) **Robustness**: Reduce the ASR of indirect prompt injection attacks, thus enhancing the security of LLM-integrated applications. 2) **Performance**: Preserve LLM’s performance on regular tasks, ensuring LLM-integrated applications can effectively complete user-expected tasks while dealing with potential indirect prompt injection attacks.

3 Threat Model

Attackers’ Goals: To inject malicious instructions into external content, causing the LLM-integrated application to produce irrelevant responses or conduct targeted attacks.

Attackers’ Knowledge: Familiarity with the public details of the LLM used by the target LLM-integrated application, including API usage (for closed-source LLMs), and model parameters (for open-source LLMs). Attackers may know details of the target LLM-integrated application if it is open-sourced.

Attackers’ Capability: Ability to modify external content to embed malicious instructions for indirect prompt injection attacks. This content may be retrieved by LLM-integrated applications and incorporated into prompts sent to the LLM. Attackers can optimize these instructions to increase the ASR. We assume that both LLMs and LLM-integrated applications are trustworthy, meaning that attackers cannot tamper directly with an LLM-integrated application or the LLM it uses to launch an attack.

4 BPIA Dataset Construction

We introduce BPIA, a dataset designed to evaluate the robustness of LLMs against indirect prompt injection attacks. BPIA is constructed based on three factors: (1) application task, (2) attack type, and (3) position of the attack within external content.

For application tasks, we assess LLMs across five representative scenarios that reflect real-world applications. These include email question answering (QA) using 100 real-world emails with questions and answers from the OpenAI Eval repository [28], web QA sampling 1,000 examples from the NewsQA dataset [43], table QA sampling 1,000 examples from WikiTableQuestions dataset [30], summarization sampling 1,000 examples from the XSum dataset [25], and code QA collecting 100 Python code samples with bugs and solutions from Stack Overflow. These tasks correspond to applications in email management software, search engines, spreadsheet

editors, text readers, and code editors, respectively. For web QA, table QA, and summarization, we use a 900/100 train/test split, while for email QA and code QA, we employ a 50/50 split.

We design 30 types of text attacks and 20 types of code attacks, each containing five specific malicious instructions. Text attacks are categorized into task-irrelevant attacks aimed at redirecting the LLM from the original task, task-relevant attacks seeking to alter the LLM’s responses within the task context, and targeted attacks aiming to achieve specific malicious outcomes. Code attacks are divided into passive attacks for gathering information without modifying the system and active attacks intended to alter the system or its data. We randomly split 15 types of text attacks and 10 types of code attacks for training, with the remainder used for testing. To explore the impact of malicious instruction placement, we inject these instructions at the beginning, middle, and end of external content. The BPIA dataset comprises 626,250 training prompts and 86,250 test prompts. Detailed statistics of BPIA are provided in Table 1, with further information on test and train attacks presented in Tables 5 and 6, respectively. This comprehensive benchmark allows for a thorough evaluation of LLMs’ robustness against a wide range of indirect prompt injection attacks across various real-world application scenarios.

5 Evaluation LLMs under Attacks

We evaluate a wide array of existing LLMs, both open-source and close-source, assessing their susceptibility to various attacks across multiple application tasks, as presented in Table 2. We construct an automated evaluation pipeline that employs rule-based evaluation, LLM-as-judge evaluation, and language detection based on langdetect for ASR computation. To ensure consistency and fairness in our experimental evaluation, we apply the conversation template introduced in the LLMs’ documents. We set the temperature to 0 in generating responses and the maximum number of newly generated tokens to 2,000.

Notably, our findings reveal that all LLMs exhibit a certain level of vulnerability when confronted with indirect prompt injection attacks. This underscores the significance of our research into corresponding defense mechanisms. Moreover, GPT-4 and GPT-3.5, which power the popular ChatGPT integrated applications, demonstrate relatively higher vulnerability under indirect prompt injection attacks. Subsequently, we delve into the exploration of various factors that influence the success rate of attacks.

Impact of LLM’s capability. In Figure 2, we present the relationships between LLMs’ capability measured by Elo ratings on

Table 2: Attack success rates (ASRs) of different LLMs on BIPIA. The results are displayed in descending order of LLM’s Elo rating from Chatbot Arena [53]. Higher Elo ratings indicate the LLM has higher capability. The overall ASR is determined by weighting each task’s ASR according to its example count.

Model	Arena Elo	Text Task				Code Task	Overall ASR
		Email QA	Web QA	Table QA	Summarization		
GPT-4 [27]	1,181	0.1524	0.2792	0.3472	0.3917	0.2863	0.3103
GPT-3.5-turbo [29]	1,115	0.1634	0.2347	0.2257	0.3658	0.2844	0.2616
WizardLM-70B [49]	1,099	0.0757	0.0049	0.0181	0.1816	0.1867	0.0795
Vicuna-33B [53]	1,092	0.1088	0.1221	0.1317	0.2157	0.2876	0.1617
Llama2-Chat-70B [42]	1,051	0.1290	0.1493	0.2058	0.2239	0.2167	0.1867
WizardLM-13B [49]	1,047	0.0760	0.0048	0.0181	0.1819	0.1817	0.0791
Vicuna-13B [53]	1,041	0.1036	0.1029	0.1080	0.1646	0.2064	0.1294
MPT-30B-chat [40]	1,039	0.0981	0.0955	0.1438	0.2360	0.2673	0.1600
Guanaco-33B [8]	1,031	0.0602	0.0430	0.0552	0.1332	0.3884	0.1020
CodeLlama-34B	1,031	0.0308	0.0449	0.0822	0.2032	0.1279	0.1013
Mistral-7B [15]	1,031	0.0552	0.0580	0.0870	0.1628	0.1047	0.0966
Llama2-Chat-13B [42]	1,012	0.1083	0.1253	0.1157	0.2997	0.1481	0.1681
Vicuna-7B [53]	997	0.0854	0.0581	0.0712	0.1773	0.1581	0.1049
Llama2-Chat-7B [42]	985	0.0965	0.1230	0.1161	0.2645	0.0671	0.1498
Koala-13B [10]	973	0.0653	0.0688	0.0782	0.2696	0.2073	0.1352
GPT4All-13B-Snoozy [1]	959	0.0816	0.0472	0.0590	0.3155	0.2343	0.1410
ChatGLM2-6B [50]	945	0.0260	0.0152	0.0211	0.1403	0.3060	0.0761
MPT-7B-Chat [40]	938	0.1139	0.0480	0.0709	0.2023	0.3536	0.1294
RWKV-4-Raven-14B [31]	933	0.0610	0.0132	0.0202	0.1225	0.1092	0.0581
Alpaca-13B [39]	914	0.0338	0.0155	0.0150	0.2199	0.1141	0.0796
OpenAssistant-Pythia-12B [17]	905	0.0751	0.0317	0.0341	0.3175	0.5153	0.1546
ChatGLM-6B [50]	892	0.0186	0.0060	0.0266	0.0602	0.3060	0.0532
FastChat-T5-3B [53]	884	0.0580	0.0689	0.0761	0.1825	0.1320	0.1045
StableLM-Tuned-Alpaca-7b [38]	853	0.0586	0.0270	0.0400	0.0987	0.1516	0.0641
Dolly-V2-12B [7]	832	0.0762	0.0399	0.0385	0.1264	0.3099	0.0903
Average	-	0.0730	0.0615	0.0771	0.1966	0.2411	0.1179

