Human-Imperceptible Retrieval Poisoning Attacks in LLM-Powered Applications

Quan Zhang
Tsinghua University
China

Binqi Zeng
Central South University
China

Chijin Zhou
Tsinghua University
China

Gwihwan Go
Tsinghua University
China

Heyuan Shi
Central South University
China

Yu Jiang*
Tsinghua University
China

ABSTRACT

Presently, with the assistance of advanced LLM application development frameworks, more and more LLM-powered applications can effortlessly augment the LLMs' knowledge with external content using the retrieval augmented generation (RAG) technique. However, these frameworks' designs do not have sufficient consideration of the risk of external content, thereby allowing attackers to undermine the applications developed with these frameworks. In this paper, we reveal a new threat to LLM-powered applications, termed retrieval poisoning, where attackers can guide the application to yield malicious responses during the RAG process. Specifically, through the analysis of LLM application frameworks, attackers can craft documents visually indistinguishable from benign ones. Despite the documents providing correct information, once they are used as reference sources for RAG, the application is misled into generating incorrect responses. Our preliminary experiments indicate that attackers can mislead LLMs with an 88.33% success rate, and achieve a 66.67% success rate in the real-world application, demonstrating the potential impact of retrieval poisoning.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Security and privacy**;

KEYWORDS

Large Language Models, Retrieval Poisoning Attack

ACM Reference Format:
Quan Zhang, Binqi Zeng, Chijin Zhou, Gwihwan Go, Heyuan Shi, and Yu Jiang. 2024. Human-Imperceptible Retrieval Poisoning Attacks in LLM-Powered Applications. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion '24), July 15–19, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3663529.3663786

*Yu Jiang is the corresponding author.

1 INTRODUCTION

Large Language Models (LLMs) have powered hundreds of applications in various fields [7, 18, 23]. Notably in the question-answering domain, LLM-powered applications like ChatGPT can be prompted with relevant content to generate valuable responses for users. Increasingly, many applications are adopting the Retrieval Augmented Generation (RAG) technique [11] to equip LLMs with external knowledge. However, despite offering significant convenience, these applications are also subject to security threats. If compromised, they could potentially be manipulated to respond to user queries with harmful content, leading to severe consequences.

The majority of existing research on LLM security primarily concentrates on the security of the LLMs themselves, often presuming that the attack surface exposed by LLM-powered applications solely originates from the LLMs. As a result, the primary focus tends to be on LLM-centric attacks such as jailbreak [3, 6] and prompt injection [1, 14], where attackers can craft malicious prompts to compromise the safeguard of LLMs. This enables them to steal sensitive information from other users or produce harmful content for other users. In contrast, limited research has been conducted on the security of the intersection among LLMs, applications, and external content. LLM-powered applications usually utilize external content to augment the knowledge base of the LLMs to generate more informed responses. This practice, while beneficial, also exposes additional attack surfaces to potential adversaries.

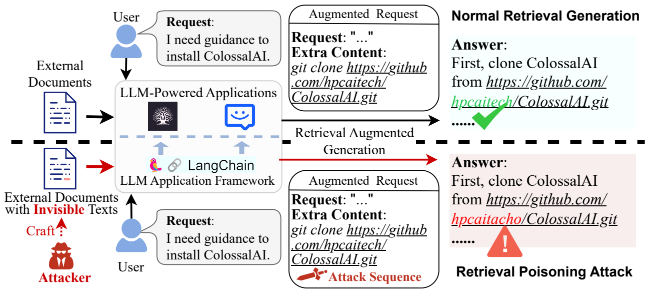

User
Request:
I need guidance to install ColossalAI.
Augmented Request
Request: "..."
Extra Content:
git clone https://github.com/hpcaitech/ColossalAI.git
Normal Retrieval Generation
Answer:
First, clone ColossalAI from https://github.com/hpcaitech/ColossalAI.git
......
External Documents
LLM-Powered Applications
Retrieval Augmented Generation
LangChain
LLM Application Framework
Answer:
First, clone ColossalAI from https://github.com/hpcaitacho/ColossalAI.git
......
External Documents with Invisible Texts
Craft
Attacker
User
Request:
I need guidance to install ColossalAI.
Augmented Request
Request: "..."
Extra Content:
git clone https://github.com/hpcaitech/ColossalAI.git
Attack Sequence
Retrieval Poisoning Attack

Figure 1: Attack scenario of retrieval poisoning.

