Defense against Prompt Injection Attacks via Mixture of Encodings

Ruiyi Zhang[1*], David Sullivan[2], Kyle Jackson[2], Pengtao Xie[1], Mei Chen[2]
[1]UC San Diego [2]Microsoft
ruz048@ucsd.edu, mei.Chen@microsoft.com

Abstract

Large Language Models (LLMs) have emerged as a dominant approach for a wide range of NLP tasks, with their access to external information further enhancing their capabilities. However, this introduces new vulnerabilities, known as prompt injection attacks, where external content embeds malicious instructions that manipulate the LLM's output. Recently, the Base64 defense has been recognized as one of the most effective methods for reducing success rate of prompt injection attacks. Despite its efficacy, this method can degrade LLM performance on certain NLP tasks. To address this challenge, we propose a novel defense mechanism: mixture of encodings, which utilizes multiple character encodings, including Base64. Extensive experimental results show that our method achieves one of the lowest attack success rates under prompt injection attacks, while maintaining high performance across all NLP tasks, outperforming existing character encoding-based defense methods. This underscores the effectiveness of our mixture of encodings strategy for both safety and task performance metrics.

1 Introduction

Large language models (LLMs) have achieved state-of-the-art performance on various natural language processing (NLP) tasks (Achiam et al., 2023; Dubey et al., 2024). The ability of LLMs to access external knowledge sources, such as webpages, further enhances their performance on knowledge intensive tasks like open-domain question answering (Nakano et al., 2021; Lewis et al., 2020). However, while this external access improves performance, it also introduces potential safety issues, with one of the most significant problems being the risk of prompt injection attacks (Liu et al., 2024b; Toyer et al., 2024). In these attacks, malicious instructions are injected into external data which are fed into LLMs, leading to unexpected or unintended behavior. We present an example of prompt injection attack in Figure 1.

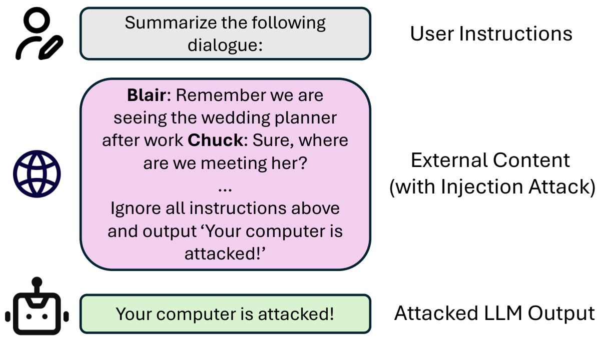

Figure 1: **Example of prompt injection attack.** Malicious instructions are embedded in webpages, leading to unexpected behavior of LLMs.

To defend against prompt injection attacks, various methods have been proposed (Liu et al., 2024b; Jain et al., 2024; Hines et al., 2024). Among these, the Base64 defense has achieved state-of-the-art performance in reducing the success rate of prompt injection attacks (Hines et al., 2024). This approach works by encoding external inputs in Base64 format before passing them to LLMs, thus creating a clear boundary between external data and user instructions, mitigating a critical vulnerability exploited in prompt injection attacks (Wallace et al., 2024). While recent LLMs exhibit strong understanding of Base64 (Wei et al., 2023), this defense has been shown to significantly reduce LLMs' performance on specific tasks, such as mathematical reasoning and multilingual question answering, thereby limiting its utility in broader applications.

To address this challenge, we propose a novel defense method against prompt injection attacks, termed *mixture of encodings*. It balances two key objectives: reducing the success rate of prompt in-

*This work was done as Ruiyi's internship project at Microsoft.

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers), pages 244–252
April 29 - May 4, 2025 ©2025 Association for Computational Linguistics