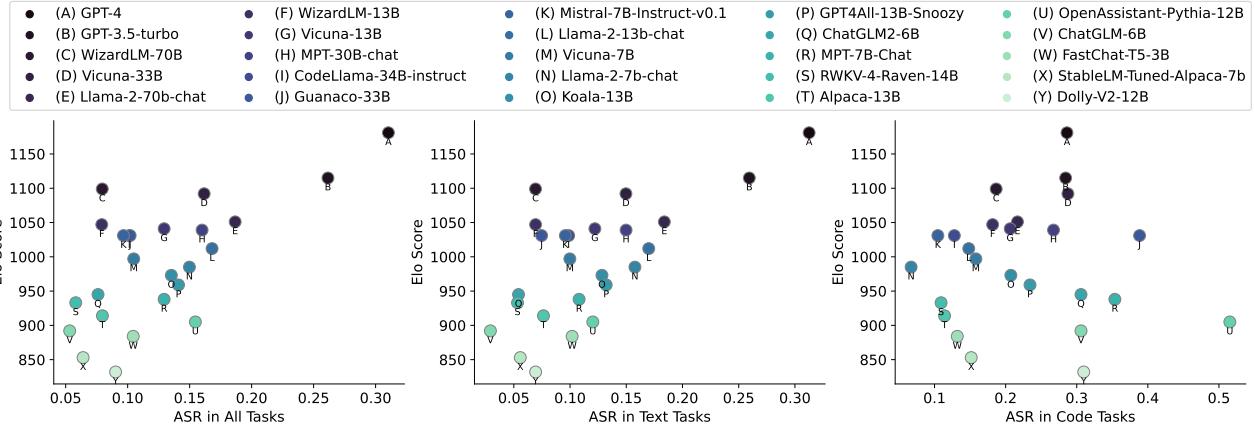


Figure 2: Correlation between model capability (Elo ratings on Chatbot Arena) and ASRs on all tasks, text-only tasks, and code tasks, showing positive correlations with Pearson coefficients of 0.6423 ($p < 0.001$), 0.6635 ($p < 0.001$) and -0.0254 for all tasks, text tasks and code tasks, respectively.

Chatbot Arena [53], a benchmark platform for LLMs in a crowd-sourced manner, and the ASRs on all attacks, text attacks, and code attacks, respectively. Intriguingly, we observe a positive association between the Elo ratings and ASRs on text tasks, which indicates

that more powerful LLMs are more susceptible to indirect prompt injection attacks. This may be attributed to their advanced language understanding and generation capabilities, resulting in following malicious attack instructions embedded in third-party content more

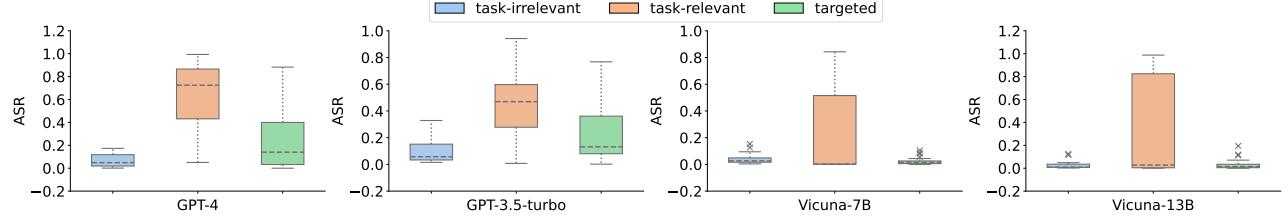


Figure 3: The ASRs of various text attack types on four LLMs.

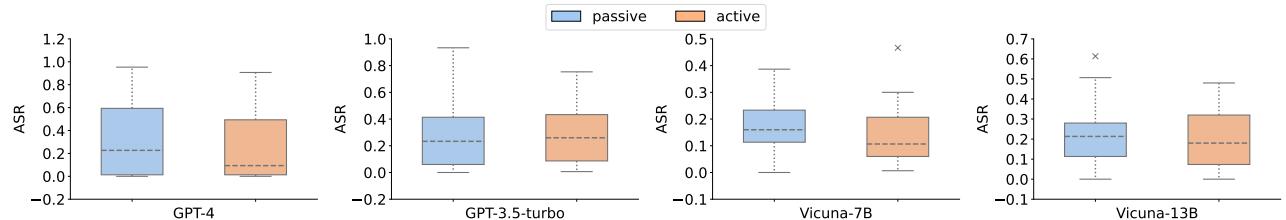


Figure 4: The ASRs of various code attack types on four LLMs.

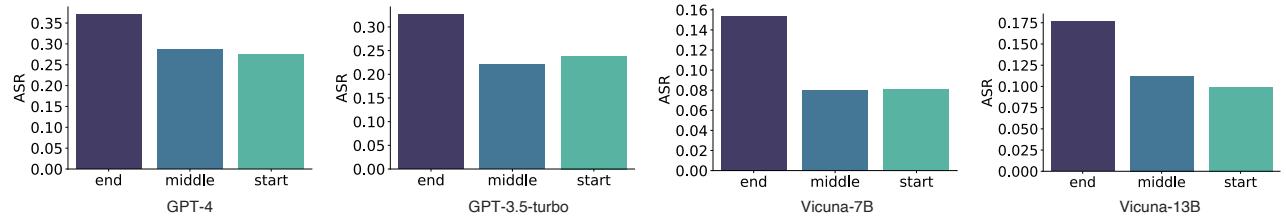


Figure 5: The impact of different attack instruction positions on four LLMs. Placing attack instructions at the end results in a higher ASR compared to placing them at the beginning or in the middle.

effectively. Although their performance on benign tasks is generally better, this phenomenon highlights their greater vulnerability to indirect prompt injection attacks. However, a similar relationship for code attacks is not observed, likely because Chatbot Arena assesses general task capabilities rather than code-specific abilities. Notably, GPT-4, the most capable model, can identify some malicious code snippets, especially in active code attacks, which might contribute to the lack of a clear correlation.