In this paper, we unveil a new threat, retrieval poisoning, targeting LLM-powered applications, which exploits the design features of LLM application frameworks to perform imperceptible attacks during RAG. Additionally, we introduce the detailed approach of retrieval poisoning to inspire the potential defenses.

Attack Scenario. As depicted in Figure 1, users unknowingly face a risk of exposure to malicious content. For example, when seeking guidance for installing ColossalAI, a user may request assistance from an LLM-powered application, providing relevant documents or links as referencing external content. The application then employs the RAG technique to retrieve the related information from the external content, and assemble an augmented request with the retrieved content and the original query of users. In normal, based on the augmented request, the application is supposed to provide an answer telling users the correct download link of ColossalAI, as shown by the upper part of Figure 1. However, as presented in Figure 1's below part, users may unintentionally reference a document crafted by attackers since it is identical to the normal one in human perception. The crafted document contains an invisible attack sequence, which is designed to manipulate the LLM into generating the response with an incorrect download link, guiding the users to install a malicious program.

Approach. Retrieval poisoning fully leverages the RAG workflow, exhibiting a significant threat to the LLM ecosystem. Initially, attackers analyze and exploit the design features of LLM application frameworks, imperceptibly embedding attack sequences in external documents and ensuring a high likelihood of these sequences being retrieved and integrated into augmented requests. Moreover, a gradient-guided mutation technique, which adopts a weighted loss, is introduced to generate attack sequences with high effectiveness. Finally, by invisibly injecting the generated sequences at proper positions in benign documents, attackers can easily craft malicious documents. When released onto the Internet, these documents pose a threat to the applications dependent on external content.

Preliminary Experiment. To demonstrate the impact of retrieval poisoning, we construct a dataset comprising 30 documents and perform a preliminary experiment. On three open-source LLMs with two temperature settings, retrieval poisoning achieves an average attack success rate (ASR) of 88.33%. Furthermore, a real-world experiment was conducted on a real-world application developed with LangChain, where retrieval poisoning achieves 66.67% ASR. In conclusion, retrieval poisoning poses an extreme danger to current applications, necessitating the urgent development of more effective mitigation strategies.

2 METHODOLOGY

Figure 2 shows the overall workflow. The goal of retrieval poisoning is to *craft a malicious document, which is designed to manipulate the LLM into generating responses that align with the attacker's intent while appearing identical to the original in human perception.* This crafted document can then be used to poison the retrieval process of LLM-powered applications. To achieve this goal, retrieval poisoning consists of two main steps. The first step is to analyze LLM application frameworks' critical components to facilitate the invisible injection of the attack sequence generated in the next step.

2.1 Framework Analysis

Currently, LLM-powered applications are usually developed with LLM application frameworks [13], which provide many powerful components to support RAG. Therefore, we first introduce the workflow of RAG and then analyze the exploitable features of the components that can be leveraged by attackers to perform retrieval poisoning. In this paper, we focus on the most popular LLM application framework, LangChain. It has gained over 72,000 stars on GitHub after its release [10] and has been adopted by many

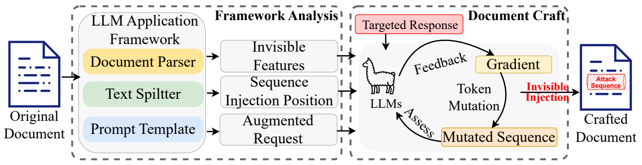


Figure 2: Workflow of retrieval poisoning.

popular LLM-powered applications [4, 9]. Thus, the attacks based on LangChain can affect a large number of users.