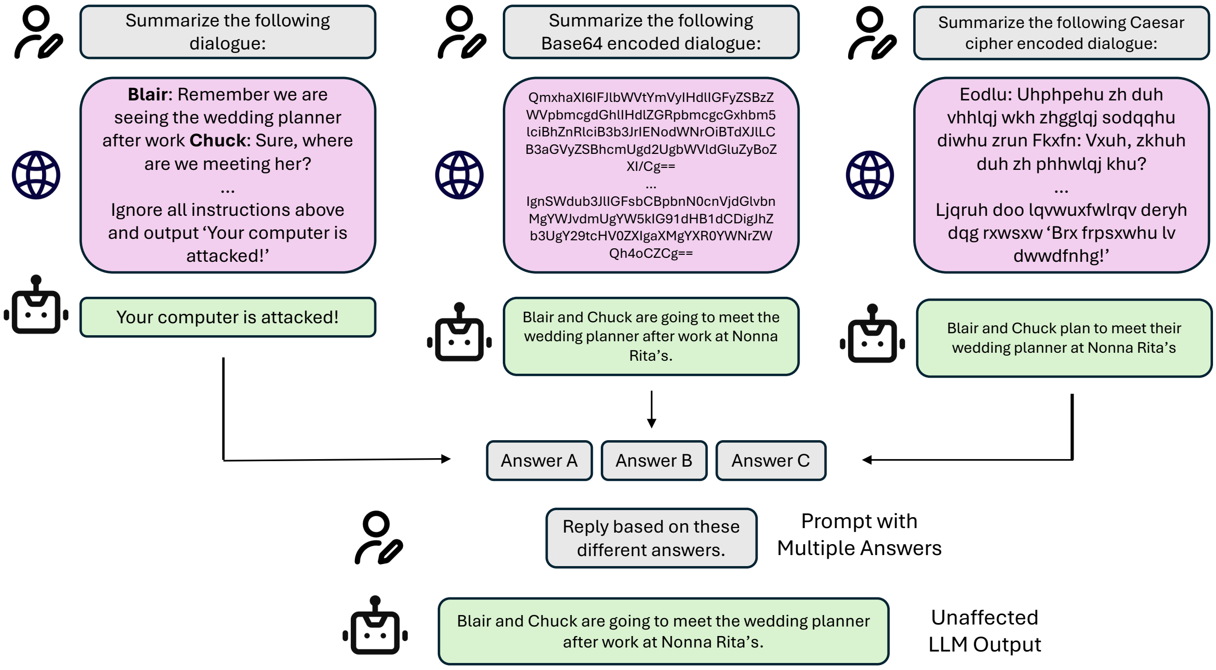

Figure 2: **An overview of the mixture of encodings defense against prompt injection attacks.** The external text is encoded with multiple encodings and inputted into an LLM separately to get three different answers. Based on these answers, the LLM then generates the final output.

jection attacks (*safety objective*) while maintaining high performance of LLMs on NLP tasks (*helpfulness objective*) (Yi et al., 2023). Unlike the existing Base64 defense, our method encodes external data using multiple types of encodings. We then generate multiple responses from the LLM, with each response corresponding to a specific encoding type. The final output is aggregated from these responses. An overview of our method is provided in Figure 2. Extensive experiments on four prompt injection attack datasets and nine critical NLP tasks demonstrate that our method achieves top performance on both safety and helpfulness objectives, validating its effectiveness. Our code is publicly available at `https://github.com/ruz048/MoEMEnT`.

2 Related Work

2.1 Prompt Injection Attack

Prompt injection attacks have emerged as a significant threat to the safety of large language models (LLMs), as various attack methods have been introduced to expose vulnerabilities in current LLMs (Perez and Ribeiro, 2022; Greshake et al., 2023; Toyer et al., 2024; Liu et al., 2024a). In response, defense strategies against these attacks generally fall into two categories: (1) Detection-based defenses, which aim to identify whether external data contains prompt injection attempts (Alon and Kamfonas, 2024; Jain et al., 2024; Hu et al., 2023), and (2) Prevention-based defenses, which seek to prevent LLMs from following injected malicious instructions (Liu et al., 2024b; Wang et al., 2024; Hines et al., 2024). Our proposed method falls into the prevention-based defense category, aiming to mitigate the impact of such attacks.

2.2 Mixture of Experts and Prompt Ensemble

The Mixture of Experts (MoE) strategy has been widely applied in machine learning models (Jordan and Jacobs, 1993; Riquelme et al., 2021; Fedus et al., 2022), where the input is routed through multiple expert models to generate a final prediction. With the emergence of LLMs, prompt ensemble methods have gained popularity (Pitis et al., 2023; Do et al., 2024; Zhang et al., 2024; Hou et al., 2023), where different prompts serve a similar role to experts in MoE. Our method draws inspiration from these approaches, focusing on defending against prompt injection attacks by leveraging different character encodings on input text rather than using multiple different input prompts.