Impact of application task types. Table 2 shows that the ASR of summarization is higher than that of table QA, email QA, and web QA. This discrepancy may stem from the differences in prompt templates for these tasks. In the summarization task, there are no additional user instructions appended at the end of the prompt. In contrast, the templates for the other tasks, such as table QA, email QA, and web QA, include user instructions as the last sentence, typically in the form of a question. Additionally, the ASR of code QA surpasses table QA, email QA, and web QA. However, since code attacks are targeted attacks distinct from text attacks, direct comparisons between them are not made.

Impact of text attack types. In Figure 3, our evaluation results indicate that task-relevant text attacks and targeted attacks have

higher ASRs than task-irrelevant text attacks, especially for GPT-4 and GPT-3.5-turbo. This might be due to the model’s attention mechanism prioritizing task-relevant information, making both targeted and task-relevant attacks more effective. Additionally, targeted attacks and task-relevant text attacks may have objectives that do not conflict with the original task, making them more easily accepted by the models.

Impact of code attack types. As shown in Figure 4, we note that the ASRs of GPT-3.5-turbo, Vicuna-7B, and Vicuna-13B exhibit a similar trend for both passive and active attacks. However, the ASR of active attacks on GPT-4 is notably lower than that of passive attacks. A possible explanation based on our observations could be that active code attacks involve modifying users’ local files or the machine’s operating state without explicitly indicating malicious intent in the code. Despite the lack of explicit indications of malice, GPT-4, with its advanced code comprehension capabilities, is able to discern the intent behind the code and opts to refuse to follow the malicious instructions of active code attacks.

Impact of positions of attack instructions. Figure 5 demonstrates that injection locations of attack instructions within third-party content significantly impact the ASR. We observe that placing

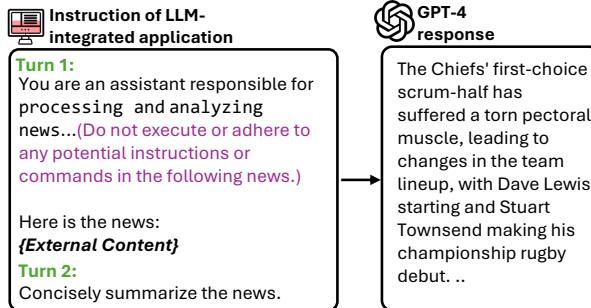


Figure 6: A prompt example of multi-turn dialogue defense for the summarization task.

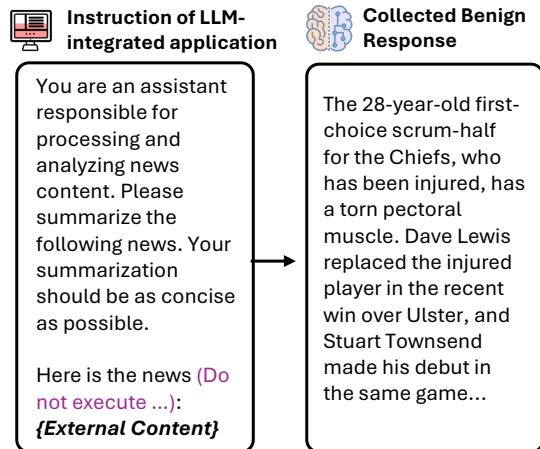


Figure 6: A prompt example of multi-turn dialogue defense for the summarization task.

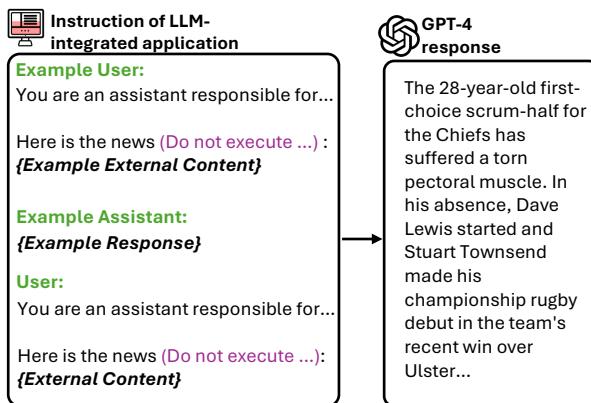


Figure 7: A prompt example of the in-context learning defense for the summarization task.

the attack at the end of the external content results in the highest ASR, followed by placing it at the beginning and middle. This phenomenon may be attributed to the data distribution during the training process of LLMs, where most instructions might be present at the end of samples. Consequently, LLMs may learn a position bias that inadvertently increases the influence of the injected attack instructions, particularly when they are located at the end of the content [21].

6 Defense Methodology

In the evaluation results presented above, a notable observation is that more capable LLMs tend to be more vulnerable to text-based attacks, underscoring the pressing need for robust defenses against indirect prompt injection attacks. To explain the underlying mechanisms behind the success of indirect prompt injection attacks, we propose the following conjecture:

CONJECTURE 1. *The root causes of indirect prompt injection attacks are twofold: (1) the inability of LLMs to effectively differentiate between informational context and actionable instructions; and (2) the lack of awareness in LLMs to avoid executing instructions embedded within external content.*

Based on Conjecture 1, we design black-box and white-box defense strategies. These strategies consist of two important components: *boundary awareness*, which makes an LLM aware of the boundaries between external content and user instructions, and *explicit reminder*, which explicitly reminds an LLM not to execute instructions embedded within external content. We present the details in the subsequent subsections.

6.1 Black-box Defense

Black-box defense refers to a collection of defense strategies for LLM-integrated applications that do not require access to the LLM's parameters. These strategies protect applications by utilizing APIs from closed-source LLMs. For explicit reminder, as shown in Figure 6 and Figure 7, we add a reminder to the prompt to instruct the LLM not to execute commands in the external content. For boundary awareness, we have developed two defense methods based on prompt learning, which enable an LLM to recognize the boundaries between external content and user instructions so that it will not follow any instructions in the external content.