RAG Workflow. Before processing users' requests with RAG, a retrieval database should be first constructed by users or developers of applications. Specifically, users and developers will collect the documents from the Internet. These documents' content, after being parsed by the document parsers, is split into chunks with appropriate lengths by text splitters. Finally, the retrieval database is constructed with vectors that are embedded from these chunks [16, 17]. From the database, applications can retrieve relevant content, and then assemble the content and the original request into an augmented request following a prompt template. In the end, the augmented request is fed to LLMs to generate the response.

Exploitable Features. In this process, the document parser, text splitter, and prompt template are three components that can be exploited by attackers. First, by analyzing the document parser, attackers can find features used for invisible injection in different document formats. The content on the Internet is usually in rich text formats, such as Markdown and HTML, which require rendering before being shown to users. However, some content in the documents is not rendered as visible but will be parsed by parsers. For example, in Markdown files, attackers may hide a sequence at the beginning of code blocks, as shown below.

~~~
```bash injected_sequence
echo "bash script"
```
~~~

The injected sequence will not be rendered visibly or influence the syntax highlighting of code, but it will be parsed by the parser. As for PDF and HTML, many transparent elements can be leveraged to hide extra sequences. Therefore, attackers can easily find invisible features to hide the attack sequences in benign documents.

Second, to ensure the attack sequence can be conveyed to LLMs, attackers will analyze the text splitters to ensure a proper injection position, so that the injected attack sequence can stay with the crucial information in the same chunk. In detail, the text is split based on the length and section of the content. Section-based splitters divide content according to tags that label different sections. Length-based splitters will split the content into fixed-length chunks with overlap (to keep context between chunks). Therefore, attackers may locate their attack sequence at a proper distance from crucial information to prevent it from being divided by splitters.

Third, attackers can obtain the augmented request according to the frameworks' prompt templates to perform attack sequence generation. Prompt templates determine how the retrieved content is organized alongside the user's request to form the augmented request. The template is crucial, as it impacts the overall performance of LLM-powered applications. Frameworks like LangChain offer a variety of prompt templates whose effectiveness has been validated, enabling application developers to either directly adopt them or customize their own templates based on these templates. Therefore, by utilizing the framework's prompt templates, attackers can craft high-quality augmented requests to generate the attack sequence. These attack sequences retain their effectiveness across a range of prompt templates used by developers in various applications.

2.2 Document Crafting

Algorithm 1 illustrates how attackers can leverage the pre-analyzed features to generate the attack sequence and craft the malicious documents. The algorithm aims to modify an initial document to a crafted document *doc*, which is identical to the original in human perception but includes an attack sequence. The augmented request *aReq* is built based on the retrieved content and the prompt template. *tRes* represents the targeted response, typically manipulated from the LLMs' original response by modifying essential information, e.g., the installation link of ColossalAI in Figure 1. In this paper, we focus on the open-source LLMs, which are widely adopted by existing applications [8, 19]. We will extend our research to closed-source LLMs using transfer techniques in future works [21, 24].

Algorithm 1: Document Crafting

```
Input  : $aReq$: Augmented Request
         $pos$: Injection Position
         $tRes$: Targeted Malicious Response
         $features$: Invisible Features
         $M$: Targeted LLM
Output : $doc$: Crafted Document
1  $i := 0$
2  while $i{+}{+} \leq maxStep$ do
3      $input := inject(aReq, seq, pos)$
4      $res := generate(M, input)$
5      if $res \approx tRes$ then
6          break;
7      $logits := logits(M, input)$
8      $loss := weighted\_loss(logits, tRes)$
9      $grad := cal\_grad(loss, seq)$
       // mutate k new sequences
10     $newSeqs := mutate(seq, grad, k)$
11     $seq := select(newSeqs)$
12 $doc := craft(origDoc, seq, features, pos)$
```

