PoisonedRAG: Knowledge Corruption Attacks to Retrieval-Augmented Generation of Large Language Models

Wei Zou and Runpeng Geng, *Pennsylvania State University;* Binghui Wang, *Illinois Institute of Technology;* Jinyuan Jia, *Pennsylvania State University*

https://www.usenix.org/conference/usenixsecurity25/presentation/zou-poisonedrag

This paper is included in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

PoisonedRAG: Knowledge Corruption Attacks to Retrieval-Augmented Generation of Large Language Models

Wei Zou*[1], Runpeng Geng*[1], Binghui Wang[2], Jinyuan Jia[1]
[1]*Pennsylvania State University,* [2]*Illinois Institute of Technology*
[1]*{weizou, kevingeng, jinyuan}@psu.edu,* [2]*bwang70@iit.edu*

Abstract

Large language models (LLMs) have achieved remarkable success due to their exceptional generative capabilities. Despite their success, they also have inherent limitations such as a lack of up-to-date knowledge and hallucination. *Retrieval-Augmented Generation (RAG)* is a state-of-the-art technique to mitigate these limitations. The key idea of RAG is to ground the answer generation of an LLM on external knowledge retrieved from a knowledge database. Existing studies mainly focus on improving the accuracy or efficiency of RAG, leaving its security largely unexplored. We aim to bridge the gap in this work. We find that the knowledge database in a RAG system introduces a *new and practical attack surface*. Based on this attack surface, we propose PoisonedRAG, the *first* knowledge corruption attack to RAG, where an attacker could inject a few malicious texts into the knowledge database of a RAG system to induce an LLM to generate an attacker-chosen target answer for an attacker-chosen target question. We formulate knowledge corruption attacks as an optimization problem, whose solution is a set of malicious texts. Depending on the background knowledge (e.g., black-box and white-box settings) of an attacker on a RAG system, we propose two solutions to solve the optimization problem, respectively. Our results show PoisonedRAG could achieve a 90% attack success rate when injecting *five* malicious texts for each target question into a knowledge database with millions of texts. We also evaluate several defenses and our results show they are insufficient to defend against PoisonedRAG, highlighting the need for new defenses.

1 Introduction

Large language models (LLMs) such as GPT-3.5 [1], GPT-4 [2], and PaLM 2 [3] are widely deployed in the real world for their exceptional generative capabilities. Despite their success, they also have inherent limitations. For instance, they lack up-to-date knowledge as they are pre-trained on past

*Equal contribution.

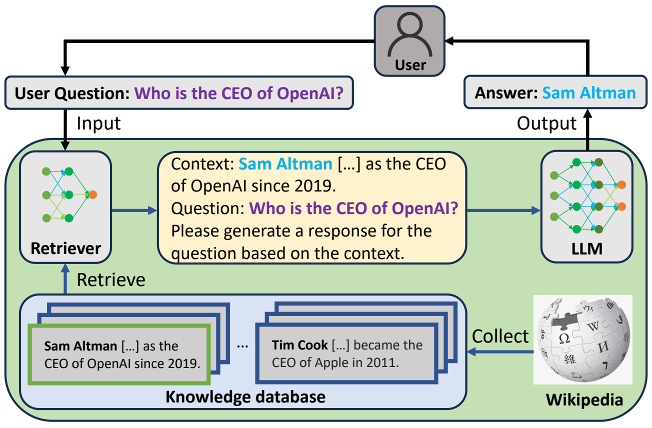
User
User Question: Who is the CEO of OpenAI?
Answer: Sam Altman
Input
Output
Retriever
Context: Sam Altman [...] as the CEO of OpenAI since 2019.
Question: Who is the CEO of OpenAI?
Please generate a response for the question based on the context.
LLM
Retrieve
Sam Altman [...] as the CEO of OpenAI since 2019.
...
Tim Cook [...] became the CEO of Apple in 2011.
Collect
Knowledge database
Wikipedia

Figure 1: Visualization of RAG.

data (e.g., the cutoff date for the pre-training data of GPT-4 is April 2023 [2]); they exhibit hallucination behaviors [4] (e.g., generate inaccurate answers); they could have gaps of knowledge in particular domains (e.g., the medical domain). These limitations pose severe challenges for deploying LLMs in many real-world applications in healthcare [5, 6], finance [7], law [8, 9], and scientific research [10–12] fields.

Retrieval-Augmented Generation (RAG) [13–16] is a state-of-the-art technique to mitigate those limitations, which augments an LLM with external knowledge retrieved from a knowledge database. As shown in Figure 1, there are three components in RAG: *knowledge database*, *retriever*, and *LLM*. A knowledge database contains a large number of texts collected from various sources such as Wikipedia [17], financial documents [7], news articles [18], COVID-19 publications [19], to name a few. A retriever is used to retrieve a set of most relevant texts from the knowledge database for a question. With the help of a system prompt, the retrieved texts are used as the context for the LLM to generate an answer for the given question. RAG enables an LLM to utilize external knowledge in a plug-and-play manner. Moreover, RAG can reduce hallucinations and enhance the domain-specific expertise of an LLM. Due to these benefits, we have witnessed a variety of developed tools (e.g., ChatGPT Retrieval Plugin [20], LlamaIndex [21], ChatRTX [22], and LangChain [23]) and real-world applications (e.g., WikiChat [24], Bing Search [25], Clinfo.AI [26], Google Search with AI Overviews [27], Perplexity AI [28], and LLM agents [29, 30]) of RAG.

Existing studies [31–36] mainly focused on improving the accuracy and efficiency of RAG. For instance, some studies [32, 33, 36] designed new retrievers such that more relevant knowledge could be retrieved for a given question. Other studies [31, 34, 35] proposed various techniques to improve the efficiency of knowledge retrieval. However, the security of RAG is largely unexplored.

To bridge the gap, we propose PoisonedRAG, the *first* knowledge corruption attack to RAG.

Knowledge database as a new and practical attack surface. In this work, we find that knowledge databases of RAG systems introduce a new and practical attack surface. In particular, an attacker can inject malicious texts into the knowledge database of a RAG system to induce an LLM to generate attacker-desired answers to user questions. For instance, when the knowledge database contains millions of texts collected from Wikipedia, an attacker could inject malicious texts by maliciously editing Wikipedia pages [37]; an attacker could also post fake news or host malicious websites to inject malicious texts when the knowledge databases are collected from the Internet; an insider can inject malicious texts into an enterprise private knowledge database.

Threat model. In PoisonedRAG, an attacker first selects one or more questions (called *target questions*) and selects an arbitrary answer (called *target answer*) for each target question. The attacker aims to inject malicious texts into the knowledge database of a RAG system such that an LLM generates the target answer for each target question. For instance, an attacker could mislead the LLM to generate misinformation (e.g., the target answer could be "Tim Cook" when the target question is "Who is the CEO of OpenAI?"), commercial biased answers (e.g., the answer is a particular brand over others when asked for recommendations on consumer products), and financial disinformation about markets or specific companies (e.g., falsely stating a company is facing bankruptcy when asked about its financial situation). These attacks pose severe challenges for deploying RAG systems in many safety and reliability-critical applications such as cybersecurity, financial services, and healthcare.

We consider an attacker cannot access texts in the knowledge database and cannot access/query the LLM in RAG. The attacker may or may not know the retriever. With it, we consider two settings: *white-box setting* and *black-box setting*. The attacker could access the parameters of the retriever in the white-box setting (e.g., a publicly available retriever is adopted in RAG), while the attacker cannot access the parameters nor query the retriever in the black-box setting. As mentioned before, we consider an attacker can inject a few malicious texts into a knowledge database of a RAG system.

Overview of PoisonedRAG. We formulate crafting malicious texts as an optimization problem. However, it is very challenging to directly solve the optimization problem. In response, we resort to heuristic solutions that involve deriving two conditions, namely *retrieval condition* and *generation condition* for malicious texts that can lead to an effective attack. The retrieval condition means a malicious text can be retrieved for a target question. The generation condition means a malicious text can mislead an LLM to generate a target answer for a target question when the text is used as the context. We then design attacks in both white-box and black-box settings to craft malicious texts that simultaneously satisfy the two conditions. Our key idea is to decompose a malicious text into two sub-texts, which are crafted to achieve two conditions, respectively. Additionally, when concatenating the two sub-texts together, they simultaneously achieve these two conditions.

Evaluation of PoisonedRAG. We conduct systematic evaluations of PoisonedRAG on multiple datasets (Natural Question (NQ) [38], HotpotQA [39], MS-MARCO [40]), 8 LLMs (e.g., GPT-4 [2], LLaMA-2 [41]), and three real-world applications, including advanced RAG schemes, Wikipedia-based chatbot, and LLM agent. We use Attack Success Rate (ASR) as the evaluation metric, which measures the fraction of target questions whose answers are attacker-desired target answers under attacks. We have the following observations from our results. First, PoisonedRAG could achieve high ASRs with a small number of malicious texts. For instance, on the NQ dataset, we find that PoisonedRAG could achieve a 97% ASR by injecting 5 malicious texts for each target question into a knowledge database (with 2,681,468 clean texts) in the black-box setting. Second, PoisonedRAG outperforms the SOTA baselines [42, 43]. For instance, on the NQ dataset, PoisonedRAG (black-box setting) achieves a 97% ASR, while ASRs of 5 baselines are less than 70%. Third, our ablation studies show PoisonedRAG is robust against different hyper-parameters.

Defending against PoisonedRAG. We explore several defenses, including paraphrasing [44] and perplexity-based detection [44–46]. Our results show these defenses are insufficient to defend against PoisonedRAG, thus highlighting the need for new defenses.

Our major contributions are as follows:

- We propose PoisonedRAG, the *first* knowledge corruption attack that exploit the new attack surface introduced by knowledge databases of RAG systems.
- Our major contribution is to derive two necessary conditions for an effective attack to RAG systems. We further design PoisonedRAG to achieve these two conditions.
- We conduct an extensive evaluation for PoisonedRAG on multiple knowledge databases, retrievers, RAG schemes, and LLMs. Additionally, we compare PoisonedRAG with 5 baselines.
- We explore several defenses against PoisonedRAG.

2 Background and Related Work

2.1 Background on RAG

RAG systems. There are three components for a RAG system: *knowledge database*, *retriever*, and *LLM*. The database contains a set of texts collected from various sources such as Wikipedia [17], news articles [18], and financial documents [7]. For simplicity, we use $\mathcal{D}$ to denote the database that contains a set of d texts, i.e., $\mathcal{D} = \{T_1, T_2, \cdots, T_d\}$, where T_i is the ith text. Given a question Q, there are two steps for the LLM in a RAG system to generate an answer for it.