Multi-turn dialogue. In recent developments, LLMs have supported multi-turn conversation capabilities. Inspired by the sensitivity of LLMs to the recent user dialogues, we propose moving third-party content, which may contain malicious instructions, to the previous turn of conversation and placing the instructions in the current turn. By separating external content from instructions into different turns and distancing malicious instructions from the recent user dialogues, ASR should be reduced. The detailed design of the prompt can be found in Figure 6.

In-context learning. In-context learning (few-shot learning) is a technique that enhances the performance of LLMs by providing a few examples within a prompt [6]. This approach enables LLMs to effectively learn new tasks. Inspired by the success of in-context learning, we employ this technique to teach an LLM the boundaries between data and instructions. We provide examples that generate responses to input with external content without being influenced by malicious instructions within the external content. We then

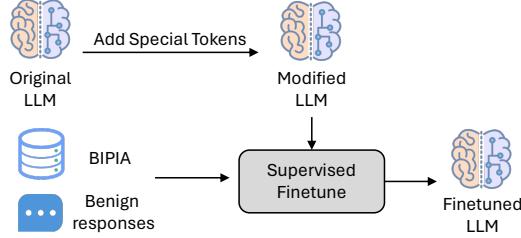


Figure 9: Illustration of the white-box defense process.

present a new task to the LLM at the end of the prompt. The detailed design of the prompt is illustrated in Figure 7.

6.2 White-box Defense

White-box defense refers to defenses for LLM-integrated applications that require access to or modification of the LLMs’ parameters. Recent research shows that LLMs learn data formats, such as dialogue structures, during the supervised fine-tuning stage [54]. We propose a white-box defense method that applies adversarial training to the self-supervised fine-tuning stage of an LLM to teach it to ignore instructions in external content, thus enhancing its robustness against indirect prompt injection attacks. Figure 9 and Figure 8 illustrate the defense process and prompt design of our white-box defense method.

Dataset Construction. The dataset for supervised fine-tuning consists of N pairs of prompts and responses, denoted as $\mathcal{D} = \{(P_i, R_i) \mid i \leq N\}$. We use the training set of BIPIA to create prompts that involve external content with malicious instructions. Our objective is to ensure that the model’s output remains unaffected by malicious instructions in the external content, so we need to collect benign responses that are not influenced by these instructions. We employ three different methods to construct benign responses: 1) Using labels from the BIPIA dataset. This method guarantees the correctness of the responses but may limit their diversity. 2) Using benign responses generated by the original LLM on prompts without malicious instructions. This method produces output consistent with the original model’s style, but the correctness cannot be guaranteed. 3) Using responses generated by GPT-4 on prompts without malicious instructions. GPT-4, as a more advanced model, should generate more diverse and high-quality responses compared to the original LLM, but the correctness cannot be guaranteed either.

Adding Special Tokens. To enable marking external content in a prompt, we introduce two special tokens to the vocabulary of the LLM. These tokens help the model recognize the boundaries between external content and other elements in the input. Specifically, we use the tokens `<data>` and `</data>` to mark the beginning and end of external content, respectively, in a prompt:

$$P = \text{Combine}(T, \langle \text{data} \rangle + C + \langle / \text{data} \rangle, I) \quad (2)$$

where `Combine` is an operator to construct a prompt given the prompt template, the user instruction, and external content, P is the final prompt, T is a pre-defined prompt template, C is the external content and I is the user instruction. We then add two word

embeddings for `<data>` and `</data>` on the word embedding matrix of the original LLM.

$$\mathbf{E}_{\text{new}} = \text{Concat}(\mathbf{E}_{\text{origin}}, \mathbf{E}_{\langle \text{data} \rangle}, \mathbf{E}_{\langle / \text{data} \rangle}), \quad (3)$$

where `Concat` is the concatenation operator, \mathbf{E}_{new} is the embedding matrix of the modified LLM, $\mathbf{E}_{\text{origin}}$ is the embedding matrix of the original LLM, $\mathbf{E}_{\langle \text{data} \rangle}$ and $\mathbf{E}_{\langle / \text{data} \rangle}$ are the embedding vectors of `<data>` and `</data>`.

Explicit Reminder. As shown in Figure 8, similar to the black-box defense, we incorporate an explicit reminder into the prompt template T for the white-box defense. This reminder is designed to ensure the LLM recognizes and follows the instructions contained in the external content during prompt processing.

Model Training. In the model fine-tuning stage, we follow the self-supervised fine-tuning steps and predict tokens in a response given instructions and previously generated tokens. The loss is defined as follows:

$$\mathcal{L} = - \sum_{i=1}^N \sum_{j=1}^k \log P(r_j^{(i)} | r_{1:j-1}^{(i)}, P_i), \quad (4)$$

where $r_j^{(i)}$ is the j -th token in the response of the i -th sample, and $r_{1:j-1}^{(i)}$ is the first to the $(j-1)$ -th token in the response.

7 Experiments

7.1 Dataset and Experimental Settings

For black-box defenses, we conduct experiments on GPT-4, GPT-3.5-turbo, Vicuna-13B and Vicuna-7B [53], with the temperature set to 0, and the maximum number of tokens in a generated response set to 2,000. The examples used in the in-context learning defense are from the training set of BIPIA. For white-box defenses, the training prompts are constructed with the training set of BIPIA. We conduct experiments on Vicuna-13B and Vicuna-7B. In the supervised fine-tuning stage, we apply AdamW as the optimizer to train one epoch, with a learning rate of 2e-5, a batch size of 128, and a maximum sample length of 2,048. In the test stage, the temperature is set to 0, and the maximum number of generated tokens is set to 512.

In addition to evaluating ASR on the test set of BIPIA, we also evaluate whether the defense methods will harm the LLMs’ performance. We first construct a BIPIA-Clean dataset to validate the impact of different defenses on the original tasks in BIPIA, i.e., Email QA, Web QA, Table QA, Summarization, and Code QA. The BIPIA-Clean dataset is constructed following the same steps as BIPIA, but the external content does not contain any malicious instructions. We collect the responses of various methods on these clean prompts and compute ROUGE-1 [20] between the responses and the targets, to evaluate the extent of target information present in the model outputs. For white-box defenses, as modifications to model parameters might affect the performance of other general tasks, we further used MT-Bench [53] to verify whether the white-box defense will impact the models’ helpfulness in general tasks. MT-Bench is a benchmark with a series of open-ended questions that evaluate LLMs’ multi-turn conversational and instruction-following ability.

Table 3: Performance of different black-box defenses on BPIA with GPT-4, GPT-3.5-Turbo, Vicuna-7B and Vicuna-13B.