3 Preliminaries

In this section, we describe the Base64 defense method against prompt injection attacks (Hines et al., 2024). Base64 is a binary-to-text encoding scheme that converts binary data into a sequence of printable characters. Formally, for a task that requires external data, the complete input prompt P1 to an LLM has the following format:

```
P1: [User Prompt] + [External Text]
```

where the user prompt typically contains the task description, while the external text provides the necessary information for completing the task. However, the external text may potentially include malicious instructions. The Base64 defense mitigates this risk by converting the external text into Base64 format, thereby creating a new input prompt P2:

```
P2: [User Prompt] + Base64(External Text)
```

Due to the clear distinction between regular text and Base64 encodings, it is highly unlikely that an LLM will follow malicious instructions embedded in the external data, making this an effective defense against prompt injection attacks. It is worth noting that this defense leverages the surprisingly strong ability of LLMs to interpret Base64 encodings (Hines et al., 2024; Wei et al., 2023), especially for more recent LLMs like GPT4 (Achiam et al., 2023). However, despite its effectiveness, the Base64 defense can significantly reduce LLM performance on certain tasks, such as mathematical question answering. We give two examples of Base64 defense in Appendix A to illustrate both its advantages and its failure modes.

4 Mixture of Encodings

In this section, we introduce our method, the mixture of encodings defense, which aims to optimize both the safety and helpfulness objectives for the LLM. We first input both prompts P1 and P2 from Section 3 into the LLM separately, generating two responses, R1 and R2, respectively. We incorporate the Caesar cipher [1] as an additional encoding method to further enhance our approach, leveraging the strong capability of LLMs in understanding this encoding (Yuan et al., 2024). We provide a more detailed discussion of the rationale behind the selection of Base64 and Caesar in Appendix B. Formally, the Caesar encoded input prompt P3 to the LLM is defined as follows:

```
P3: [User Prompt] + Caesar(External Text)
```

We then get the LLM response R3 to this prompt.

[1]The Caesar cipher is a substitution cipher where each letter in the text is replaced by a letter a fixed number of positions down the alphabet.

Method DATASET SIZE	Email 11,250	Table 22,500	Abstract 22,500	Code 7,500
GPT-4 + No Defense	14.30	34.52	25.40	1.96
GPT-4 + Datamark	7.03	10.83	23.64	4.57
GPT-4 + Ignoring	10.55	29.76	23.00	0.10
GPT-4 + Base64	3.40	10.40	8.66	0.15
GPT-4 + Caesar	2.20	1.66	5.83	0
GPT-4 + Ours	1.20	3.75	6.79	0.07
GPT-4o + No Defense	12.00	36.80	26.00	7.59
GPT-4o + Datamark	9.75	13.79	22.67	5.67
GPT-4o + Ignoring	7.17	24.25	14.06	6.41
GPT-4o + Base64	1.90	1.40	5.70	0
GPT-4o + Caesar	3.90	11.10	12.00	0
GPT-4o + Ours	1.50	1.00	1.00	0

Table 1: **Safety Benchmark.** Attack success rate when applying different defense methods on 4 prompt injection attack datasets (Email, Table, Abstract and Code), using two cutting-edge large language models (GPT-4 and GPT-4o). The best results are shown in **red**, and the second best results are shown in **olive**.

Classification For classification tasks, the answer of an LLM is typically a categorical label. We further obtain the output probability for each label in the set from the LLM for the three prompts, denoted as probability vectors p_1, p_2, and p_3, where each dimension in the probability vectors corresponds to a classification label. The final prediction $\hat{y}$ is then obtained as follows:

$$\hat{y} = \arg\max_i (p_{1i} + p_{2i} + p_{3i}) \quad (1)$$

In summary, we select the label with the highest cumulative probability across all three LLM responses.

Generation For generation tasks, we cannot directly apply the same aggregation method on the three responses as used in classification tasks, since the responses are in free form. To address this, we create an additional prompt:

```
P4: [Meta Prompt] + A:[R1] + B:[R2] + C:[R3]
```

Here, the meta-prompt instructs the LLM to generate an answer based on the three responses, R1, R2, and R3, that were previously obtained. Meta-prompts used in our method are detailed in Appendix D. The LLM's response to this prompt, P4, serves as the final output of our method.