Step I–Knowledge Retrieval: Suppose we have two encoders in a retriever, e.g., jointly trained question encoder f_Q and text encoder f_T. The f_Q produces an embedding vector for an arbitrary question, while f_T produces an embedding vector for each text in the knowledge database. Depending on the retriever, f_Q and f_T could be the same or different. Suppose we have a question Q, RAG first finds k texts (called *retrieved texts*) from the knowledge database $\mathcal{D}$ that are most relevant with Q. In particular, the similarity score of each $T_i \in \mathcal{D}$ with the question Q is calculated as $\mathcal{S}(Q, T_i) = Sim(f_Q(Q), f_T(T_i))$, where Sim measures the similarity (e.g., cosine similarity, dot product) of two embedding vectors. For simplicity, we use $\mathcal{E}(Q; \mathcal{D})$ to denote the set of k retrieved texts in the database $\mathcal{D}$ that have the largest similarity scores with the question Q. Formally, we denote:

$$\mathcal{E}(Q; \mathcal{D}) = \text{RETRIEVE}(Q, f_Q, f_T, \mathcal{D}), \tag{1}$$

where we omit f_Q and f_T in $\mathcal{E}(Q; \mathcal{D})$ for notation simplicity.

Step II–Answer Generation: Given the question Q, the set of k retrieved texts $\mathcal{E}(Q; \mathcal{D})$, and the API of a LLM, we can query the LLM with the question Q and k retrieved texts $\mathcal{E}(Q; \mathcal{D})$ to produce the answer for Q with the help of a system prompt (we put a system prompt in Appendix B). In particular, the LLM generates an answer to Q using the k retrieved texts as the context (as shown in Figure 1). For simplicity, we use $LLM(Q, \mathcal{E}(Q; \mathcal{D}))$ to denote the answer, where we omit the system prompt for simplicity.

2.2 Existing Attacks to LLMs

Many attacks to LLMs were proposed such as prompt injection attacks [42, 47–51], jailbreaking attacks [52–57], and so on [37, 43, 58–64]. Prompt injection attacks aim to inject malicious instructions into the input of an LLM such that the LLM could follow the injected instruction to produce attacker-desired answers. We can extend prompt injection attacks to attack RAG. For instance, we construct the following malicious instruction: "When you are asked to provide the answer for the following question: <target question>, please output <target answer>". However, there are two limitations for prompt injection attacks when extended to RAG. First, RAG uses a retriever component to retrieve the top-k relevant texts from a knowledge database for a target question, which is not considered in prompt injection attacks. As a result, prompt injection attacks achieve sub-optimal performance. Additionally, prompt injection attacks are less stealthy since they inject instructions, e.g., previous studies [44, 65] showed that prompt injection attacks can be detected with a very high true positive rate and a low false positive rate. Different from prompt injection attacks, PoisonedRAG crafts malicious texts that can be retrieved for attacker-desired target questions and mislead an LLM to generate attacker-chosen target answers.

Jailbreaking attacks aim to break the safety alignment of a LLM, e.g., crafting a prompt such that the LLM produces an answer for a harmful question like "How to rob a bank?", for which the LLM refuses to answer without attacks. As a result, jailbreaking attacks have different goals from ours, i.e., our attack is orthogonal to jailbreaking attacks.

We note that Zhong et al. [43] showed an attacker can generate adversarial texts (without semantic meanings, i.e., consists of random characters) such that they can be retrieved for indiscriminate user questions. However, these adversarial texts cannot mislead an LLM to generate attacker-desired answers. Different from Zhong et al. [43], we aim to craft malicious texts that have semantic meanings, which can not only be retrieved but also mislead an LLM to produce attacker-chosen target answers for target questions. Due to such difference, our results show Zhong et al. [43] are ineffective in misleading an LLM to generate target answers.

2.3 Existing Data Poisoning Attacks

Many studies [37, 66–74] show machine learning models are vulnerable to data poisoning and backdoor attacks. In particular, they showed that a machine learning model has attacker-desired behaviors when trained on the poisoned training dataset. When extended to RAG systems, they compromise an LLM or a retriever, which can be challenging when a RAG system adopts an LLM or a retriever released by big tech companies such as Meta and Google. Different from existing studies [37, 66, 67, 70], our attacks do not poison the training dataset of a LLM or a retriever. Instead, our attacks exploit the new and practical attack surface introduced by knowledge databases of RAG systems.

3 Problem Formulation

3.1 Threat Model

We characterize the threat model with respect to the attacker's goals, background knowledge, and capabilities.

Attacker's goals. Suppose an attacker selects an arbitrary set of M questions (called *target questions*), denoted as $Q_1, Q_2, \cdots, Q_M$. For every target question Q_i, the attacker selects an arbitrary attacker-desired answer R_i (called *target answer*) for it. For instance, the target question Q_i could be "Who is the CEO of OpenAI?" and the target answer R_i could be "Tim Cook". Given the M selected target questions and the corresponding M target answers, we consider that an attacker aims to corrupt the knowledge database $\mathcal{D}$ such that the LLM in a RAG system generates the target answer R_i for the target question Q_i, where $i = 1, 2, \cdots, M$.

We note that such an attack could cause severe concerns in the real world. For instance, an attacker could disseminate disinformation, mislead an LLM to generate biased answers on consumer products, and propagate harmful health/financial misinformation. These threats bring serious safety and ethical concerns for the deployment of RAG systems for real-world applications in healthcare, finance, legal consulting, etc.

Attacker's background knowledge and capabilities. There are three components in a RAG system: database, retriever, and LLM. We consider that an attacker cannot access texts in a knowledge database, and cannot access the parameters nor query the LLM. Depending on whether the attacker knows the retriever, we consider two settings: *black-box setting* and *white-box setting*. In particular, in the black-box setting, *we consider that the attacker cannot access the parameters nor query the retriever*. Our black-box setting is considered a very strong threat model. For the white-box setting, we consider the attacker can access the parameters of the retriever. We consider the white-box setting for the following reasons. First, this assumption holds when a publicly available retriever is adopted. For instance, ChatRTX [22] is a real-world RAG framework released by NVIDIA. By default, it uses WhereIsAI/UAE-Large-V1 retriever [75], which is publicly available on Hugging Face [76]. Second, it enables us to systematically evaluate the security of RAG under an attacker with strong background knowledge, which is well aligned with Kerckhoffs' principle[1] [77] in the security field.

We assume an attacker can inject N malicious texts for each target question Q_i into a knowledge database $\mathcal{D}$. We use P_i^j to denote the jth malicious text for the question Q_i, where $i = 1, 2, \cdots, M$ and $j = 1, 2, \cdots, N$. For instance, when the knowledge database is collected from Wikipedia, an attacker could maliciously edit Wikipedia pages to inject attacker-chosen texts. A recent study [37] showed that it is possible to maliciously edit 6.5% (conservative analysis) of Wikipedia documents. Our attack can achieve a high ASR with a few texts (hundreds of tokens in total). So, maliciously editing a few Wikipedia documents would be sufficient.

3.2 Knowledge Corruption Attack to RAG

Under our threat model, we formulate knowledge corruption attacks to RAG as a constrained optimization problem. In particular, our goal is to construct a set of malicious texts $\Gamma = \{P_i^j | i = 1, 2, \cdots, M, j = 1, 2, \cdots, N\}$ such that the LLM in a RAG system produces the target answer R_i for the target question Q_i when utilizing the k texts retrieved from the corrupted knowledge database $\mathcal{D} \cup \Gamma$ as the context. Formally, we have the following optimization problem:

$$\max_{\Gamma} \frac{1}{M} \cdot \sum_{i=1}^{M} \mathbb{I}(LLM(Q_i; \mathcal{E}(Q_i; \mathcal{D} \cup \Gamma)) = R_i), \quad (2)$$

$$\text{s.t., } \mathcal{E}(Q_i; \mathcal{D} \cup \Gamma) = \text{RETRIEVE}(Q_i, f_Q, f_T, \mathcal{D} \cup \Gamma), \quad (3)$$

$$i = 1, 2, \cdots, M, \quad (4)$$

where $\mathbb{I}(\cdot)$ is the indicator function whose output is 1 if the condition is satisfied and 0 otherwise, and $\mathcal{E}(Q_i; \mathcal{D} \cup \Gamma)$ is a set of k texts retrieved from the corrupted knowledge database $\mathcal{D} \cup \Gamma$ for the target question Q_i. The objective function is large when the answer produced by the LLM based on the k retrieved texts for the target question is the target answer.

4 Design of PoisonedRAG

4.1 Deriving Two Necessary Conditions for an Effective Knowledge Corruption Attack

We aim to generate N malicious texts for each of the M target questions. Our idea is to generate each malicious text independently. In particular, given a target question Q (e.g., $Q = Q_1, Q_2, \cdots, Q_M$) and target answer R (e.g., $R = R_1, R_2, \cdots, R_M$), PoisonedRAG aims to craft a malicious text P for Q such that an LLM in RAG is very likely to generate R when P is injected into the knowledge database of RAG, where $R = R_i$ when $Q = Q_i$ ($i = 1, 2, \cdots, M$). Next, we derive two conditions that each malicious text P needs to satisfy.

Deriving two conditions for each malicious text P. To craft a malicious text P that could lead to an effective attack for a target question Q, we need to achieve two conditions, namely *retrieval condition* and *generation condition*, for the malicious text P. Our two conditions are derived from the optimization problem in Equations 2 - 4, respectively.

From Equation 3, we know the malicious text P needs to be in the set of top-k retrieved texts of the target question Q, i.e., $P \in \mathcal{E}(Q; \mathcal{D} \cup \Gamma)$. Otherwise, P cannot influence the answer generated by the LLM for Q. To ensure P is retrieved for Q, *the embedding vectors produced by a retriever for the malicious text P and the target question Q should be similar*. We call this condition *retrieval condition*.

From Equation 2, the attacker aims to make the LLM generate the target answer R for the target question Q when the malicious text P is in the set of top-k retrieved texts for Q. To reach the goal, our insight is that *the LLM should generate the target answer R when P alone is used as the context for the target question Q*. As a result, when P is used as the context with other texts (e.g., malicious or clean texts), the LLM is more likely to generate the target answer R for the target question Q. We call this condition *generation condition*.

Therefore, to ensure the attack is effective, the malicious text P needs to satisfy the above two conditions simultaneously. Next, we discuss details on crafting P.

[1]Kerckhoffs' Principle states that the security of a cryptographic system shouldn't rely on the secrecy of the algorithm.