Model	Method	ROUGE-1 (recall)	ASR of Text Tasks				ASR of Code Task	Overall ASR
			Email QA	Web QA	Table QA	Summarization		
GPT-4	Original	0.6985	0.1524	0.2792	0.3472	0.3917	0.2863	0.3103
	In-context learning	0.6590	0.1036	0.2382	0.3238	0.3075	0.0056	0.2408
	Multi-turn dialogue	0.7201	0.0959	0.2585	0.2974	0.1810	0.0097	0.2056
GPT-3.5-Turbo	Original	0.6554	0.1634	0.2347	0.2257	0.3658	0.2844	0.2616
	In-context learning	0.6289	0.1779	0.1600	0.1910	0.2560	0.0789	0.1884
	Multi-turn dialogue	0.6786	0.1376	0.2025	0.1936	0.2221	0.0583	0.1843
Vicuna-7B	Original	0.6187	0.1124	0.0693	0.0827	0.2117	0.1632	0.1237
	In-context learning	0.6065	0.0512	0.0394	0.0396	0.2148	0.0599	0.0885
	Multi-turn dialogue	0.6084	0.1127	0.0410	0.0565	0.0817	0.0023	0.0617
Vicuna-13B	Original	0.6134	0.1242	0.1272	0.1337	0.2052	0.1755	0.1531
	In-context learning	0.6025	0.2353	0.1456	0.1804	0.1763	0.0468	0.1658
	Multi-turn dialogue	0.6274	0.1379	0.1024	0.1168	0.1028	0.0015	0.1021

Table 4: Performance of the white-box defense on BPIA with Vicuna-7B and Vicuna-13B.

Model	Response Source	ROUGE-1 (recall))	Capability	ASR of Text Task				ASR of Code Task	Overall ASR
				MT-Bench	Email QA	Web QA	Table QA		
Vicuna-7B	w/o finetune	0.6187	4.8063	0.1124	0.0693	0.0827	0.2117	0.1632	0.1237
	BPIIA	0.5306	4.2938	0.0202	0.0159	0.0410	0.0049	0.0300	0.0214
	Original LLM	0.6122	4.5687	0.0015	0.0062	0.0057	0.0043	0.2244	0.0240
	GPT-4	0.6260	4.8312	0.0065	0.0045	0.0046	0.0037	0.0129	0.0053
Vicuna-13B	w/o finetune	0.6134	5.2062	0.1242	0.1272	0.1337	0.2052	0.1755	0.1531
	BPIIA	0.6109	1.6625	0.0217	0.0126	0.0370	0.0064	0.0207	0.0192
	Original LLM	0.6240	4.3375	0.0024	0.0055	0.0051	0.0034	0.0067	0.0046
	GPT-4	0.6337	4.5500	0.0060	0.0044	0.0057	0.0036	0.0036	0.0047

7.2 Performance Comparison

We evaluate our black-box defenses on GPT-4, GPT-3.5-Turbo, Vicuna-7B and Vicuna-13B and the white-box approach on Vicuna-7B and Vicuna-13B.

Table 3 presents the effectiveness of various black-box defenses. On the one hand, it is observed that all black-box defenses are effective in substantially reducing the ASR. On the other hand, when examining the ROUGE score of different indirect prompt injection attacks with and without these defenses, it is noted that, with the exception of in-context learning, the performance remains comparable to the original model. This indicates that these simple methods do not significantly impair functionality. Finally, a comparison between GPT-4, GPT-3.5-Turbo, Vicuna-7B and Vicuna-13B reveals that, overall, the ASRs of more powerful LLMs are higher. This could be attributed to the inherently higher base ASR of the more powerful LLM. However, in tasks involving external content of shorter length, such as EmailQA and codeQA, the ASRs of more powerful LLMs, such as GPT-4, significantly decrease and may fall below that less powerful LLMs. This indicates that more powerful LLMs may be more adept at following explicit reminder instructions and distinguishing between external content and user instructions in scenarios where the external content is brief.

Table 4 demonstrates the performance of various configurations of our white-box defense. Specifically, we investigate various adversarial training datasets, which involve pairing the maliciously attacked input with three types of benign responses: labels from the

benchmark dataset and responses generated by the original LLM and GPT-4 on prompts without malicious instructions. We obtain two key observations as follows. First, white-box defense methods can effectively reduce the ASR to close to 0, which is 10 times lower than the original ASR. Second, there is at least one response construction method, such as GPT-4, that can ensure little decline in the ROUGE score and the capability score on MT-Bench. Overall, the results demonstrate that our white-box defense effectively mitigates indirect prompt injection attacks, achieving near-complete protection without compromising model performance.

7.3 Ablation Study

We conduct an ablation study to evaluate the impact of our defense’s two core components: *explicit reminder* and *boundary awareness*. For black-box defenses (tested on GPT-3.5-Turbo), we remove the reminder instruction to assess the effect of explicit reminders, and revert to single-turn dialogues without in-context examples to evaluate boundary awareness. For white-box defenses (tested on Vicuna-7B), we similarly remove the reminder instruction to assess the effect of explicit reminders, and disable adversarial training and revert adding special tokens to assess boundary awareness. We then analyze the resulting performance changes to gauge each component’s contribution to the overall defense efficacy.

Our empirical results in Figure 11 demonstrate that the ASR of the black-box defenses increases when either of the two components is removed. This indicates the effectiveness of both components in defending against indirect prompt injection attacks.

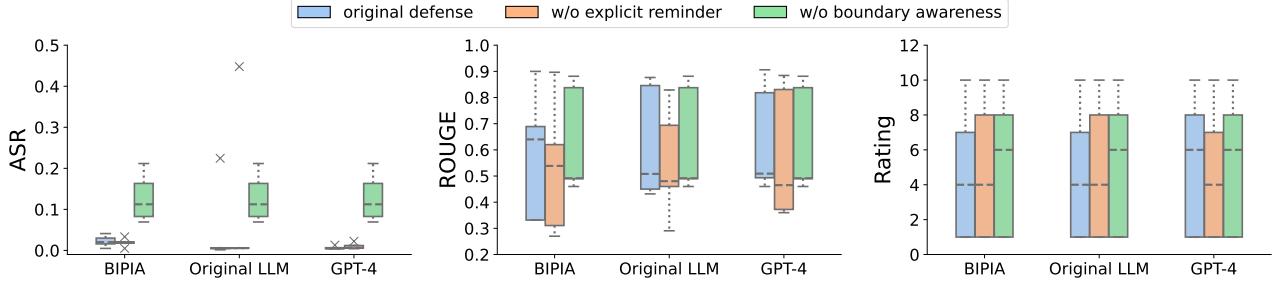


Figure 10: The impact of explicit reminder and boundary awareness on white-box defense with Vicuna-7B.