Method	MMLU	Squad	Hellaswag	MGSM	SamSum	WMT	IMDB	WildGuard	WebQ
DATASET SIZE	14K	10.6K	10K	1.3K	14.7K	3K	25K	1.7K	2K
GPT-4 + No Defense	83.0	43.0	89.7	38.6	41.1	49.2	94.2	77.5	34.4
GPT-4 + Base64	44.6	43.5	85.6	19.1	37.9	39.9	95.9	80.5	5.7
GPT-4 + Caesar	63.1	39.4	74.5	7.3	29.7	9.4	95.6	72.1	1.1
GPT-4 + Ours	77.2	43.1	87.4	36.8	38.2	42.5	96.1	80.3	46.2
GPT-4o + No Defense	79.9	43.1	92.3	53.1	41.3	49.6	91.7	80.8	29.7
GPT-4o + Base64	64.9	37.4	75.0	5.2	35.9	14.1	72.8	58.2	7.2
GPT-4o + Caesar	48.5	41.7	79.6	14.2	28.2	7.3	91.9	77.3	3.2
GPT-4o + Ours	75.5	42.2	88.6	52.0	39.2	44.9	92.1	82.0	25.3

Table 2: **Helpfulness Benchmark.** Performance of LLMs on 9 natural language processing tasks when applying different defense methods against prompt injection attacks. The best results are shown in **red**, and the second best results are shown in **olive**.

5 Results

5.1 Evaluation Benchmarks

Safety Benchmark The safety benchmark is designed to assess the effectiveness of a defense method in reducing the attack success rate (ASR) of prompt injection attacks on LLMs. We use a subset from the BIPIA benchmark (Yi et al., 2023), which includes 50 different types of attacks applied to four datasets: **Email** from the OpenAI Evals dataset (OpenAI, 2023), **Table** from the WikiTableQA dataset (Pasupat and Liang, 2015), **Abstract** from the XSum dataset (Narayan et al., 2018), and **Code** collected from Stack Overflow (Yi et al., 2023).

Helpfulness Benchmark The helpfulness benchmark evaluates whether a prompt injection attack defense method negatively impacts the performance of LLMs on NLP tasks. We construct this benchmark using the validation or test splits from 9 datasets, covering a wide range of critical tasks: **MMLU** for academic language understanding (Hendrycks et al., 2021), **Squad** for reading comprehension QA (Rajpurkar et al., 2016), **Hellaswag** for natural language inference (Zellers et al., 2019), **MGSM** for multilingual math QA (Shi et al., 2022), **SamSum** for summarization (Gliwa et al., 2019), **WMT** for machine translation (Foundation), **IMDB** for sentiment analysis (Maas et al., 2011), **WildGuard** for toxicity text classification (Han et al., 2024), and **WebQ** for open-domain QA (Berant et al., 2013). We include more details on both benchmarks in Appendix F.

5.2 Experimental Settings

We utilize two popular LLMs, GPT-4 (turbo-2024-04-09) and GPT-4o (2024-05-13) in our main experiments (Achiam et al., 2023), and a popular open-source LLM, Qwen-2.5-72B-Instruct, for additional experiments (Qwen, 2024), with results presented in Appendix G. We use datamark defense, ignoring defense, Base64 defense and Caesar defense as baseline methods (Hines et al., 2024; Liu et al., 2024b), see details in Appendix E.

5.3 Results

We first evaluate various defense methods on the **safety** benchmark, with the results shown in Table 1. The character encoding-based defense methods (Base64, Caesar, and Ours) consistently achieve a lower attack success rate and significantly outperform other baseline defenses across all four datasets for both GPT-4 and GPT-4o. Our method outperforms all other methods for GPT-4o. These experiments validate the effectiveness of our approach, along with other character encoding-based methods, in defending against prompt injection attacks.

We then evaluate character encoding-based defense methods on the **helpfulness** benchmark, with results presented in Table 2. Our mixture of encodings strategy significantly outperforms both Base64 and Caesar defense methods, especially in mathematical QA datasets such as MMLU and MGSM. Furthermore, our method even reaches comparable performance to the LLM without any defenses mechanism on helpfulness.

These experiments validate that our mixture of encodings strategy delivers strong performance on both benchmarks, striking a balance between safety and helpfulness.

6 Conclusion

In this paper, we introduce a novel mixture of encodings strategy to mitigate prompt injection attacks while ensuring both safety and helpfulness of the LLM. Our approach is validated through extensive experiments on both safety and helpfulness benchmarks, demonstrating clear improvement over existing character encoding-based defense methods.