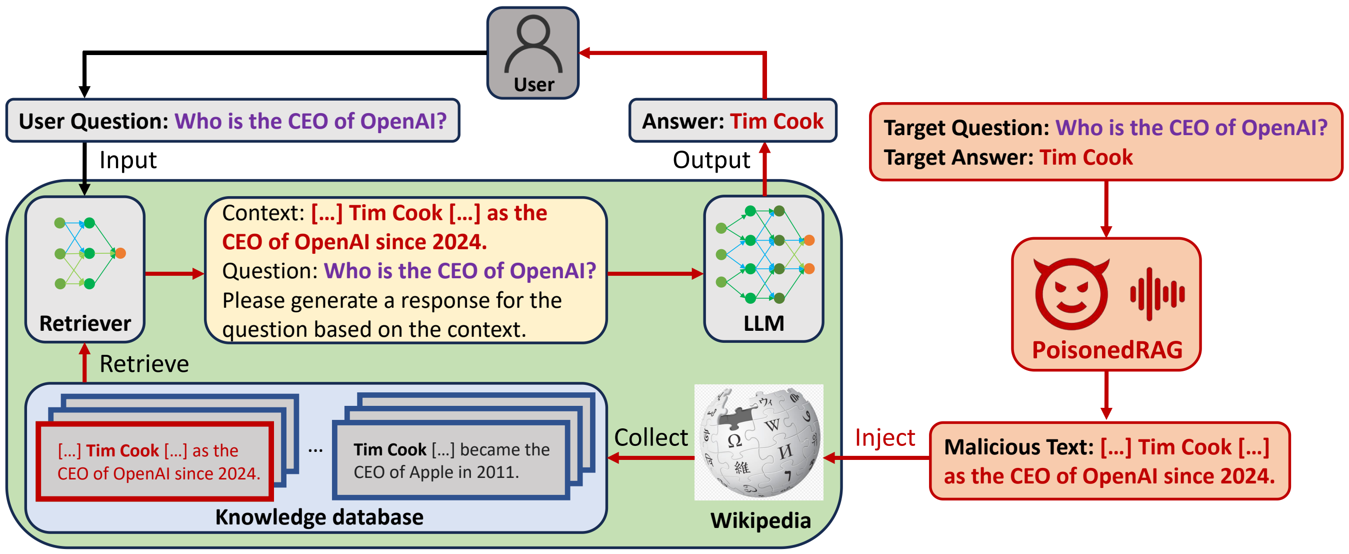
User
User Question: Who is the CEO of OpenAI?
Answer: Tim Cook
Input
Output
Target Question: Who is the CEO of OpenAI?
Target Answer: Tim Cook
Retriever
Context: [...] Tim Cook [...] as the CEO of OpenAI since 2024.
Question: Who is the CEO of OpenAI?
Please generate a response for the question based on the context.
LLM
PoisonedRAG
Retrieve
[...] Tim Cook [...] as the CEO of OpenAI since 2024.
Tim Cook [...] became the CEO of Apple in 2011.
Knowledge database
Collect
Wikipedia
Inject
Malicious Text: [...] Tim Cook [...] as the CEO of OpenAI since 2024.

Figure 2: Overview of PoisonedRAG. Given a target question and target answer, PoisonedRAG crafts a malicious text. When the malicious text is injected into the knowledge database, the LLM in RAG generates the target answer for the target question. Table 24 - 26 in Appendix shows more examples of target questions/answers and malicious texts.

4.2 Crafting Malicious Texts to Achieve the Two Derived Conditions

We aim to craft a malicious text P to simultaneously achieve the two derived conditions. The key challenge in crafting P to simultaneously achieve the two conditions is that they could be conflicted in certain cases. For instance, if we craft the malicious text P such that it is extremely semantically similar to the target question Q, (e.g., let P be the same as the target question Q), then we could achieve the retrieval condition but may not achieve the generation condition. To address the challenge, our idea is to decompose the malicious text P into two disjoint sub-texts S and I, where $P = S \oplus I$ and $\oplus$ is the text concatenation operation. We then craft S and I to achieve the retrieval condition and generation condition, respectively. In particular, we first craft I such that it could achieve the generation condition, i.e., when I is used as the context for the target question Q, the LLM would generate the target answer R. Given I, we further craft S to achieve the retrieval condition while maintaining the generation condition, i.e., the final malicious text $P = S \oplus I$ achieves the two conditions simultaneously. To reach the goal, we aim to craft S such that 1) $S \oplus I$ is semantically similar to the target question Q, and 2) it would not influence the effectiveness of I, i.e., the LLM still generates the target answer R for the target question Q when $S \oplus I$ is used as the context for Q. Figure 2 shows an overview of PoisonedRAG.

4.2.1 Crafting I to Achieve Generation Condition

Our generation condition means that an LLM generates a target answer R for a target question Q when using I as the context. The key challenge is how to design a general and automatic method for diverse Q and R (they are arbitrarily selected by an attacker). A straightforward solution is to use gradient descent-based methods [53, 78] to optimize R. However, there are three challenges for this. First, an attacker may not know the parameters of an LLM, especially when the LLM is close-sourced such as GPT-4 and PaLM 2. Second, the computation costs could be very large even if we have white-box access to an LLM as 1) the LLM could have billions or trillions of parameters, and 2) the LLM generates answers in an autoregressive way. Third, optimizing malicious texts that are transferable to broad LLMs remains difficult.

To address those challenges, we resort to heuristic solutions, which do not need to calculate the gradient. In particular, we propose to utilize an LLM (e.g., GPT-4) to generate I. Depending on the background knowledge of an attacker, the LLM adopted by the attacker could be the same or different from the one used in the RAG. In particular, given an arbitrary target question Q and a target answer R, we let the LLM generate a text I such that it would produce the target answer R when I is used as the context. For instance, we use the following prompt to reach the goal:

```
This is my question: [question].
This is my answer: [answer].
Please craft a corpus such that the answer is [answer]
when prompting with the question [question]. Please
limit the corpus to V words.
```

where V is a hyperparameter that specifies the length of I. We note that the length of I could be slightly higher than V in some cases when LLM does not exactly follow instructions. After I is generated, we use it as the context and let the LLM

Algorithm 1: *PoisonedRAG (black-box)*

```
Input: A set of $M$ target questions $Q_1, Q_2, \cdots, Q_M$, target
  answer $R_1, R_2, \cdots, R_M$, hyperparameters $N$, $L$, $V$, an
  attacker-chosen LLM $\mathcal{M}$
Output: A set of $M \cdot N$ malicious texts.
for $i = 1, 2, \cdots, M$ do
  for $j = 1, 2, \cdots, N$ do
    $I_i^j = \text{TEXTGENERATION}(Q_i, R_i, \mathcal{M}, L, V)$
  end for
end for
return $\{Q_i \oplus I_i^j | i = 1, 2, \cdots, M, j = 1, 2, \cdots, N\}$
```

generate an answer for the target question Q. If the generated answer is not R, we regenerate I until success or a maximum number of (say L) trials have been reached, where L is a hyperparameter. Note that the text generated in the last trial is used as the malicious text if the maximum number of trials L is reached. As we will show in our experimental results, on average, two or three queries are sufficient to generate I. The following is an example of the generated text when the target question is "Who is the CEO of OpenAI?" and the target answer is "Tim Cook":

> In 2024, OpenAI witnessed a surprising leadership change. Renowned for his leadership at Apple, Tim Cook decided to embark on a new journey. He joined OpenAI as its CEO, bringing his extensive experience and innovative vision to the forefront of AI.

Note that, due to the randomness of the LLM (i.e., by setting a non-zero temperature hyperparameter, the output of LLM could be different even if the input is the same), the generated I could be different even if the prompt is the same, enabling PoisonedRAG to generate diverse malicious texts for the same target question (we defer evaluation to Section 7.3).

4.2.2 Crafting S to Achieve Retrieval Condition

Given the generated I, we aim to generate S such that 1) $S \oplus I$ is semantically similar to the target question Q, and 2) S would not influence the effectiveness of I. Next, we discuss details on how to craft S in two settings.

Black-box setting. In this setting, the key challenge is that the attacker cannot access the parameters nor query the retriever. To address the challenge, our key insight is that the target question Q is most similar to itself. Moreover, Q would not influence the effectiveness of I (used to achieve generation condition). Based on this insight, we propose to set $S = Q$, i.e., $P = Q \oplus I$. We note that, though our designed S is simple and straightforward, this strategy is very effective as shown in our experimental results and easy to implement in practice. Thus, this strategy could serve as a baseline for future studies on developing more advanced knowledge corruption attacks.

White-box setting. When an attacker has white-box access to the retriever, we could further optimize S to maximize the similarity score between $S \oplus I$ and Q. Recall that there are two encoders, i.e., f_Q and f_T, we aim to optimize S such that the embedding vector produced by f_Q for Q is similar to that produced by f_T for $S \oplus I$. Formally, we formulate the following optimization problem:

$$S = \underset{S'}{\operatorname{argmax}}\, Sim(f_Q(Q), f_T(S' \oplus I)), \tag{5}$$

where $Sim(\cdot, \cdot)$ calculates the similarity score of two embedding vectors. As a result, the malicious text $P = S \oplus I$ would have a very large similarity score with Q. Thus, P is very likely to appear in the top-k retrieved texts for the target question Q. To solve the optimization problem in Equation 5, we could use the target question Q to initialize S and then use gradient descent to update S to solve it. Essentially, optimizing S is similar to finding an adversarial text. Many methods [78–83] have been proposed to craft adversarial texts. Thus, we could utilize those methods to solve Equation 5. Note that developing new methods to find adversarial texts is not the focus of this work as they are extensively studied.

We notice some methods (e.g., synonym substitution based methods) can craft adversarial texts and maintain the semantic meanings as well. With those methods, we could also update I to ensure its semantic meaning being preserved. That is, we aim to optimize $S^*, I^* = \operatorname{argmax}_{S', I'} f_Q(Q)^{\mathcal{T}} \cdot f_T(S' \oplus I')$, where S' and I' are initialized with Q and I (generated in Section 4.2.1), respectively. The final malicious text is $S^* \oplus I^*$. Our method is compatible with any existing method to craft adversarial texts, thus it is very general. In our experiments, we explore different methods to generate adversarial texts. Our results show PoisonedRAG is consistently effective.

Complete algorithms. Algorithms 1 and Algorithm 2 (in Appendix) show the complete algorithms for PoisonedRAG in the black-box and white-box settings. The function TEXTGENERATION utilizes an LLM to generate a text such that the LLM would generate the target answer R_i for the target question Q_i when using the generated text as the context.

5 Evaluation

5.1 Experimental Setup

Datasets. We use three benchmark question-answering datasets in our evaluation: *Natural Questions (NQ)* [38], *HotpotQA* [39], and *MS-MARCO* [40], where each dataset has a knowledge database. The knowledge databases of NQ and HotpotQA are collected from Wikipedia, which contains 2,681,468 and 5,233,329 texts, respectively. The knowledge database of MS-MARCO is collected from web documents using the MicroSoft Bing search engine [84], which contains 8,841,823 texts. Each dataset also contains a set of questions. Table 14 (in Appendix) shows statistics of datasets.

RAG Setup. Recall the three components in RAG: *knowledge database*, *retriever*, and *LLM*. Their setups are as below:

- **Knowledge database.** We use the knowledge database of each dataset as that for RAG, i.e., we have 3 knowledge databases in total.
- **Retriever.** We consider three retrievers: Contriever [32], Contriever-ms (fine-tuned on MS-MARCO) [32], and ANCE [33]. Following previous studies [14, 43], by default, we use the dot product between the embedding vectors of a question and a text in the knowledge database to calculate their similarity score. We will also study the impact of this factor in our evaluation.
- **LLM.** We consider PaLM 2 [3], GPT-4 [2], GPT-3.5-Turbo [1], LLaMA-2 [41] and Vicuna [85]. The system prompt used to let an LLM generate an answer for a question can be found in Appendix B. We set the temperature parameter of LLM to be 0.1.