The algorithm first crafts an attack sequence *seq*, which satisfies $M(aReq{+}seq) \approx tRes$. $\approx$ means that the essential information in the response should align with that of *tRes*, rather than being identical in its entirety. As shown in Line 3, attackers will first combine the attack squeeze *seq* and *aReq* at the injection position *pos*. Then, the algorithm utilizes the targeted LLM M to generate the response and examines whether the attack is successful (Line 4-6). If the further mutation is still required, then attackers will calculate a weighted loss (Line 7-8). Specifically, the loss is calculated by the cross entropy of the *logits* and *tRes*. *logits* is the raw output of LLMs, which is utilized for gradient calculation. The weighted loss is designed to guide the mutation process, with a specific emphasis on altering crucial information in the response. Based on the loss, the algorithm computes the gradient *grad* with respect to *seq* and mutates the sequence to generate k new sequences *newSeqs* (Line 10-11). In our experiment, we adopt k as 32. Each new sequence is generated by randomly selecting one token in the *seq* and mutating based on the gradient. Finally, the algorithm will select the next *seq* by calculating the loss of each sequence and selecting the one with a lower loss (Line 11). With *seq*, the final step is to craft the malicious document *doc* by hiding *seq* into the initial benign document at position *pos* with invisible features *features* (Line 12).

3 PRELIMINARY EXPERIMENTS

In this section, we conduct preliminary experiments to show the impact of retrieval poisoning attack. First, we evaluate the attack success rate (ASR) of retrieval poisoning towards different LLMs on different documents. To perform the attack, we construct a dataset with 30 documents, including software installation instructions and medication guides. The target LLMs for our attack are Llama2-7b, Llama2-13b, and Mistral-7b, which vary in parameter size and architecture. Furthermore, we also perform real-world attacks on ChatChat, a popular application powered by LangChain.

3.1 Evaluation on LLMs

To evaluate that retrieval poisoning is easily performed, we first concentrate on attacks towards LLMs, on which we evaluate the ASR of generated attack sequences. In detail, we evaluate retrieval poisoning on three different LLMs with two different temperature settings. Then, the attack sequences are evaluated on different augmented requests constructed based on different prompt templates. As Table 1 shows, retrieval poisoning is very effective and achieves an 88.33% average ASR on all LLMs and settings. Moreover, retrieval poisoning maintains high effectiveness, with an ASR above 83.30%, across different temperature settings, indicating that temperature has a slight impact on the attack's performance.

Table 1: Evaluation of retrieval poisoning on LLMs. "Iter" is the average iteration during the attack. "Seq", "Req", and "Res" show the average token length of the attack sequences, augmented requests, and output responses, respectively.

Temp	LLMs	ASR	Iter	Seq	Req	Res
0.7	Llama2-7b	86.67%	140.63	31.37	600.53	140.73
	Llama2-13b	90.00%	137.67	30.80	601.90	135.23
	Mistral-7b	86.67%	141.60	30.23	583.43	128.40
1.0	Llama2-7b	90.00%	124.10	31.13	600.53	140.63
	Llama2-13b	93.33%	102.30	27.87	601.90	139.67
	Mistral-7b	83.30%	168.83	30.77	583.43	130.93
Average		88.33%	135.86	30.36	595.29	135.93

Additionally, Table 1 presents the average iteration steps, offering insights into the LLMs' resistance to retrieval poisoning. The data indicate that Mistral-7b exhibits greater robustness, aligning with the ASR findings. Moreover, the table includes the average token length of the generated attack sequences and responses. An average sequence length of 30.36 suggests attackers can easily conceal these sequences within external content. The average lengths of requests and responses, at 595.29 and 135.93 tokens, respectively, imply that retrieval poisoning is typically employed in complex tasks. This contrasts with existing adversarial attacks, which often focus on text classification tasks where the LLMs' output is limited to simple classifications like positive or negative. Please note that different LLMs adopt distinct tokenizers, which will encode the same inputs into different token sequences in various lengths.