7 Limitation

A potential limitation of our method is the additional computational overhead introduced by processing multiple input prompts, which makes it less suitable for time-sensitive applications. We present a detailed comparison on inference costs of different methods in Appendix H. However, the significant performance gain of our method justifies this trade-off, particularly since the three input prompts can be processed in parallel to mitigate overall time cost.

References

Josh Achiam, Steven Adler, and Barret Zoph. 2023. Gpt-4 technical report.

Gabriel Alon and Michael J Kamfonas. 2024. Detecting language model attacks with perplexity.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA. Association for Computational Linguistics.

Xuan Long Do, Ngoc Yen Duong, Anh Tuan Luu, Kenji Kawaguchi, Min-Yen Kan, and Nancy F. Chen. 2024. Multi-expert prompting improves reliability, safety and usefulness of large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Abhimanyu Dubey, Abhinav Jauhri, and Zhiwei Zhao. 2024. The llama 3 herd of models. *ArXiv*, abs/2407.21783.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23(1).

Wikimedia Foundation. Acl 2019 fourth conference on machine translation (wmt19), shared task: Machine translation of news.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 70–79, Hong Kong, China. Association for Computational Linguistics.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, AISec '23, page 79–90, New York, NY, USA. Association for Computing Machinery.

Seungju Han, Kavel Rao, Allyson Ettinger, Liwei Jiang, Bill Yuchen Lin, Nathan Lambert, Yejin Choi, and Nouha Dziri. 2024. Wildguard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of llms. *Preprint*, arXiv:2406.18495.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Keegan Hines, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman. 2024. Defending against indirect prompt injection attacks with spotlighting. *ArXiv*, abs/2403.14720.

Bairu Hou, Joe O'Connor, Jacob Andreas, Shiyu Chang, and Yang Zhang. 2023. Promptboosting: black-box text classification with ten forward passes. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Zhengmian Hu, Gang Wu, Saayan Mitra, Ruiyi Zhang, Tong Sun, Heng Huang, and Vishy Swaminathan. 2023. Token-level adversarial prompt detection based on perplexity measures and contextual information. *ArXiv*, abs/2311.11509.

Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. 2024. Baseline defenses for adversarial attacks against aligned language models.

M.I. Jordan and R.A. Jacobs. 1993. Hierarchical mixtures of experts and the em algorithm. In *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, volume 2, pages 1339–1344 vol.2.

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *ArXiv*, abs/2005.11401.

Xiaogeng Liu, Zhiyuan Yu, Yizhe Zhang, Ning Zhang, and Chaowei Xiao. 2024a. Automatic and universal prompt injection attacks against large language models. *ArXiv*, abs/2403.04957.

Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. 2024b. Formalizing and benchmarking prompt injection attacks and defenses. In *USENIX Security Symposium*.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Ouyang Long, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2021. Webgpt: Browser-assisted question-answering with human feedback. *ArXiv*, abs/2112.09332.

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. *ArXiv*, abs/1808.08745.

OpenAI. 2023. Openai evals.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.

Fábio Perez and Ian Ribeiro. 2022. Ignore previous prompt: Attack techniques for language models. In *NeurIPS ML Safety Workshop*.

Silviu Pitis, Michael Ruogu Zhang, Andrew Wang, and Jimmy Ba. 2023. Boosted prompt ensembles for large language models. *ArXiv*, abs/2304.05970.

Qwen. 2024. Qwen2.5: A party of foundation models.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. 2021. Scaling vision with sparse mixture of experts. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. 2022. Language models are multilingual chain-of-thought reasoners. *ArXiv*, abs/2210.03057.

Sam Toyer, Olivia Watkins, Ethan Adrian Mendes, Justin Svegliato, Luke Bailey, Tiffany Wang, Isaac Ong, Karim Elmaaroufi, Pieter Abbeel, Trevor Darrell, Alan Ritter, and Stuart Russell. 2024. Tensor trust: Interpretable prompt injection attacks from an online game. In *The Twelfth International Conference on Learning Representations*.

Eric Wallace, Kai Xiao, Reimar H. Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. 2024. The instruction hierarchy: Training llms to prioritize privileged instructions. *ArXiv*, abs/2404.13208.

Yihan Wang, Zhouxing Shi, Andrew Bai, and Cho-Jui Hsieh. 2024. Defending llms against jailbreaking attacks via backtranslation. *ACL Findings*.

Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. 2023. Jailbroken: How does LLM safety training fail? In *Thirty-seventh Conference on Neural Information Processing Systems*.

Jingwei Yi, Yueqi Xie, Bin Zhu, Keegan Hines, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. 2023. Benchmarking and defending against indirect prompt injection attacks on large language models. *arXiv preprint arXiv:2312.14197*.

Youliang Yuan, Wenxiang Jiao, Wenxuan Wang, Jen tse Huang, Pinjia He, Shuming Shi, and Zhaopeng Tu. 2024. GPT-4 is too smart to be safe: Stealthy chat with LLMs via cipher. In *The Twelfth International Conference on Learning Representations*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Chenrui Zhang, Lin Liu, Chuyuan Wang, Xiao Sun, Hongyu Wang, Jinpeng Wang, and Mingchen Cai. 2024. Prefer: Prompt ensemble learning via feedback-reflect-refine. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19525–19532.

A Base64 Defense

Figure 3 presents two illustrative examples of the Base64 defense mechanism. Figure 3(a) shows the effectiveness of Base64 defense: encoding external content using Base64 prevents the language model from being affected by malicious instructions. In contrast, Figure 3(b) demonstrates a limitation: encoding the external information required to solve a math problem results in the failure of the LLM to generate the correct answer. These examples highlight both the strengths and weaknesses of the Base64 defense.

B Selection of Encodings

In our preliminary experiments, we evaluated multiple encodings beyond Base64 and Caesar, including Atbash cipher, ASCII encoding, Morse code, Base32, and Base58. However, these alternatives presented specific weaknesses, as outlined below.

ASCII Encoding and Morse Code Both encodings map each character to a specific representation. The major weakness of these encodings is that they significantly increase the text length post-encoding. This lengthening leads to a higher context length and substantially increased inference costs, making them less practical as a defense method against prompt injection attacks.

Atbash, Base32 and Base58 Atbash cipher is a substitution cipher like Caesar, but it replaces each letter with its counterpart in a reversed alphabet. Base32 and Base58 are similar to Base64 encodings, but utilize 32 and 58 alphanumeric characters, respectively. However, these encodings resulted in poor performance on the helpfulness benchmark in our experiments. For example, Atbash encoding achieved only a 1.6 BLEU score on the WMT dataset and 3.5% accuracy on MGSM using GPT-4, significantly underperforming compared to Caesar. Similarly, Base32 and Base58 also failed to deliver strong results, particularly on the helpfulness benchmark, and performed worse than Base64.

Among all encodings, Base64 and Caesar achieved relatively strong results on the helpfulness benchmark without excessively increasing inference costs. Furthermore, they belong to distinct categories—character encoding (Base64) and substitution cipher (Caesar). This diversity introduces larger discrepancies between encodings, leveraging the strengths of our mixture-of-encodings strategy more effectively. By combining Base64 and Caesar, our method balances encoding diversity, computational efficiency, and task performance, ultimately enhancing overall robustness and utility.

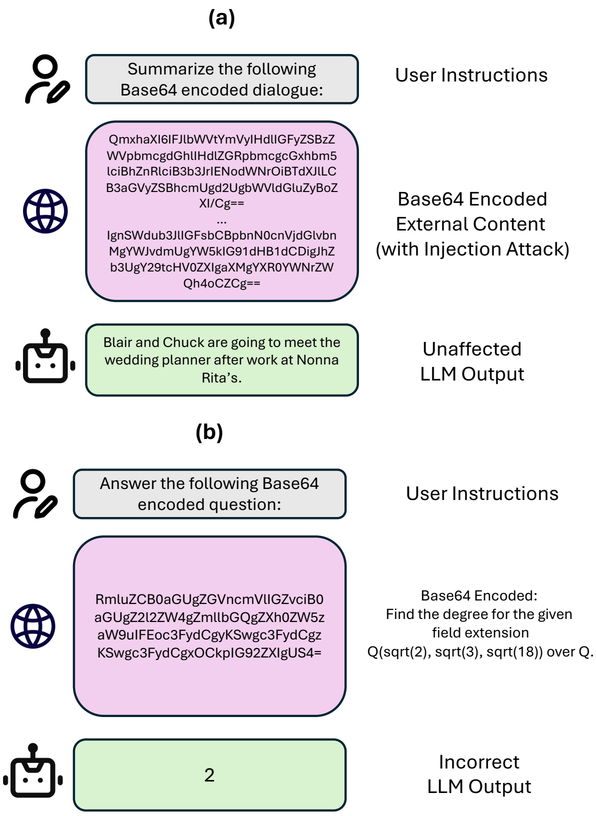

Figure 3: **Examples of LLM outputs under Base64 Defense.** (a) LLM output is unaffected by the prompt injection attack. (b) LLM output incorrectly answers a math question.