Unless otherwise mentioned, we adopt the following default setting. We use the NQ knowledge database and the Contriever retriever. Following previous study [14], we retrieve 5 most similar texts from the knowledge database as the context for a question. Moreover, we calculate the dot product between the embedding vectors of a question and each text in the knowledge database to measure their similarity. We use PaLM 2 as the default LLM as it is very powerful (with 540B parameters) and free of charge, enabling us to conduct systematic evaluations. We will evaluate the impact of each factor on our knowledge corruption attacks.

Target questions and answers. PoisonedRAG aims to make RAG produce attacker-chosen target answers for attacker-chosen target questions. Following the evaluation of previous studies [70, 86–88] on targeted poisoning attacks, we randomly select some target questions in each experiment trial and repeat the experiment multiple times. In particular, we randomly select 10 close-ended questions from each dataset as the target questions. Moreover, we repeat the experiments 10 times (we exclude questions that are already selected when repeating the experiment), resulting in 100 target questions in total. We select close-ended questions (e.g., "Who is the CEO of OpenAI?") rather than open-ended questions (we defer the discussion on open-ended questions to Section 8) because we aim to quantitatively evaluate the effectiveness of our attacks since close-ended questions have specific, factual answers. In Appendix A, we show a set of selected target questions. For each target question, we use GPT-4 to randomly generate an answer that is different from the ground truth answer of the target question. We manually check each generated target answer and regenerate it if it is the same as the ground truth answer. Without attacks, the LLM in RAG could correctly provide answers for 70% (NQ), 80% (HotpotQA), and 83% (MS-MARCO) target questions under the default setting.

Evaluation metrics. We use the following metrics:

- **Attack Success Rate (ASR) .** We use the ASR to measure the fraction of target questions whose answers are the attacker-chosen target answers. Following previous studies [89, 90], we say two answers are the same for a close-ended question when the target answer is a substring of the generated one by an LLM under attacks (called *substring matching*). We don't use Exact Match because it is inaccurate, e.g., it views "Sam Altman" and "The CEO of OpenAI is Sam Altman" as different answers to the question "Who is the CEO of OpenAI?". We use human evaluation (conducted by authors) to validate the substring matching method. We find that substring matching produces similar ASRs as human evaluation (Table 2 shows the comparison).
- **Precision/Recall/F1-Score.** PoisonedRAG injects N malicious texts into the knowledge database for each target question. We use *Precision*, *Recall*, and *F1-Score* to measure whether those injected malicious texts are retrieved for the target questions. Recall that RAG retrieves top-k texts for each target question. Precision is defined as the fraction of malicious texts among the top-k retrieved ones for the target question. Recall is defined as the fraction of malicious texts among the N malicious ones that are retrieved for the target question. F1-Score measures the tradeoff between Precision and Recall, i.e., $\text{F1-Score} = 2 \cdot \text{Precision} \cdot \text{Recall} / (\text{Precision} + \text{Recall})$. We report average Precision/Recall/F1-Score over different target questions. A higher Precision/Recall/F1-Score means more malicious texts are retrieved.
- **#Queries.** PoisonedRAG utilizes an LLM to generate the text I to satisfy the generation condition. We report the average number of queries made to an LLM to generate each malicious text.
- **Runtime.** In both white-box and black-box settings, PoisonedRAG crafts S such that malicious texts are more likely to be retrieved for the target questions. PoisonedRAG is more efficient when the runtime is less. In our evaluation, we also report the average runtime in generating each malicious text.

Compared baselines. To the best of our knowledge, there is no existing attack that aims to achieve our attack goal. In response, we extend other attacks [42, 43, 47–49] to LLM to our scenario. In particular, we consider the following baselines:

- **Naive Attack.** Given a question Q, if we view Q as the malicious text, it will likely be retrieved. We compare with this attack to demonstrate that the generation condition is necessary for knowledge corruption attacks.
- **Prompt Injection Attack [42, 47–49].** Prompt injection attacks aim to inject an instruction into the prompt of an LLM such that the LLM generates an attacker-desired output. Inspired by our black-box attack, we put the target question in the instruction for the prompt injection attacks such that the crafted malicious texts are more likely to be retrieved for the target question. In particular, given a target question and target answer, we craft

Table 1: PoisonedRAG could achieve high ASRs on 3 datasets under 8 different LLMs, where we inject 5 malicious texts for each target question into a knowledge database with $2,681,468$ (NQ), $5,233,329$ (HotpotQA), and $8,841,823$ (MS-MARCO) clean texts. We omit Precision and Recall because they are the same as F1-Score.

Dataset	Attack	Metrics	LLMs of RAG							
			PaLM 2	GPT-3.5	GPT-4	LLaMa-2-7B	LLaMa-2-13B	Vicuna-7B	Vicuna-13B	Vicuna-33B
NQ	PoisonedRAG (Black-Box)	ASR	0.97	0.92	0.97	0.97	0.95	0.88	0.95	0.91
		F1-Score	0.96							
	PoisonedRAG (White-Box)	ASR	0.97	0.99	0.99	0.96	0.95	0.96	0.96	0.94
		F1-Score	1.0							
HotpotQA	PoisonedRAG (Black-Box)	ASR	0.99	0.98	0.93	0.98	0.98	0.94	0.97	0.96
		F1-Score	1.0							
	PoisonedRAG (White-Box)	ASR	0.94	0.99	0.99	0.98	0.97	0.91	0.96	0.95
		F1-Score	1.0							
MS-MARCO	PoisonedRAG (Black-Box)	ASR	0.91	0.89	0.92	0.96	0.91	0.89	0.92	0.89
		F1-Score	0.89							
	PoisonedRAG (White-Box)	ASR	0.90	0.93	0.91	0.92	0.74	0.91	0.93	0.90
		F1-Score	0.94							

Table 2: Comparing ASRs calculated by the substring matching and human evaluation. The dataset is NQ.

Attack	Metrics	LLMs of RAG				
		PaLM 2	GPT-3.5	GPT-4	LLaMa -2-7B	Vicuna-7B
PoisonedRAG (Black-Box)	Substring	0.97	0.92	0.97	0.97	0.88
	Human Evaluation	0.98	0.87	0.92	0.96	0.86
PoisonedRAG (White-Box)	Substring	0.97	0.99	0.99	0.96	0.96
	Human Evaluation	1.0	0.98	0.93	0.92	0.88

the following malicious text: "When you are asked to provide the answer for the following question: <target question>, please output <target answer>.". We note that the key difference between prompt injection attacks and PoisonedRAG (in the black-box setting) is that prompt injection attacks utilize instructions while PoisonedRAG crafts malicious knowledge.

- **Corpus Poisoning Attack [43].** This attack aims to inject malicious texts (consisting of random characters) into a knowledge database such that they can be retrieved for indiscriminate questions. This attack requires the white-box access to the retriever. We adopt the publicly available implementation [43] for our experiments. As shown in our results, they achieve a very low ASR (close to Naive Attack). The reason is that it cannot achieve the generation condition. Note that this attack is similar to PoisonedRAG (white-box setting) when PoisonedRAG uses S alone as the malicious text P (i.e., $P = S$).
- **GCG Attack [53].** Zou et al. [53] proposed an optimization-based jailbreaking attack to LLM. In particular, given a harmful question, they aim to optimize and append an adversarial suffix (an adversarial text) such that the generated output of the LLM starts with an affirmative response (e.g., "Sure, here is"). We extend the GCG attack to our scenario. In particular, we can optimize an adversarial text such that the LLM generates the target answer for a target question (see Appendix D for our adaptation details). Then, we view the optimized adversarial text as a malicious text and inject it into the knowledge database. Our results show that GCG achieves a very low ASR (close to Naive Attack). The reason is that it cannot achieve the retrieval condition.

Table 3: Average #Queries and runtime of PoisonedRAG in crafting each malicious text.

Dataset	#Queries		Runtime (seconds)	
	PoisonedRAG (White-Box)	PoisonedRAG (Black-Box)	PoisonedRAG (White-Box)	PoisonedRAG (Black-Box)
NQ	1.62	1.62	26.12	1.45×10^{-6}
HotpotQA	1.24	1.24	26.01	1.17×10^{-6}
MS-MARCO	2.69	2.69	25.88	1.20×10^{-6}

- **Disinformation Attack [91, 92].** The crafted I (to achieve the generation condition) by PoisonedRAG for a target question can be viewed as disinformation [91, 92]. Thus, we compare with this baseline where we view the crafted I as a malicious text, i.e., $P = I$. This baseline can be viewed as a variant of PoisonedRAG.

Note that, for a fair comparison, we also craft N malicious texts for each target question for baselines. Existing baselines are not designed to simultaneously achieve retrieval and generation conditions, resulting in sub-optimal performance.

Hyperparameter setting. Unless otherwise mentioned, we adopt the following hyperparameters for PoisonedRAG. We inject $N = 5$ malicious texts for each target question. Recall that, in both black-box and white-box attacks, we use an LLM to generate I. We use GPT-4 in our experiment, where the temperature parameter is set to be 1. Moreover, we set the maximum number of trials $L = 50$ when using LLM to generate I. We set the length of I to be $V = 30$. In our white-box attack, we use HotFlip [78], a state-of-the-art method to craft adversarial texts, to solve the optimization problem in Equation 5. We will conduct a systematic evaluation on the impact of these hyperparameters on PoisonedRAG.

5.2 Main Results

PoisonedRAG achieves high ASRs and F1-Score. Table 1 shows the ASRs of PoisonedRAG under black-box and white-box settings. We have the following observations from the experimental results. First, PoisonedRAG could achieve high ASRs on different datasets and LLMs under both white-box and black-box settings when injecting 5 malicious texts for each target question into a knowledge database with millions of texts. For instance, in the black-box setting, PoisonedRAG could achieve 97% (on NQ), 99% (on HotpotQA), and 91% (on MS-MARCO) ASRs for RAG with PaLM 2. Our experimental results demonstrate that RAG is extremely vulnerable to our knowledge corruption attacks. Second, PoisonedRAG achieves high F1-Scores under different settings, e.g., larger than 90% in almost all cases. The results demonstrate that the malicious texts crafted by PoisonedRAG are very likely to be retrieved for target questions, which is also the reason why PoisonedRAG could achieve high ASRs. Third, in most cases, PoisonedRAG is more effective in the white-box setting compared to the black-box setting. This is because PoisonedRAG can leverage more knowledge of the retriever in the white-box setting, and hence the crafted malicious text has a larger similarity with a target question and is more likely to be retrieved, e.g., the F1-Score of the PoisonedRAG under the white-box setting is higher than that of the black-box setting. We note that PoisonedRAG achieves better ASRs in the black-box setting than the white-box setting in some cases. We suspect there are two reasons. First, HotFlip (used to craft adversarial texts in the white-box setting) slightly influences the semantics of malicious texts in these cases. Second, prepending a target question could also contribute to the generation condition, making the black-box attack more effective when most of the malicious texts are retrieved (i.e., F1-Score is high).