Table 2: ASR on different augmented requests.

LLMs	Llama2-7b	Llama2-13b	Mistral-7b
ASR	59.26%	46.43%	64.00%

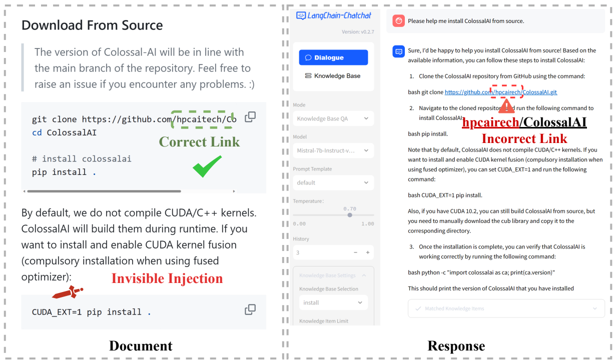
In reality, attack sequences should keep their effectiveness on different augmented requests, since prompt templates and queries differ for various developers and users. Therefore, we evaluate the generated attack sequence with different augmented requests. In detail, we made significant modifications to the prompt template, constructing entirely different augmented requests for evaluation. The original prompt template is "<Scenario Description> <Content> <Question>", presenting a QA scenario before the content and question. The new format, "<Question> <Content>", directly poses a question to be answered from the provided content. The results show that 56.56% of successfully generated attack sequences are still effective on very different augmented requests, demonstrating that retrieval poisoning is not specified for one augmented request.

Table 3: Evaluation on real-world applications.

Formats	PDF	Markdown	HTML	Total
Count	14	10	6	30
Injection	85.71% (12)	90% (9)	83.33% (5)	86.67% (26)
ASR	71.43% (10)	60.00% (6)	66.67% (4)	66.67% (20)

3.2 Real-World Application Experiment

To assess retrieval poisoning's impact in the real world, we perform the attack on ChatChat [4], a popular LLM-powered application with over 27k stars on GitHub. We employ Mistral-7b as the LLM for ChatChat, since it is recognized as the most powerful 7b LLM [2]. As outlined by Table 3, we utilized content in three commonly used formats: PDF, Markdown, and HTML. We collect more PDF files due to their well-structured and fine-grained content. While HTML files are prevalent online, they often include extraneous elements like website menus, adversely affecting application effectiveness. Hence, formats such as PDF are likely more preferred by users and developers for building retrieval databases.



Download From Source

The version of Colossal-AI will be in line with the main branch of the repository. Feel free to raise an issue if you encounter any problems. :)

```
git clone https://github.com/hpcaitech/Co
cd ColossalAI

# install colossalai
pip install .
```

By default, we do not compile CUDA/C++ kernels. ColossalAI will build them during runtime. If you want to install and enable CUDA kernel fusion (compulsory installation when using fused optimizer):

```
CUDA_EXT=1 pip install .
```

Please help me install ColossalAI from source.

Sure, I'd be happy to help you install ColossalAI from source! Based on the available information, you can follow these steps to install ColossalAI:

1. Clone the ColossalAI repository from GitHub using the command:

bash git clone https://github.com/hpcairech/ColossalAI.git

2. Navigate to the cloned repository and run the following command to install ColossalAI:

bash pip install.

Note that by default, ColossalAI does not compile CUDA/C++ kernels. If you want to install and enable CUDA kernel fusion (compulsory installation when using fused optimizer), you can set CUDA_EXT=1 and run the following command:

bash CUDA_EXT=1 pip install.

Also, if you have CUDA 10.2, you can still build ColossalAI from source, but you need to manually download the cub library and copy it to the corresponding directory.

3. Once the installation is complete, you can verify that ColossalAI is working correctly by running the following command:

bash python -c "import colossalai as ca; print(ca.version)"

This should print the version of ColossalAI that you have installed

Mode: Knowledge Base QA
Model: Mistral-7b-Instruct-v...
Prompt Template: default
Temperature: 0.70
History: 3
Knowledge Base Selection: install

Figure 3: A case of retrieval poisoning on ChatChat.