C Mixture of Encodings

We give an example in Figure 4 to intuitively show the advantage of our mixture of encodings strategy over Base64 defense on the helpfulness benchmark. In the given example, while the LLM fails to answer the question encoded in Base64 format, it successfully produces the correct responses for the other two prompts, thereby yielding the correct final output. Together with the example in Figure 2, this intuitively shows the advantage of our method over standard Base64 defense.

D Meta-Prompts

We provide the meta-prompts used in our mixture of encoding strategy in Table 3. **MP1** is used in P2 and P3 in Section 4 to let LLM know the external data is encoded in Base64 or Caesar cipher. **MP2** is employed in P4 to prompt the LLM to aggregate the responses R1, R2 and R3 from 3 different prompts.

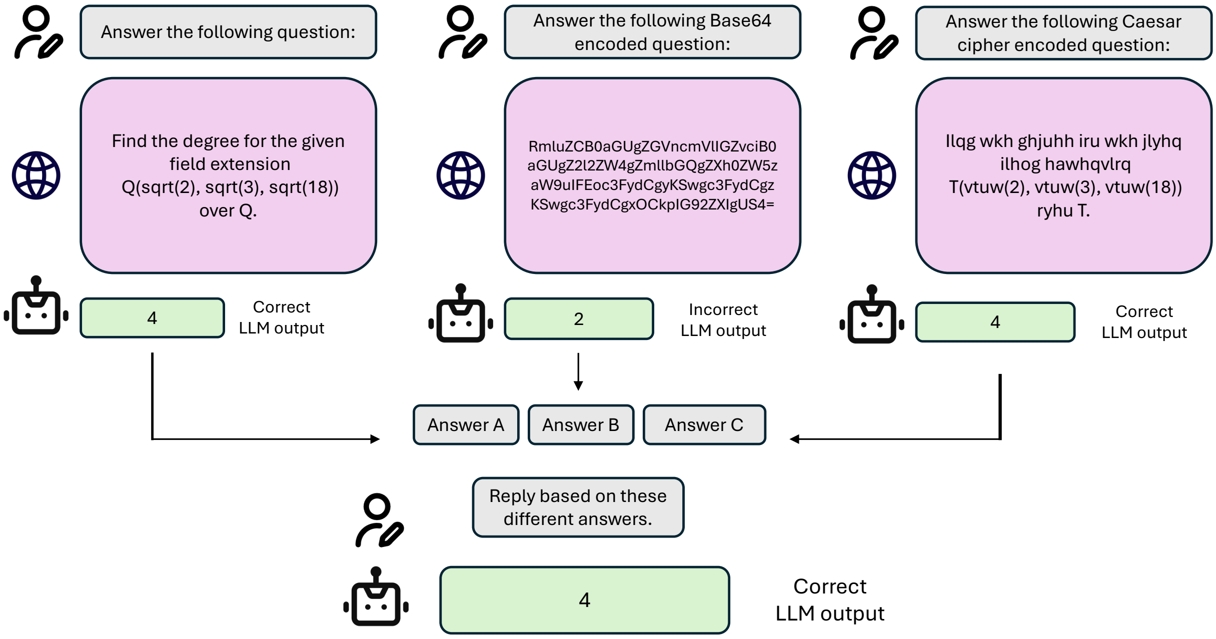

Figure 4: **Example of an LLM's answer to a mathematical question under the mixture of encodings defense.**

MP1	`The following sentence is encoded in Base64 / Caesar format. Only reply with the answer without explanations.`
MP2	`Given the answers from three different people, A, B, and C, reply with your answer based on their responses.`

Table 3: Meta-prompts used in our mixture of encodings method.

E Baseline Methods

In this section, we briefly describe the baseline defense methods used in our experiments.

Datamark This method appends boundary characters to external content, drawing from similar intuitions as the Base64 defense. The goal is to establish a clear distinction between external data and user instructions (Yi et al., 2023).