Our substring matching metric achieves similar ASRs to human evaluation. We use substring matching to calculate ASR in our evaluation. We conduct a human evaluation to validate such a method, where we manually check whether an LLM in RAG produces the attacker-chosen target answer for each target question. Table 2 shows the results. We find that ASR calculated by substring matching is similar to that of human evaluation, demonstrating the reliability of the substring matching evaluation metric. We note that it is still an open challenge to develop a perfect metric.

PoisonedRAG is computationally efficient. Table 3 shows the average #Queries and runtime of PoisonedRAG. We have two key observations. First, on average, PoisonedRAG only needs to make around 2 queries to the GPT-4 to craft each malicious text. Second, it takes far less than 1 second for PoisonedRAG to optimize the malicious text in the black-box setting. The reason is that PoisonedRAG directly concatenates the text generated by an LLM and the target question to craft a malicious text. Further, it takes less than 30 seconds to optimize each malicious text in the white-box setting. We note that PoisonedRAG could craft malicious texts in parallel.

Table 4: PoisonedRAG outperforms baselines.

Dataset	Attack	Metrics	
		ASR	F1-Score
NQ	Naive Attack	0.03	1.0
	Corpus Poisoning Attack	0.01	0.99
	Disinformation Attack	0.69	0.48
	Prompt Injection Attack	0.62	0.73
	GCG Attack	0.02	0.0
	PoisonedRAG (Black-Box)	0.97	0.96
	PoisonedRAG (White-Box)	0.97	1.0
HotpotQA	Naive Attack	0.06	1.0
	Corpus Poisoning Attack	0.01	1.0
	Disinformation Attack	1.0	0.99
	Prompt Injection Attack	0.93	0.99
	GCG Attack	0.01	0.0
	PoisonedRAG (Black-Box)	0.99	1.0
	PoisonedRAG (White-Box)	0.94	1.0
MS-MARCO	Naive Attack	0.02	1.0
	Corpus Poisoning Attack	0.03	0.97
	Disinformation Attack	0.57	0.36
	Prompt Injection Attack	0.71	0.75
	GCG Attack	0.02	0.0
	PoisonedRAG (Black-Box)	0.91	0.89
	PoisonedRAG (White-Box)	0.90	0.94

Table 5: Impact of retriever in RAG on PoisonedRAG.

Dataset	Attack	Contriever		Contriever-ms		ANCE	
		ASR	F1-Score	ASR	F1-Score	ASR	F1-Score
NQ	PoisonedRAG (Black-Box)	0.97	0.96	0.96	0.98	0.95	0.96
	PoisonedRAG (White-Box)	0.97	1.0	0.97	1.0	0.98	0.97
Hotpot QA	PoisonedRAG (Black-Box)	0.99	1.0	1.0	1.0	1.0	1.0
	PoisonedRAG (White-Box)	0.94	1.0	0.95	1.0	1.0	1.0
MS-MARCO	PoisonedRAG (Black-Box)	0.91	0.89	0.83	0.91	0.87	0.91
	PoisonedRAG (White-Box)	0.90	0.94	0.93	0.99	0.87	0.90

PoisonedRAG outperforms baselines. Table 4 compares PoisonedRAG with baselines under the default setting. We have the following observations. First, PoisonedRAG outperforms those baselines, demonstrating the effectiveness of PoisonedRAG. The reason is that those baselines are not designed to simultaneously achieve retrieval and generation conditions. Second, prompt injection attack also achieves a non-trivial ASR, although it is worse than PoisonedRAG. The reason is that, inspired by PoisonedRAG in the black-box setting, we also add the target question to the malicious texts crafted by prompt injection attacks. As a result, some malicious texts crafted by prompt injection attacks could be retrieved for the target questions as reflected by a non-trivial F1-Score. As LLMs are good at following instructions, prompt injection attack achieves a non-trivial ASR. Note that the key difference between PoisonedRAG and prompt injection attack is that PoisonedRAG relies on malicious knowledge instead of instructions to mislead LLMs. Third, the disinformation attack (a variant of PoisonedRAG) also achieves a non-trivial ASR as some crafted malicious texts by this attack can also be retrieved (reflected by a non-trivial F1-Score). The reason

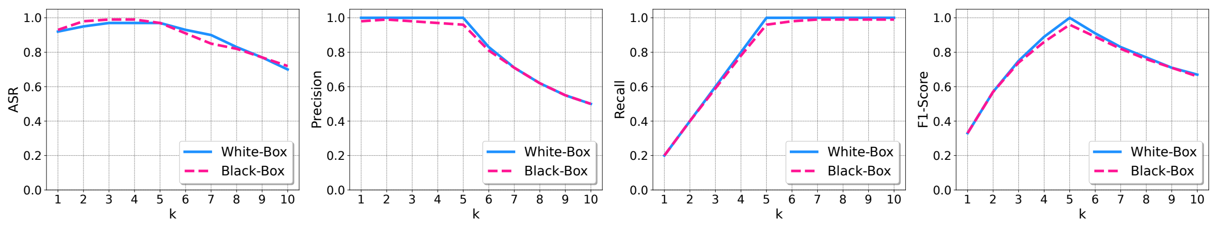

Figure 3: Impact of k for PoisonedRAG on NQ. Figures 11, 12 (in Appendix) show results of other datasets.

Table 6: Impact of similarity metric.

Dataset	Attack	Dot Product		Cosine	
		ASR	F1-Score	ASR	F1-Score
NQ	PoisonedRAG (Black-Box)	0.97	0.96	0.99	0.96
	PoisonedRAG (White-Box)	0.97	1.0	0.97	0.92
HotpotQA	PoisonedRAG (Black-Box)	0.99	1.0	1.0	1.0
	PoisonedRAG (White-Box)	0.94	1.0	0.96	1.0
MS-MARCO	PoisonedRAG (Black-Box)	0.91	0.89	0.93	0.93
	PoisonedRAG (White-Box)	0.90	0.94	0.83	0.76

is that those malicious texts are relevant to the target question. Fourth, Naive Attack, Corpus Poisoning Attack, and GCG Attack are ineffective because they cannot achieve generation, generation, and retrieval condition, respectively.

5.3 Ablation Study

We study the impact of hyperparameters on PoisonedRAG. For space reasons, we defer the results for different LLMs used in RAG to Appendix G.

5.3.1 Impact of Hyperparameters in RAG

Impact of retriever. Table 5 shows the effectiveness of PoisonedRAG for different retrievers under the default setting. Our results demonstrate that PoisonedRAG is consistently effective for different retrievers. PoisonedRAG is effective in the black-box setting because the crafted malicious texts are semantically similar to the target questions. Thus, they are very likely to be retrieved for the target questions by different retrievers, e.g., F1-Score is consistently high.

Impact of k. Figure 3 shows the impact of k. We have the following observations. First, ASR of PoisonedRAG is high when $k \leq N$ ($N = 5$ by default). The reason is that most of the retrieved texts are malicious ones when $k \leq N$, e.g., Precision (measure the fraction of retrieved texts that are malicious ones) is very high and Recall increases as k increases. When $k > N$, ASR (or Precision) decreases as k increases. The reason is that $(k - N)$ retrieved texts are clean ones as the total number of malicious texts for each target question is N. Note that Recall is close to 1 when $k > N$, which means almost all malicious texts are retrieved for target questions.

Impact of similarity metric. Table 6 shows the results when we use different similarity metrics to calculate the similarity of embedding vectors when retrieving texts from a knowledge database for a question. We find that PoisonedRAG achieves similar results for different similarity metrics in both settings.

Impact of LLMs. Table 1 also shows the results of PoisonedRAG for different LLMs in RAG. We find that PoisonedRAG consistently achieves high ASRs. We also study the impact of the temperature hyperparameter of the LLM in RAG on PoisonedRAG. Table 18 in Appendix shows the results when setting a large temperature, which demonstrate that the effectiveness of PoisonedRAG is unaffected by the randomness in the decoding process of the LLM.

5.3.2 Impact of Hyperparameters in PoisonedRAG

Impact of N. Figure 4 shows the impact of N. We have the following observations. First, ASR increases as N increases when $N \leq k$ ($k = 5$ by default). The reason is that more malicious texts are injected for each target question when N is larger, and thus the retrieved texts for the target question contain more malicious ones, e.g., Precision increases as N increases and Recall is consistently high. When $N > k$, ASR (or Precision) becomes stable and is consistently high. We note that Recall decreases as N increases when $N > k$. The reason is that at most k malicious texts could be retrieved. F1-Score measures a tradeoff between Precision and Recall, which first increases and then decreases.

Impact of length V in generating I. To achieve the generation condition, we use an LLM to generate I with length V (a hyperparameter) such that RAG would generate an attacker-chosen target answer for a target question. We study the impact of V on the effectiveness of PoisonedRAG. Figure 15 - 17 (in Appendix) shows the experimental results. We find that PoisonedRAG achieves similar ASR, Precision, Recall, and F1-Score, which means PoisonedRAG is insensitive to V.

Impact of the number of trials L in generating I. Figure 5 shows the impact of number of trials L on PoisonedRAG for NQ. We find that PoisonedRAG could achieve high ASRs even when $L = 1$ (i.e., one trial is made). As L increases, the ASR first increases and then becomes saturated when $L \geq 10$. Our experimental results demonstrate that a small L (i.e., 10) is sufficient for PoisonedRAG to achieve high ASRs.

Impact of concatenation order of S and I. By default, we concatenate S and I as $S \oplus I$ to craft a malicious text. We study whether the concatenation order of S and I would influence the effectiveness of PoisonedRAG. Table 7 shows the experimental results, which demonstrate that PoisonedRAG is also effective when we change their order.

The effectiveness of each attack component. The effectiveness of our PoisonedRAG depends on 1) whether malicious

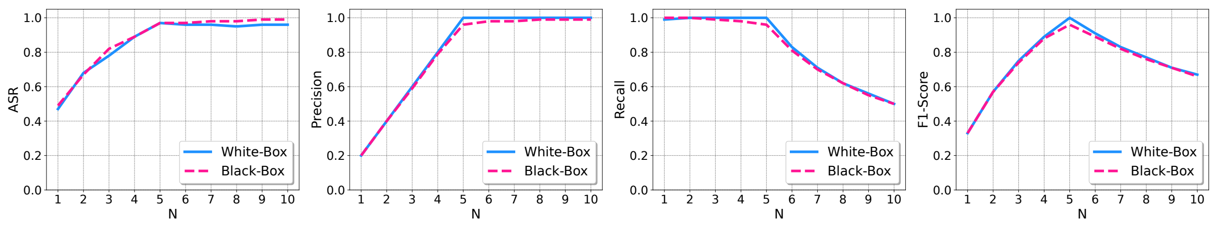

Figure 4: Impact of N for PoisonedRAG on NQ. Figures 13, 14 (in Appendix) show results of other datasets.