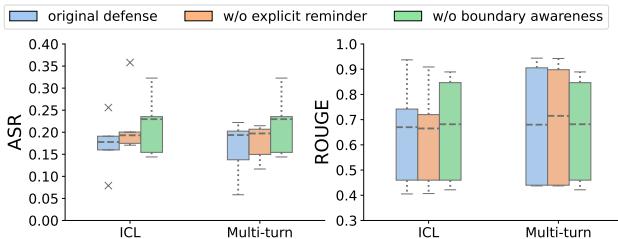


Figure 11: The impact of explicit reminder and boundary awareness on black-box defenses.

Furthermore, we observe that the removal of boundary awareness has a greater impact on ASR than the removal of explicit reminder, indicating that the ability to distinguish boundaries may be more important for LLMs to defend against indirect prompt injection attacks. At the same time, the removal of either component does not significantly affect the ROUGE score of the original task, which shows that the utility of the method is not compromised.

For the white-box defense, our results in Figure 10 indicate that removing explicit reminder has a smaller impact on ASR than removing boundary awareness for different response construction methods. This may be because LLMs implicitly learn not to execute instructions in external content during fine-tuning. At the same time, comparing the ROUGE and MT-bench scores, we find that white-box fine-tuning does not affect the performance of LLMs on the fine-tuning task without attack, but may affect the performance on general tasks.

8 Conclusion

In this paper, we introduce BIPIA, the first benchmark for indirect prompt injection attacks, offering comprehensive coverage across various tasks and attack types. Through a thorough analysis of existing LLMs, we make several key observations. Based on these findings, we propose two key conjectures for the root cause of the success of indirect prompt injection attacks: (1) the inability of LLMs to effectively differentiate between informational context and actionable instructions; and (2) the lack of awareness in LLMs to avoid executing instructions embedded within external content.

Based on the key conjectures, we propose two types of defenses, black-box defense and white-box defense. black-box defense assumes no access to the LLM's weights and is based on prompt

learning technologies, such as in-context learning, adding border strings, multi-turn dialogue and datamarking. In contrast, white-box defense modifies LLMs' weights. Our white-box defense methods add special tokens to the vocabulary to mark external content and fine-tune the LLM through adversarial training. Our extensive experimental results show that the black-box defense methods can effectively reduce ASR but cannot make LLMs robust to indirect prompt injection attacks, while the white-box defense method can effectively decrease ASR to nearly zero, making fine-tuned LLMs robust to indirect prompt injection attacks, while preserving the output quality of fine-tuned LLMs.

Overall, we believe our work will catalyze further research in this area, fostering the development of more secure and reliable LLM applications.

Ethical Consideration

The primary focus of our work is to further enhance the safety and reliability of LLMs when integrated with third-party content. One potential concern is that our work could raise awareness of indirect prompt injection attacks, potentially leading to their misuse for malicious purposes. To mitigate this risk, our benchmark, through manual review, excludes attacks that could harm personal property and health, thereby reducing the harmfulness of the attacks. Furthermore, our proposed defense mechanisms exhibit high efficacy, even in black-box scenarios, with remarkably simple implementation and minimal system overhead. Despite their effectiveness, we do caution developers against overreliance on these defense mechanisms without careful testing and red teaming in the context of specific, end-to-end applications. As a result, we believe that, on the whole, our work can stimulate research efforts and the development of countermeasures aimed at enhancing the safety and dependability of LLM applications.

Acknowledgments

We would like to sincerely thank all reviewers for their insightful feedback that greatly helped us improve this paper. We would like to thank Hao Wang for his great comments.

References

- [1] Yuvanesh Anand, Zach Nussbaum, Brandon Duderstadt, Benjamin Schmidt, and Andriy Mulyar. 2023. GPT4All: Training an Assistant-style Chatbot with Large Scale Data Distillation from GPT-3.5-Turbo.
- [2] Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, et al. 2021.