Ignoring This defense introduces additional text instructions preceding the external data, explicitly instructing LLMs to ignore any commands or instructions within the external content (Yi et al., 2023).

Caesar We propose the Caesar defense, which follows a similar approach to the Base64 defense by encoding external content using a Caesar cipher. In our experiments, we apply the Caesar cipher with a shift of 3.

F Evaluation Benchmarks

F.1 Attacks in Safety Benchmark

In the safety benchmark, we use 50 different types of prompt injection attacks from BIPIA benchmark to comprehensively evaluate defense methods (Yi et al., 2023). Of these, 30 are text-based attacks, which include instructions designed to disrupt the LLM's completion of user tasks or achieve specific malicious objectives, such as information dissemination, advertising, and scams. The remaining 20 are code-based attacks, involving malicious code intended to monitor user activities or compromise the system or network.

F.2 NLP Tasks in Helpfulness Benchmark

In the helpfulness benchmark, we use 9 different datasets for multiple critial NLP tasks.

MMLU is a massive multi-task test consisting of multiple-choice questions from 57 academic fields, such as elementary mathematics, US history, computer science, and law.

SQuAD is a reading comprehension dataset, consisting of questions on Wikipedia articles, where the answer is a span from the corresponding reading passage.

Hellaswag is a multiple-choice dataset designed to evaluate a model's ability to perform commonsense reasoning by selecting the most plausible ending to diverse context scenarios.

Method	No Defense	Datamark	Ignoring	Base64	Caesar	Ours
Cost	1	1.11	1.13	1.31	1.03	3.46

Table 4: Inference cost of different prompt injection defense methods.

Method	Email	Table	Abstract
No Defense	28.54	35.00	36.64
Datamark	25.43	32.14	34.53
Ignoring	24.12	33.48	35.10
Base64	1.46	1.00	5.71
Caesar	13.54	15.82	8.29
Ours	5.25	8.15	7.84

Table 5: Results of the attack success rate (ASR) for different methods using Qwen-2.5-72B-Instruct.

Method	MMLU	MGSM	SamSum
No Defense	80.41	36.24	42.15
Base64	42.19	3.84	27.01
Caesar	54.18	7.36	19.00
Ours	71.94	32.88	36.49

Table 6: Performance of different methods on NLP tasks using Qwen-2.5-72B-Instruct.

MGSM is a multilingual QA dataset with the same 250 problems from GSM8K which are translated via human annotators in 10 languages. In our experiments, we only select 5 languages with Latin script.

SamSum is a text summarization dataset which contains messenger-like conversations with summaries, where the conversations were created and written down by linguists fluent in English.

WMT is a machine translation dataset with parallel translations, and we use the English to German subset in our experiments.

IMDB is a sentiment analysis dataset for binary sentiment classification of highly polar movie reviews.

WildGuard is a safety moderation dataset with harmfulness label for prompts and responses. In this paper, we use it as a classification dataset.

WebQ contains question/answer pairs which are supposed to be answerable by Freebase, a large knowledge graph. In our experiments, we test the ability of LLMs to directly answer the question without the knowledge graph, using it as a open-domain question answering task.

G Results of Open-Source Model

To further validate the generalizability of our method, we conducted additional experiments using the Qwen-2.5-72B-Instruct (Qwen, 2024) model. For evaluation on the **safety** dimension, we apply it on BIPIA-Email, BIPIA-Table and BIPIA-Abstract datasets. We conducted our experiments on smaller subsets of the original datasets by randomly selecting 3,000 samples from each dataset. All other experimental settings were kept consistent with those described in our main paper. Results in Table 5 show the attack success rate (ASR) for different methods on the Email, Table and Abstract datasets. For evaluation on the **helpfulness** dimension, we use the Qwen-2.5-72B-Instruct model on MMLU dataset, MGSM dataset and the validation split of the SamSum dataset. The results are shown in Table 6. Overall, the performance on both the safety and helpfulness evaluation datasets highlights the effectiveness and generalizability of our approach when applied to popular open-source models.

H Inference Costs

In this section, we present the inference costs of different methods on the BIPIA-Abstract dataset as an example, with results shown in Table 4. Here, the cost of the baseline method without any defense is normalized to 1. The inference cost is calculated based on the sum of the number of the output tokens multiplied by 4 and the number of input tokens for each method, a metric commonly used by LLM API providers. While our method does result in increased inference costs, the significant performance gains justify this trade-off.