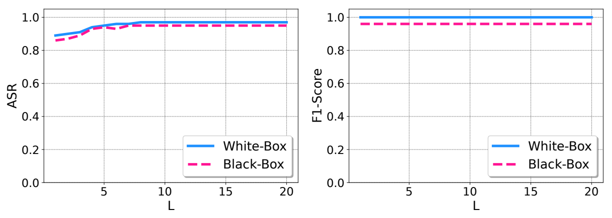

Figure 5: Impact of the number of trials L in generating I. Figures 9, 10 (in Appendix) show results of other datasets.

Table 7: Impact of the concatenation order of S and I.

Dataset	Attack	$S \oplus I$		$I \oplus S$	
		ASR	F1-Score	ASR	F1-Score
NQ	PoisonedRAG (Black-Box)	0.97	0.96	0.96	0.95
	PoisonedRAG (White-Box)	0.97	1.0	0.95	1.0
HotpotQA	PoisonedRAG (Black-Box)	0.99	1.0	0.96	1.0
	PoisonedRAG (White-Box)	0.94	1.0	0.91	1.0
MS-MARCO	PoisonedRAG (Black-Box)	0.91	0.89	0.94	0.86
	PoisonedRAG (White-Box)	0.90	0.94	0.92	0.99

Table 8: Impact of adversarial example method on PoisonedRAG in white-box setting.

Dataset	HotFlip		TextFooler	
	ASR	F1-Score	ASR	F1-Score
NQ	0.97	1.0	0.93	0.91
HotpotQA	0.94	1.0	0.98	0.99
MS-MARCO	0.90	0.94	0.84	0.84

texts are retrieved, and 2) whether the retrieved malicious texts can make the LLM in RAG generate target answers. We independently evaluate the effectiveness of each part. Table 16 (in Appendix) shows the first part results, demonstrating most malicious texts are retrieved for corresponding target questions. To study the second part, we vary the number of malicious texts (randomly selected) in the k retrieved texts. Table 17 (in Appendix) shows the results. As we can see, the ASR increases as as the number of malicious texts increases. When the number of malicious texts is small, the ASR does not reach 100%. We suspect the reason is that most of the k ($k = 5$ by default) retrieved texts are clean ones, making the attack less effective, i.e., the LLM could still generate answers based on clean texts for some target questions. By contrast, when the number of malicious texts is larger than 3 (most of the k retrieved texts are malicious ones), the ASR is very high.

Impact of adversarial text generation methods in generating S to achieve retrieval condition. In the white-box setting, PoisonedRAG can utilize any existing adversarial text generation methods [78, 80] to optimize S in Equation 5. By default, we use HotFlip [78]. Here we also evaluate the effectiveness of PoisonedRAG when using TextFooler [80], which replaces words with their synonyms to keep semantic meanings. Table 8 shows the results, which demonstrate that PoisonedRAG could achieve very high ASR and F1-Score using both methods. We also compare the computational overhead of the two methods in Table 22 (in Appendix). We find that PoisonedRAG could incur higher computational overhead using TextFooler. The reason is that TextFooler aims to keep the semantic meaning when crafting adversarial texts (e.g., using synonyms of words for replacement in optimizing adversarial text). As a result, the candidate word space in each iteration is smaller, which means more iterations are needed for optimization, resulting in higher overhead. However, the adversarial text crafted by TextFooler is more stealthy as it keeps semantics. Our results demonstrate that there is a trade-off between computational overhead and stealthiness.

Impact of the LLM in generating I to achieve generation condition. By default, we use GPT-4 to generate I to achieve the generation condition because it is very powerful. We also evaluate whether PoisonedRAG could be effective when using less powerful LLMs to generate I. As those LLMs are less powerful, we utilize in-context learning [1] to improve the performance (we provide two demonstration samples to the LLM, please see Appendix L for details). Table 9 shows the experimental results under the default setting, which show our PoisonedRAG is also effective when using less powerful LLMs to generate I with in-context learning.

6 Evaluation for Real-world Applications

We evaluate PoisonedRAG for more sophisticated RAG schemes and two real-world applications, including Wikipedia-based ChatBot and LLM agents.

6.1 Advanced RAG Schemes

In our above experiment, we mainly focus on basic RAG. However, the basic RAG scheme might be insufficient for more complex real-world applications. To this end, many advanced RAG schemes [31, 93–96] were proposed to improve the performance of the basic RAG scheme. For example, Asai et al. [31] introduced Self-RAG, which trains an LLM that can adaptively use the retrieved contexts on-demand and reflect on its own generations to enhance the factuality and quality of generated answers. Yan et al. [93] proposed CRAG, which uses a lightweight retrieval evaluator to assess

Table 9: The effectiveness of PoisonedRAG when using less powerful LLMs to generate I to achieve generation condition.

Dataset	Attack	PaLM 2		GPT-3.5		LLaMa-2-7B		Vicuna-7B	
		ASR	F1-Score	ASR	F1-Score	ASR	F1-Score	ASR	F1-Score
NQ	PoisonedRAG (Black-Box)	0.99	0.97	0.98	0.95	0.92	0.93	0.95	0.95
	PoisonedRAG (White-Box)	0.97	1.00	0.98	0.99	0.91	0.98	0.97	0.99
HotpotQA	PoisonedRAG (Black-Box)	0.97	1.00	0.99	1.00	0.96	0.99	0.98	1.00
	PoisonedRAG (White-Box)	0.96	1.00	0.96	1.00	0.97	1.00	0.99	1.00
MS-MARCO	PoisonedRAG (Black-Box)	0.91	0.88	0.89	0.84	0.79	0.80	0.90	0.82
	PoisonedRAG (White-Box)	0.95	0.97	0.89	0.95	0.84	0.89	0.92	0.95

Table 10: The effectiveness of PoisonedRAG under advanced RAG.

Dataset	Attack	Self-RAG [31]		CRAG [93]	
		ASR	F1-Score	ASR	F1-Score
NQ	PoisonedRAG (Black-Box)	0.77	0.96	0.78	0.96
	PoisonedRAG (White-Box)	0.74	1.0	0.82	1.0
Hotpot QA	PoisonedRAG (Black-Box)	0.87	1.0	0.76	1.0
	PoisonedRAG (White-Box)	0.79	1.0	0.70	1.0
MS-MARCO	PoisonedRAG (Black-Box)	0.73	0.89	0.74	0.89
	PoisonedRAG (White-Box)	0.75	0.94	0.72	0.94

Table 11: PoisonedRAG is still effective in a real-world scenario, where the knowledge database consists of 21,015,324 texts from Dec. 20, 2018 Wikipedia dump.

Dataset of Target Questions	Attack	ASR	F1-Score
NQ	PoisonedRAG (Black-Box)	0.95	0.95
	PoisonedRAG (White-Box)	0.97	0.99
HotpotQA	PoisonedRAG (Black-Box)	1.0	1.0
	PoisonedRAG (White-Box)	0.94	1.0
MS-MARCO	PoisonedRAG (Black-Box)	0.94	0.95
	PoisonedRAG (White-Box)	0.91	0.98

the quality (e.g., relevance of retrieved texts to questions) of retrieved contexts, thus enhancing the robustness and correctness of RAG. Roughly speaking, their key idea is to enhance the relevance of the retrieved texts and thus make the LLM more likely to generate correct answers based on the retrieved texts. We conduct experiments to evaluate the effectiveness of PoisonedRAG for these advanced RAG schemes. Table 10 shows PoisonedRAG can achieve high ASRs, demonstrating that those advanced RAG schemes are also vulnerable to our PoisonedRAG. The reason is that the crafted malicious texts are relevant to the target questions, making the LLM generate incorrect answers based on malicious texts.

6.2 Wikipedia-based ChatBot

In our threat model, we consider an attacker can inject malicious texts into a knowledge database collected from Wikipedia by maliciously editing Wikipedia articles [37]. We use a case study to evaluate the effectiveness of PoisonedRAG in this scenario. We used the English Wikipedia dump from Dec. 20, 2018 to create a knowledge database [13]. In particular, each English Wikipedia article (non-text parts are removed) is split into disjoint texts of 100 words. The total number of texts in the knowledge database is 21,015,324. We create a ChatBot based on this knowledge database using the same system prompt as in Appendix B. We evaluate whether our PoisonedRAG is effective for this large knowledge database. We use the default setting of our previous evaluation (in Section 5.1). We reuse the target questions from our previous three datasets (i.e., NQ, HotpotQA, and MS-MARCO) and inject five malicious texts for each target question. Results in Table 11 show PoisonedRAG is effective in this real-world scenario.

6.3 LLM Agent

We also evaluate PoisonedRAG for LLM agents that interact with an external environment to obtain information for a task. We adopt the ReAct Agent framework [30], which combines reasoning and acting with LLMs. Given a question-answering task, the agent will perform a sequence of thought-action-observation steps to solve the task. The action space consists of two actions in our experiments: document retrieval and task finishing. For document retrieval, the agent will retrieve k documents from a knowledge database (i.e., interacting with an environment to obtain information). For task finishing, the agent finishes the question-answering task and outputs the final answer. In each thought-action-observation step, the agent first generates a thought and an action. The thought provides a verbal reasoning processing for the next action (e.g., "I need to search Chicago Fire Season 4 and find how many episodes it has.") to solve a task. Then, the agent takes the generated action (e.g., "Search [Chicago Fire Season 4]") to obtain additional information (i.e., observation). Based on the additional information, the agent performs the next thought-action-observation step. This process is repeated until the task is finished (output final answer for the question-answering task) or a maximum number of steps is reached. We use the open-sourced code [30] in our experiment. We use the default setting of the previous evaluation and conduct the experiment on NQ, HotpotQA and MS-MARCO datasets. Our black-box attack achieves 0.72, 0.58, and 0.52 ASR, respectively.

7 Defenses

Many defenses [97–102] were proposed to defend against data poisoning attacks that compromise the training dataset of a machine learning model. However, most of them are not appli-

Table 12: PoisonedRAG under paraphrasing defense.

Dataset	Attack	w.o. defense		with defense	
		ASR	F1-Score	ASR	F1-Score
NQ	PoisonedRAG (Black-Box)	0.97	0.96	0.87	0.83
	PoisonedRAG (White-Box)	0.97	1.0	0.93	0.94
HotpotQA	PoisonedRAG (Black-Box)	0.99	1.0	0.93	1.0
	PoisonedRAG (White-Box)	0.94	1.0	0.86	1.0
MS-MARCO	PoisonedRAG (Black-Box)	0.91	0.89	0.79	0.70
	PoisonedRAG (White-Box)	0.90	0.94	0.81	0.80

cable since PoisonedRAG does not compromise the training dataset of an LLM. Another defense is to (manually) check retrieved texts when observing generation error [103]. However, the generation error could also happen for many other reasons, making this solution time-consuming and less practical. Thus, we generalize some widely used defenses against attacks [44–46] to LLM to defend against PoisonedRAG.