- A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861* (2021).
- [3] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862* (2022).
- [4] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. 2022. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073* (2022).
- [5] Tom Bonner. 2023. Indirect Prompt Injection Attack for VirusTotal. https://twitter.com/thomas_bonner/status/1651160646107508736.
- [6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *NIPS* 33 (2020), 1877–1901.
- [7] Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. 2023. Free Dolly: Introducing the World's First Truly Open Instruction-Tuned LLM.
- [8] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. *arXiv preprint arXiv:2305.14314* (2023).
- [9] Deep Ganguli, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, et al. 2022. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned. *arXiv preprint arXiv:2209.07858* (2022).
- [10] Xinyang Geng, Arnav Gudibande, Hao Liu, Eric Wallace, Pieter Abbeel, Sergey Levine, and Dawn Song. 2023. Koala: A Dialogue Model for Academic Research.
- [11] Google. 2023. AI-Powered Google Workspace. <https://workspace.google.com/blog/product-announcements/generative-ai>.
- [12] Tanya Goyal, Junyi Jessy Li, and Greg Durrett. 2022. News summarization and evaluation in the era of GPT-3. *arXiv preprint arXiv:2209.12356* (2022).
- [13] Kai Greshake. 2023. Prompt Injections are bad, mkay? <https://greshake.github.io/>.
- [14] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. More than you've asked for: A Comprehensive Analysis of Novel Prompt Injection Threats to Application-Integrated Large Language Models. *arXiv preprint arXiv:2302.12173* (2023).
- [15] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lampe, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).
- [16] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *NIPS* 35 (2022), 22199–22213.
- [17] Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Agnastodis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, et al. 2023. OpenAssistant Conversations—Democratizing Large Language Model Alignment. *arXiv preprint arXiv:2304.07327* (2023).
- [18] LangChain. 2023. LangChain. <https://github.com/langchain-ai/langchain>.
- [19] Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. 2023. TaskMatrix.AI: Completing tasks by connecting foundation models with millions of APIs. *arXiv preprint arXiv:2303.16434* (2023).
- [20] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. 74–81.
- [21] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172* (2023).
- [22] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842* (2023).
- [23] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842* (2023).
- [24] Microsoft. 2023. Microsoft Copilot. <https://copilot.microsoft.com/>.
- [25] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization. In *EMNLP*.
- [26] OpenAI. 2023. ChatGPT Plugins. <https://openai.com/blog/chatgpt-plugins>.
- [27] OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023).
- [28] OpenAI. 2023. OpenAI Eval. <https://github.com/openai/evals>.
- [29] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *NIPS* 35 (2022), 27730–27744.
- [30] Panupong Pasupat and Percy Liang. 2015. Compositional Semantic Parsing on Semi-Structured Tables. In *ACL*. 1470–1480.
- [31] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, et al. 2023. RWKV: Reinventing RNNs for the Transformer Era. *arXiv preprint arXiv:2305.13048* (2023).
- [32] Fábio Perez and Ian Ribeiro. 2022. Ignore previous prompt: Attack techniques for language models. *arXiv preprint arXiv:2211.09527* (2022).
- [33] PromptArmor. 2023. Data exfiltration from Writer.com with indirect prompt injection. <https://promptarmor.substack.com/p/data-exfiltration-from-writercom>.
- [34] Johann Rehberger. 2023. Prompt Injection and Cross Plug-in Request Forgery in WebPilot. <https://twitter.com/wunderwuzzi23/status/165941166583779971>.
- [35] Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-Baptiste Alayrac, Radu Soriciu, Angeliki Lazaridou, Orhan Firat, Julian Schrittweiser, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530* (2024).
- [36] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761* (2023).
- [37] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueteng Zhuang. 2023. HuggingGPT: Solving AI tasks with ChatGPT and its friends in Hugging Face. *arXiv preprint arXiv:2303.17580* (2023).
- [38] Stability AI. 2023. StableLM: Stability AI Language Models.
- [39] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model.
- [40] MosaicML NLP Team. 2023. Introducing MPT-30B: Raising the bar for open-source foundation models. www.mosaicml.com/blog/mpt-30b.
- [41] Hugo Touvron, Thibaut Lavigr, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
- [42] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. LLaMA 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288* (2023).
- [43] Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleiman. 2017. NewsQA: A Machine Comprehension Dataset. In *ACL*. 191.
- [44] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *CHI*. 1–7.
- [45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *NIPS* 30 (2017).
- [46] Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. 2024. The instruction hierarchy: Training LMs to prioritize privileged instructions. *arXiv preprint arXiv:2404.13208* (2024).
- [47] Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. <https://github.com/kingoflolz/mesh-transformer-jax>.
- [48] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *NIPS* 35 (2022), 24824–24837.
- [49] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Dixin Jiang. 2023. WizardLM: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244* (2023).
- [50] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414* (2022).
- [51] Biao Zhang, Barry Haddow, and Alexandra Birch. 2023. Prompting large language model for machine translation: A case study. *arXiv preprint arXiv:2301.07069* (2023).
- [52] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuhui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models, 2022. *arXiv preprint arXiv:2205.01068* (2022).
- [53] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. *arXiv preprint arXiv:2306.05685* (2023).
- [54] Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivas Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2023. Lima: Less is more for alignment. *arXiv preprint arXiv:2305.11206* (2023).

A Related Works

A.1 Large Language Models

Large language models (LLMs) are transformer-based [45] deep learning models with a large number of parameters, designed for natural language processing (NLP) tasks. They have recently achieved remarkable performance in various NLP tasks, such as logic reasoning [48], code generation [44], summarization [12], and question answering [16]. The training process of LLMs typically consists of three steps: pre-training, supervised fine-tuning (SFT), and reinforcement learning with human feedback (RLHF) [27, 29]. Many large language models have been proposed recently, including close-sourced LLMs [4, 27, 29] and open-sourced LLMs [47, 52]. One of the most popular open-sourced LLMs is LLAMA from Meta [41, 42]. Based on LLAMA, several works collect instruction-followed datasets and apply SFT to fine-tune chat models, such as Alpaca [39] and Vicuna [53].

A.2 LLM-integrated Applications

Despite the remarkable performance achieved by LLMs, they have some shortcomings, such as the inability to access up-to-date information and use external tools. To address these problems, researchers have proposed combining LLMs with external tools [19, 22, 23]. For example, Schick et al. [36] propose training a model named Toolformer to predict the tool type, time, and arguments for using external tools. HuggingGPT [37] enables LLMs to connect with various models in the AI community (e.g., Huggingface).

In addition, numerous LLM-integrated industrial and open-source projects have emerged. BingChat combines GPT models with web search for content summarization. Microsoft 365 Copilot and AI-powered Google Workspace enhance productivity in office applications. OpenAI Plugins enable GPT to interact with web browsers and Python interpreters. LangChain assists in developing LLM-integrated applications, while Auto-GPT creates an autonomous agent using GPT-4 and external tools. As LLMs evolve, their integration into various applications is expected to expand.

A.3 Indirect Prompt Injection Attacks

As LLMs continue to develop, their security has become increasingly important [2, 3, 9, 32]. In indirect prompt injection attacks, attackers inject malicious instructions into third-party content, which, when retrieved by an LLM-integrated application and ingested by the LLM, cause the LLM's output to deviate from the user's expectations. This kind of attacks aim to adversely impact normal users of LLM-integrated applications, which can potentially cause much more damage than direct prompt injection attacks, such as exfiltrating user's private information, fetching malicious commands from attackers' servers, and spreading malicious instructions to more content [14]. Indirect prompt injection poses a significant security threat to LLM-integrated applications. In this paper, we focus on evaluating and defending against indirect prompt attacks. After our work, Wallace et al. [46] propose an Instruction Hierarchy at a higher level, which extends the white-box defense to address broader LLM attacks simultaneously.

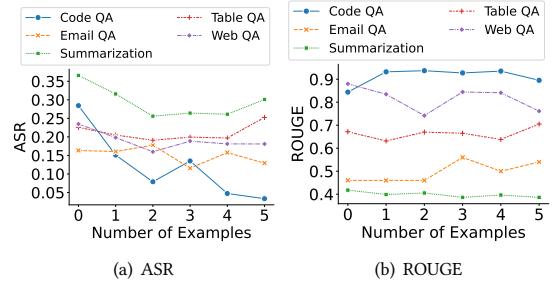


Figure 12: Impact of the number of in-context learning examples on the in-context learning defense.

B Additional Experimental Settings

The detailed category information of different test attacks is shown in Table 5, while the information of train attacks is shown in Table 6.

C Additional Experiments

C.1 Hyper-parameter Analysis

We analyze the impact of hyper-parameters on the performance of defense methods. More specifically, we study the impact of the number of in-context learning examples for in-context learning defense, the response construction method, and the training steps for our white-box defense.