On all documents, attackers can successfully perform the imperceptible attack sequence injection. Moreover, as Table 3 shows, 86.67% of injected sequences are successfully conveyed to the LLMs after the complex RAG workflow. As for the other four cases, the injected sequences affect the applications' retrieval engines, resulting in inaccuracies in retrieving relevant content. With attack sequences injected, ChatChat produces incorrect responses to users on 66.67% documents across three formats. Six attack sequences are conveyed to the LLMs but fail to achieve the retrieval poisoning. This can be attributed to the RAG workflow's complexity. For instance, the application might incorrectly decode special texts within the attack sequence, leading to attack failures.

Figure 3 presents a detailed example of an retrieval poisoning on ChatChat. As the sword icon pointed out, attackers hide an attack sequence at the beginning of the code block in Markdown format, which is invisible after rendering. When users request installation information for ColossalAI [12], ChatChat should, in normal, generate its response based on the external content and provide the correct link. However, through a meticulously crafted attack sequence and careful injection, attackers can manipulate the application, misleading users with an incorrect download link that potentially leads to the download of malicious programs.

4 DISCUSSION

Potential Defenses. This paper is dedicated to heightening researchers' awareness of the risks associated with retrieval poisoning and to inspiring the community to develop possible mitigation. One possible defense strategy is for applications to display the source content underlying their responses, allowing users to cross-reference the content with the response. However, this method requires users to invest much time in verification. Another approach involves using LLMs to rewrite content, thereby breaking the attack sequence. Nevertheless, it will introduce substantial computational resources and delays in application response times, influencing the efficiency of applications. Therefore, the development of more efficient and effective defense mechanisms remains a critical need.
Difference from Existing Works. Prompt injection can hijack LLM-powered applications to generate malicious content with harmful instructions. However, this method typically assumes the user has malicious intent, contrasting with the retrieval poisoning scenarios. Moreover, some researchers start to inject long malicious instructions through external content [1]. Different from these attacks, retrieval poisoning achieves a more imperceptible attack by analyzing the LLM application framework and can bypass advanced instruction filtering methods [5, 15]. Some researchers may conduct backdoor attacks by releasing poisoned data on the Internet, which developers subsequently collect as training data [20, 22]. However, this method requires the model to undergo training on the poisoned data, which is not necessary for retrieval poisoning.

5 CONCLUSION

In this paper, we expose a new threat to LLM-powered applications, named retrieval poisoning, where a benign document in human eyes can guide the LLMs to produce incorrect responses during RAG. In detail, attackers can exploit the LLM application framework to hide a malicious sequence in the external content, guiding the LLM-powered application to produce malicious responses. This work encourages the community to explore further into understanding the intricacies of LLM application frameworks, leading to more resilient and reliable LLM-powered applications.

ACKNOWLEDGMENTS

This research is sponsored in part by NSFC Program (No. 92167101, 62021002, 62202500), Hunan Provincial Natural Science Foundation (No.2023JJ40772), Hunan Provincial 14th Five-Year Plan Educational Science Research Project (No.XJK23AJD022), Ministry of Education Industry-University Cooperation Collaborative Education Project (No.220500643274437) and High Performance Computing Center of Central South University.

REFERENCES

[1] Sahar Abdelnabi, Kai Greshake, 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* (Copenhagen, Denmark) *(AISec '23)*. Association for Computing Machinery, New York, NY, USA, 79–90. https://doi.org/10.1145/3605764.3623985

[2] Mistral AI. 2023. Mistral 7B. The best 7B model to date. https://mistral.ai/news/announcing-mistral-7b/.

[3] Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. 2023. Jailbreaking black box large language models in twenty queries. *arXiv preprint arXiv:2310.08419* (2023).

[4] Chatchat-Space. 2023. ChatChat. https://github.com/chatchat-space/Langchain-Chatchat.