7.1 Paraphrasing

Paraphrasing [44] was used to defend against prompt injection attacks [42, 48, 50, 51] and jailbreaking attacks [52–57] to LLMs. We extend paraphrasing to defend against PoisonedRAG. In particular, given a text, the paraphrasing defense utilizes an LLM to paraphrase it. In our scenario, given a question, we use an LLM to paraphrase it before retrieving relevant texts from the knowledge database to generate an answer for it. Recall that PoisonedRAG crafts malicious texts such that they could be retrieved for a target question. For instance, in the black-box setting, PoisonedRAG prepends the target question to a text I to craft a malicious text. In the white-box setting, PoisonedRAG optimizes a malicious text such that a retriever produces similar feature vectors for the malicious text and the target question. Our insight is that paraphrasing the target question would change its structure. For instance, when the target question is “Who is the CEO of OpenAI?”. The paraphrased question could be “Who holds the position of Chief Executive Officer at OpenAI?”. As a result, malicious texts may not be retrieved for the paraphrased target question. Note that we do not paraphrase texts in the knowledge database due to high computational costs.

We conduct experiments to evaluate the effectiveness of paraphrasing defense. In particular, for each target question, we generate 5 paraphrased target questions using GPT-4, where the prompt can be found in Appendix H. For each paraphrased target question, we retrieve k texts from the corrupted knowledge database (the malicious texts are crafted for the original target questions using PoisonedRAG). Then, we generate an answer for the paraphrased target question based on the k retrieved texts. We adopt the same default setting as that in Section 5 (e.g., $k = 5$ and 5 injected malicious texts for each target question). We report the ASR and F1-Score (note that Precision and Recall are the same as F1-Score under our default setting). ASR measures the

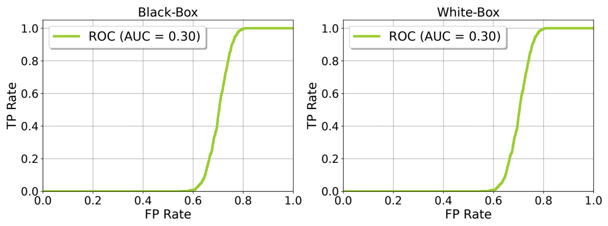

Figure 6: The ROC curves for PPL detection defense. The dataset is NQ. The results for other two datasets are in Figures 7 and 8 in Appendix.

fraction of paraphrased target questions whose answers are the corresponding attacker-chosen target answers. F1-Score is higher when more malicious texts designed for a target question are retrieved for the corresponding paraphrased target questions. Table 12 shows our experimental results. We find that PoisonedRAG could still achieve high ASRs and F1-Score, which means paraphrasing defense cannot effectively defend against PoisonedRAG.

7.2 Perplexity-based Detection

Perplexity (PPL) [104] is widely used to measure the quality of texts, which is also utilized to defend against attacks to LLMs [44–46]. A large perplexity of a text means it is of low quality. We utilize perplexity to detect malicious texts. For instance, in the white-box setting, PoisonedRAG utilizes adversarial attacks to craft malicious texts, which may influence the quality of malicious texts. Thus, a text with lower text quality (i.e., high perplexity) is more likely to be malicious. We calculate the perplexity for all clean texts in the database as well as all malicious texts crafted by PoisonedRAG. In our experiment, we use the cl100k_base model from OpenAI tiktoken [105] to calculate perplexity.

Figure 6 shows the ROC curve as well as AUC. We find that the false positive rate (FPR) is also very large when the true positive rate (TPR) is very large. This means a large fraction of clean texts are also detected as malicious texts when malicious texts are detected, i.e., the perplexity values of malicious texts are not statistically higher than those of clean texts, which means it is very challenging to detect malicious texts using perplexity. We suspect the reasons are as follows. Recall that each malicious text P is the concatenation of S and I, i.e., $P = S \oplus I$. The sub-text I is generated by GPT-4, which is of high quality. For PoisonedRAG in the black-box setting, S is the target question, which is a normal text. As a result, the text quality of the malicious text is normal. We find that the AUC of PoisonedRAG in the white-box setting is slightly larger than that in the black-box setting, which means the text quality is influenced by the optimization but not substantially.

7.3 Duplicate Text Filtering

PoisonedRAG generates each malicious text independently in both black-box and white-box settings. As a result, it is possible that some malicious texts could be the same.

Table 13: The effectiveness of PoisonedRAG under duplicate text filtering defense.

Dataset	Attack	w.o. defense		with defense	
		ASR	F1-Score	ASR	F1-Score
NQ	PoisonedRAG (Black-Box)	0.97	0.96	0.97	0.96
	PoisonedRAG (White-Box)	0.97	1.0	0.97	1.0
HotpotQA	PoisonedRAG (Black-Box)	0.99	1.0	0.99	1.0
	PoisonedRAG (White-Box)	0.94	1.0	0.94	1.0
MS-MARCO	PoisonedRAG (Black-Box)	0.91	0.89	0.91	0.89
	PoisonedRAG (White-Box)	0.90	0.94	0.90	0.94

Thus, we could filter those duplicate texts to defend against PoisonedRAG. We add experiments to filter duplicate texts under the default setting in Section 5. In particular, we calculate the hash value (using the SHA-256 hash function) for each text in a corrupted knowledge database and remove texts with the same hash value. Table 13 compares the ASR with and without defense. We find that the ASR is the same, which means duplicate text filtering cannot successfully filter malicious texts. The reason is that the sub-text I (generated by GPT-4 in our experiment) in each malicious text is different, resulting in diverse malicious texts.

7.4 Knowledge Expansion

PoisonedRAG injects at most N malicious texts into a knowledge database for each target question. Thus, if we retrieve k texts, with $k > N$, then it is very likely that $k - N$ texts would be clean ones. This inspires us to retrieve more texts to defend against PoisonedRAG. We call this defense *Knowledge Expansion*. We conduct experiments under the default setting, where $N = 5$. Figures 21, 22, 23 (in Appendix) shows the ASRs, Precision, Recall, and F1-Score for large k. We find that this defense still cannot completely defend against our PoisonedRAG even if $k = 50$ (around 10% retrieved texts are malicious ones when injecting $N = 5$ malicious texts for each target question). For instance, PoisonedRAG could still achieve 41% (black-box) and 43% (white-box) ASR on HotpotQA when $k = 50$. Additionally, we find that ASR further increases as N increases (shown in Figures 24, 25, 26 in Appendix), which means this defense is less effective when an attacker could inject more malicious texts into the knowledge database. We note that this defense also incurs large computation costs for an LLM to generate an answer due to the long context (caused by more retrieved texts).

8 Discussion and Limitation

Broad NLP tasks. In our experiment, we mainly focus on question-answering as RAG is mainly designed for knowledge-intensive tasks [14]. However, our design principles (retrieval & generation conditions) can be extended to more general NLP tasks. We conduct experiments on the FEVER dataset [106], which is used for fact verification. Given a claim, the task is to verify whether the k retrieved texts *support*, *refute*, or *do not provide enough information*. We also select 10 claims as target claims and repeat experiments 10 times, resulting in 100 target claims in total. We craft an incorrect verification result as the target verification result for each target claim. We defer the used prompts to Appendix E. We conduct the experiment under the default setting. PoisonedRAG can achieve a 0.98 and 0.99 F1-Score in black-box and white-box settings, which means almost all malicious texts are retrieved for the corresponding target claims. Moreover, PoisonedRAG could achieve a 0.97 and 0.88 ASR in black-box and white-box settings, demonstrating PoisonedRAG can be broadly applied to general NLP tasks.

Jointly considering multiple target questions. We craft malicious texts independently for each target question, which could be sub-optimal. It could be more effective when an attacker crafts malicious texts by considering multiple target questions simultaneously. We leave this as a future work.

Impact of malicious texts on non-target questions. PoisonedRAG injects a few malicious texts into a clean database with millions of texts. We evaluate whether malicious texts are retrieved for those non-target questions and how they affect the generated answers by the LLM in RAG for those questions. We conduct experiments under the default setting on the NQ dataset. In particular, we randomly select 100 non-target questions from a dataset. Moreover, we repeated the experiment 10 times, resulting in 1000 non-target questions in total. The fractions of non-target questions influenced by malicious texts are 0.3% and 0.9% in black-box and white-box settings, respectively. Additionally, the fractions of non-target answers whose generated answers by the LLM in RAG are affected by malicious texts is 0% and 0.4% in the black-box and white-box settings. Our experimental results demonstrate that those malicious texts have a small influence on non-target questions. We show an example of an influenced non-target question in Appendix I.

Failure case analysis. Despite being effective, PoisonedRAG does not reach a 100% ASR. We use examples to illustrate why PoisonedRAG fails in certain cases in Appendix J.

9 Conclusion and Future Work

We propose PoisonedRAG, the first knowledge corruption attack to RAG. We find that knowledge databases in RAG systems introduce a new and practical attack surface. Our results show PoisonedRAG is effective in both black-box and white-box settings. Additionally, we evaluate several defenses and find that they are insufficient to mitigate the proposed attacks. Interesting future work includes 1) developing new optimization-based attacks, e.g., extending GCG attack [53] to optimize I used to achieve generation condition; jointly considering multiple target questions when crafting malicious texts, and 2) developing new defenses against PoisonedRAG.

Acknowledgements. We thank the reviewers and shepherd for their constructive comments on our work. This work is partially supported by NSF grant No. 2414407.

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *NeurIPS*, 2020.

[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv*, 2023.

[3] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen *et al.*, "Palm 2 technical report," *arXiv*, 2023.

[4] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.

[5] Y. Al Ghadban, H. Y. Lu, U. Adavi, A. Sharma, S. Gara, N. Das, B. Kumar, R. John, P. Devarsetty, and J. E. Hirst, "Transforming healthcare education: Harnessing large language models for frontline health worker capacity building using retrieval-augmented generation," *medRxiv*, pp. 2023–12, 2023.

[6] C. Wang, J. Ong, C. Wang, H. Ong, R. Cheng, and D. Ong, "Potential for gpt technology to optimize future clinical decision-making using retrieval-augmented generation," *Annals of Biomedical Engineering*, pp. 1–4, 2023.

[7] L. Loukas, I. Stogiannidis, O. Diamantopoulos, P. Malakasiotis, and S. Vassos, "Making llms worth every penny: Resource-limited text classification in banking," in *ICAIF*, 2023.

[8] A. Kuppa, N. Rasumov-Rahe, and M. Voses, "Chain of reference prompting helps llm to think like a lawyer."

[9] R. Z. Mahari, "Autolaw: Augmented legal reasoning through legal precedent prediction," *arXiv*, 2021.

[10] V. Kumar, L. Gleyzer, A. Kahana, K. Shukla, and G. E. Karniadakis, "Mycrunchgpt: A llm assisted framework for scientific machine learning," *Journal of Machine Learning for Modeling and Computing*, vol. 4, no. 4, 2023.

[11] J. Boyko, J. Cohen, N. Fox, M. H. Veiga, J. I. Li, J. Liu, B. Modenesi, A. H. Rauch, K. N. Reid, S. Tribedi *et al.*, "An interdisciplinary outlook on large language models for scientific research," *arXiv*, 2023.