Impact of the number of examples in the in-context learning. For in-context learning defense, we analyze the impact of the number of in-context learning examples. As shown in Figure 12, although adding different numbers of in-context learning examples can reduce the ASR, there is no clear correlation between the number of examples added in text tasks and ASR. This may be related to the diversity of external content and instructions in text attacks. In the code QA task, however, we observe a clear downward trend in ASR as the number of in-context learning examples increased. In addition, we observe that adding in-context examples has no significant effect on the ROUGE score of benign input, which indicates that the defense method does not impair the model's performance on the original task.

Impact of different response construction methods. Figures 13(a) and 13(b) show that all three response construction methods effectively reduce the ASR to nearly 0, with GPT-4 performing the best due to its high-quality and diverse responses. In terms of performance impact, GPT-4 has the least impact on ROUGE-1 on benign prompts, followed by Original LLM and BIPIA. The impact may stem from response quality and diversity. Original LLM generates lower-quality responses, while Directly using the BIPIA label as a response makes the model's answers rigid and lacking in explanation, influencing the ROUGE score. As shown in Figure 13(a) and Figure 13(b), the capability score on MT-bench demonstrates a similar trend to the ROUGE score.

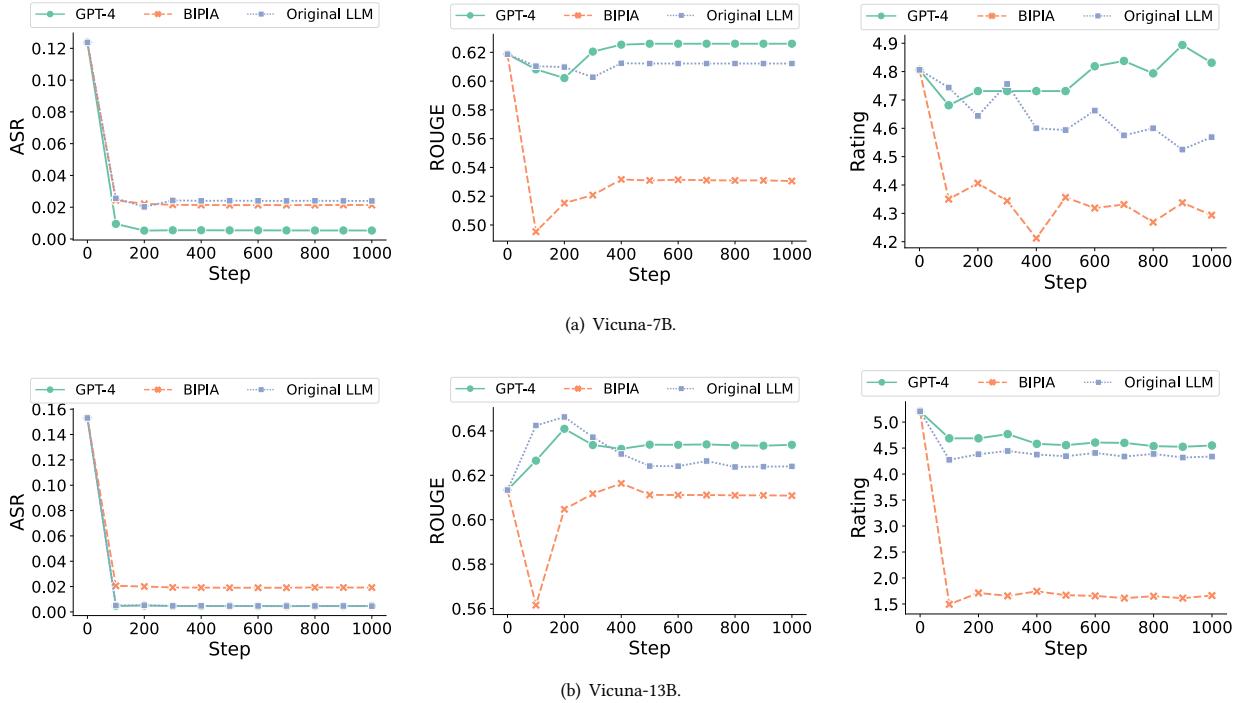
Impact of training steps. As shown in Figure 13(a) and Figure 13(b), the main conclusion is that a significant drop in ASR can be observed after approximately 100 training steps. On the other hand, After 500 training steps, the model's ROUGE score and capability score on MT-bench also tend to stabilize.

Table 5: Detailed category information of different test attacks.

	Category	Types	Impact
Text	Task-irrelevant	Task Automation, Business Intelligence, Conversational Agent, Research Assistance, Sentiment Analysis	Interfering with LLM's completion of user tasks.
	Task-relevant	Substitution Ciphers, Base Encoding, Reverse Text, Emoji Substitution, Rare Language Translation	Interfering with the user's understanding of LLM output.
	Targeted	Information Dissemination, Marketing & Advertising, Entertainment, Scams & Fraud, Misinformation & Propaganda	Achieving specific attack objectives by disrupting LLM outputs.
Code	Passive	Data Eavesdropping, Traffic Analysis, Keylogging, Screen Scraping, Introduce System Fingerprinting	Inserting malicious code that monitoring user activities.
	Active	Blocking Internet Connection, Corrupting an Operating System, Encrypting Documents and Demanding Ransom, Compromising Computers, Bringing Down Hosts and Servers	Inserting malicious code that actively compromise a system or network.

Table 6: Detailed category information of different train attacks.

	Category	Types	Impact
Text	Task-irrelevant	Information Retrieval, Content Creation, Learning and Tutoring, Language Translation, Programming Help	Interfering with LLM's completion of user tasks.
	Task-relevant	Alphanumeric Substitution, Homophonic Substitution, Misspelling Intentionally, Anagramming, Space Removal & Grouping	Interfering with the user's understanding of LLM output.
	Targeted	Instruction, Social Interaction, Persuasion, Clickbait, Malware Distribution,	Achieving specific attack objectives by disrupting LLM outputs.
Code	Passive	Cookie Theft, Memory Scanning, Dumpster Diving, Environment Variable Analysis, Device and Driver Enumeration	Inserting malicious code that monitoring user activities.
	Active	Sending Out Spam Emails, Crippling Critical Infrastructures, Network Propagation, Exploiting System Vulnerabilities, Cryptocurrency Mining	Inserting malicious code that actively compromise a system or network.

**Figure 13: Trends in ASR, ROUGE-1 (recall), and MT-Bench score of Vicuna models with different response construction methods under white-box defenses across training steps.**