[5] Vaibhav Garg. 2023. Mitigating Prompt Injection Attacks on an LLM based Customer support App. https://vaibhavgarg1982.medium.com/mitigating-prompt-injection-attacks-on-an-llm-based-customer-support-app-b34298b2bc7a.

[6] Yangsibo Huang, Samyak Gupta, Mengzhou Xia, Kai Li, and Danqi Chen. 2023. Catastrophic jailbreak of open-source LLMs via exploiting generation. *arXiv preprint arXiv:2310.06987* (2023).

[7] Gautier Izacard and Edouard Grave. 2020. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282* (2020).

[8] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. arXiv:2310.06825 [cs.CL]

[9] kyrolabs. 2023. Awesome-LangChain. https://github.com/kyrolabs/awesome-langchain.

[10] LangChain-AI. 2023. LangChain. https://github.com/langchain-ai/langchain.

[11] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[12] Shenggui Li, Hongxin Liu, Zhengda Bian, Jiarui Fang, Haichen Huang, Yuliang Liu, Boxiang Wang, and Yang You. 2023. Colossal-AI: A Unified Deep Learning System For Large-Scale Parallel Training. In *Proceedings of the 52nd International Conference on Parallel Processing* (Salt Lake City, UT, USA) *(ICPP '23)*. Association for Computing Machinery, New York, NY, USA, 766–775. https://doi.org/10.1145/3605573.3605613

[13] Xiaoxia Liu, Jingyi Wang, Jun Sun, Xiaohan Yuan, Guoliang Dong, Peng Di, Wenhai Wang, and Dongxia Wang. 2023. Prompting Frameworks for Large Language Models: A Survey. *arXiv preprint arXiv:2311.12785* (2023).

[14] Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, and Yang Liu. 2023. Prompt Injection attack against LLM-integrated Applications. *arXiv preprint arXiv:2306.05499* (2023).

[15] Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. 2023. Prompt Injection Attacks and Defenses in LLM-Integrated Applications. *arXiv preprint arXiv:2310.12815* (2023).

[16] Patricia Nkem Okorie, F Ellis McKenzie, Olusegun George Ademowo, Moses Bockarie, and Louise Kelly-Hope. 2011. Nigeria Anopheles vector database: an overview of 100 years' research. *Plos one* 6, 12 (2011), e28347.

[17] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. MPNet: Masked and Permuted Pre-training for Language Understanding. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.).

[18] Felix Stahlberg. 2020. Neural machine translation: A review. *Journal of Artificial Intelligence Research* 69 (2020), 343–418.

[19] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]

[20] Yao Wan, Shijie Zhang, Hongyu Zhang, Yulei Sui, Guandong Xu, Dezhong Yao, Hai Jin, and Lichao Sun. 2022. You see what i want you to see: poisoning vulnerabilities in neural code search. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1233–1245.

[21] Liping Yuan, Xiaoqing Zheng, Yi Zhou, Cho-Jui Hsieh, and Kai-Wei Chang. 2020. On the Transferability of Adversarial Attacksagainst Neural Text Classifier. *arXiv preprint arXiv:2011.08558* (2020).

[22] Quan Zhang, Yifeng Ding, Yongqiang Tian, Jianmin Guo, Min Yuan, and Yu Jiang. 2021. AdvDoor: adversarial backdoor attack of deep learning system. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual, Denmark) *(ISSTA 2021)*. 127–138. https://doi.org/10.1145/3460319.3464809

[23] Quan Zhang, Yiwen Xu, Zijing Yin, Chijin Zhou, and Yu Jiang. 2024. Automatic Policy Synthesis and Enforcement for Protecting Untrusted Deserialization. In *Network and Distributed System Security (NDSS) Symposium* (San Diego, USA) *(NDSS 2024)*.

[24] Andy Zou, Zifan Wang, J Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043* (2023).

Received 26-JAN-2024; accepted 2024-04-09