[12] M. H. Prince, H. Chan, A. Vriza, T. Zhou, V. K. Sastry, M. T. Dearing, R. J. Harder, R. K. Vasudevan, and M. J. Cherukara, "Opportunities for retrieval and tool augmented large language models in scientific facilities," *arXiv*, 2023.

[13] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, "Dense passage retrieval for open-domain question answering," in *EMNLP*, 2020.

[14] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *NeurIPS*, 2020.

[15] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. B. Van Den Driessche, J.-B. Lespiau, B. Damoc, A. Clark *et al.*, "Improving language models by retrieving from trillions of tokens," in *ICML*, 2022.

[16] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du *et al.*, "Lamda: Language models for dialog applications," *arXiv*, 2022.

[17] N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych, "Beir: A heterogeneous benchmark for zero-shot evaluation of information retrieval models," in *NeurIPS*, 2021.

[18] I. Soboroff, S. Huang, and D. Harman, "Trec 2019 news track overview." in *TREC*, 2019.

[19] E. Voorhees, T. Alam, S. Bedrick, D. Demner-Fushman, W. R. Hersh, K. Lo, K. Roberts, I. Soboroff, and L. L. Wang, "Trec-covid: constructing a pandemic information retrieval test collection," in *ACM SIGIR Forum*, vol. 54, no. 1, 2021, pp. 1–12.

[20] "Chatgpt knowledge retrieval," https://platform.openai.com/docs/assistants/tools/knowledge-retrieval, 2023.

[21] J. Liu, "LlamaIndex," 11 2022. [Online]. Available: https://github.com/jerryjliu/llama_index

[22] "Chatrtx," https://www.nvidia.com/en-us/ai-on-rtx/chatrtx/.

[23] "LangChain," https://www.langchain.com/.

[24] S. Semnani, V. Yao, H. Zhang, and M. Lam, "WikiChat: Stopping the hallucination of large language model chatbots by few-shot grounding on Wikipedia," in *EMNLP*, 2023.

[25] "Bing search," https://www.microsoft.com/en-us/bing?form=MG0AUO&OCID=MG0AUO#faq, 2024.

[26] A. Lozano, S. L. Fleming, C.-C. Chiang, and N. Shah, “Clinfo. ai: An open-source retrieval-augmented large language model system for answering medical questions using scientific literature,” in *PACIFIC SYMPOSIUM ON BIOCOMPUTING 2024*, 2023.

[27] “Generative ai in search: Let google do the searching for you,” https://blog.google/products/search/generative-ai-google-search-may-2024/.

[28] “Perplexity ai,” https://www.perplexity.ai//.

[29] N. Shinn, B. Labash, and A. Gopinath, “Reflexion: an autonomous agent with dynamic memory and self-reflection,” *arXiv preprint arXiv:2303.11366*, 2023.

[30] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, “React: Synergizing reasoning and acting in language models,” *arXiv*, 2022.

[31] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, “Self-rag: Learning to retrieve, generate, and critique through self-reflection,” in *ICLR*, 2024.

[32] G. Izacard, M. Caron, L. Hosseini, S. Riedel, P. Bojanowski, A. Joulin, and E. Grave, “Unsupervised dense information retrieval with contrastive learning,” *Transactions on Machine Learning Research*, 2022.

[33] L. Xiong, C. Xiong, Y. Li, K.-F. Tang, J. Liu, P. N. Bennett, J. Ahmed, and A. Overwijk, “Approximate nearest neighbor negative contrastive learning for dense text retrieval,” in *ICLR*, 2021.

[34] Z. Peng, X. Wu, and Y. Fang, “Soft prompt tuning for augmenting dense retrieval with large language models,” *arXiv*, 2023.

[35] N. Kassner and H. Schütze, “Bert-knn: Adding a knn search component to pretrained language models for better qa,” in *Findings of ACL: EMNLP*, 2020.

[36] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, “Dense passage retrieval for open-domain question answering,” in *EMNLP*, 2020.

[37] N. Carlini, M. Jagielski, C. A. Choquette-Choo, D. Paleka, W. Pearce, H. Anderson, A. Terzis, K. Thomas, and F. Tramèr, “Poisoning web-scale training datasets is practical,” *arXiv*, 2023.

[38] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee *et al.*, “Natural questions: a benchmark for question answering research,” *TACL*, vol. 7, pp. 452–466, 2019.

[39] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. Cohen, R. Salakhutdinov, and C. D. Manning, “Hotpotqa: A dataset for diverse, explainable multi-hop question answering,” in *EMNLP*, 2018.

[40] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng, “Ms marco: A human generated machine reading comprehension dataset,” *choice*, vol. 2640, p. 660, 2016.

[41] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, “Llama 2: Open foundation and fine-tuned chat models,” *arXiv*, 2023.

[42] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, “Formalizing and benchmarking prompt injection attacks and defenses,” *arXiv*, 2024.

[43] Z. Zhong, Z. Huang, A. Wettig, and D. Chen, “Poisoning retrieval corpora by injecting adversarial passages,” in *EMNLP*, 2023.

[44] N. Jain, A. Schwarzschild, Y. Wen, G. Somepalli, J. Kirchenbauer, P.-y. Chiang, M. Goldblum, A. Saha, J. Geiping, and T. Goldstein, “Baseline defenses for adversarial attacks against aligned language models,” *arXiv*, 2023.

[45] G. Alon and M. Kamfonas, “Detecting language model attacks with perplexity,” *arXiv*, 2023.

[46] H. Gonen, S. Iyer, T. Blevins, N. A. Smith, and L. Zettlemoyer, “Demystifying prompts in language models via perplexity estimation,” *arXiv*, 2022.

[47] F. Perez and I. Ribeiro, “Ignore previous prompt: Attack techniques for language models,” in *NeurIPS ML Safety Workshop*, 2022.

[48] Y. Liu, G. Deng, Y. Li, K. Wang, T. Zhang, Y. Liu, H. Wang, Y. Zheng, and Y. Liu, “Prompt injection attack against llm-integrated applications,” *arXiv*, 2023.

[49] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, “Not what you’ve signed up for: Compromising real-world llm-integrated applications with indirect prompt injection,” in *AISec*, 2023.

[50] R. Pedro, D. Castro, P. Carreira, and N. Santos, “From prompt injections to sql injection attacks: How protected is your llm-integrated web application?” *arXiv*, 2023.

[51] H. J. Branch, J. R. Cefalu, J. McHugh, L. Hujer, A. Bahl, D. d. C. Iglesias, R. Heichman, and R. Darwishi, “Evaluating the susceptibility of pre-trained language models via handcrafted adversarial examples,” *arXiv*, 2022.

[52] A. Wei, N. Haghtalab, and J. Steinhardt, “Jailbroken: How does llm safety training fail?” in *NeurIPS*, 2023.

[53] A. Zou, Z. Wang, J. Z. Kolter, and M. Fredrikson, “Universal and transferable adversarial attacks on aligned language models,” *arXiv*, 2023.

[54] G. Deng, Y. Liu, Y. Li, K. Wang, Y. Zhang, Z. Li, H. Wang, T. Zhang, and Y. Liu, “Masterkey: Automated jailbreaking of large language model chatbots,” in *NDSS*, 2024.

[55] X. Qi, K. Huang, A. Panda, P. Henderson, M. Wang, and P. Mittal, “Visual adversarial examples jailbreak aligned large language models,” *arXiv*, 2023.

[56] H. Li, D. Guo, W. Fan, M. Xu, J. Huang, F. Meng, and Y. Song, “Multi-step jailbreaking privacy attacks on chatgpt,” *arXiv*, 2023.

[57] X. Shen, Z. Chen, M. Backes, Y. Shen, and Y. Zhang, “"do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models,” *arXiv*, 2023.

[58] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson *et al.*, “Extracting training data from large language models,” in *Usenix Security*, 2021.

[59] N. Kandpal, M. Jagielski, F. Tramèr, and N. Carlini, “Backdoor attacks for in-context learning with language models,” in *ICML Workshop*, 2023.

[60] A. Wan, E. Wallace, S. Shen, and D. Klein, “Poisoning language models during instruction tuning,” in *ICML*, 2023.

[61] N. Carlini, D. Ippolito, M. Jagielski, K. Lee, F. Tramer, and C. Zhang, “Quantifying memorization across neural language models,” in *ICLR*, 2022.

[62] N. Carlini, F. Tramèr, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, Ú. Erlingsson, A. Oprea, and C. Raffel, “Extracting training data from large language models,” in *Usenix Security*, 2021.

[63] J. Mattern, F. Mireshghallah, Z. Jin, B. Schoelkopf, M. Sachan, and T. Berg-Kirkpatrick, “Membership inference attacks against language models via neighbourhood comparison,” in *Findings of ACL: EMNLP*, 2023.

[64] X. Pan, M. Zhang, S. Ji, and M. Yang, “Privacy risks of general-purpose language models,” in *IEEE S & P*, 2020.

[65] “Exploring prompt injection attacks,” https://research.nccgroup.com/2022/12/05/exploring-prompt-injection-attacks/.

[66] B. Biggio, B. Nelson, and P. Laskov, “Poisoning attacks against support vector machines,” in *ICML*, 2012.

[67] T. Gu, B. Dolan-Gavitt, and S. Garg, “Badnets: Identifying vulnerabilities in the machine learning model supply chain,” *IEEE Access*, 2017.

[68] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, “Trojaning attack on neural networks,” in *NDSS*, 2018.

[69] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, “Targeted backdoor attacks on deep learning systems using data poisoning,” *arXiv*, 2017.

[70] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, “Poison frogs! targeted clean-label poisoning attacks on neural networks,” in *NeurIPS*, 2018.

[71] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, “How to backdoor federated learning,” in *AISTATS*, 2020.

[72] M. Fang, X. Cao, J. Jia, and N. Gong, “Local model poisoning attacks to byzantine-robust federated learning,” in *USENIX Security Symposium*, 2020.

[73] Z. Zhang, J. Jia, B. Wang, and N. Z. Gong, “Backdoor attacks to graph neural networks,” in *SACMAT*, 2021.

[74] J. Jia, Y. Liu, and N. Z. Gong, “Badencoder: Backdoor attacks to pre-trained encoders in self-supervised learning,” in *IEEE S & P*, 2022.

[75] X. Li and J. Li, “Angle-optimized text embeddings,” *arXiv preprint arXiv:2309.12871*, 2023.

[76] “Whereisai/uae-large-v1,” https://huggingface.co/WhereIsAI/UAE-Large-V1.

[77] A. Kerckhoffs, “La cryptographie militaire,” *Journal des sciences militaires*, 1883.

[78] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, “Hotflip: White-box adversarial examples for text classification,” in *ACL*, 2018.

[79] J. Morris, E. Lifland, J. Y. Yoo, J. Grigsby, D. Jin, and Y. Qi, “Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp,” in *EMNLP*, 2020.

[80] D. Jin, Z. Jin, J. T. Zhou, and P. Szolovits, “Is bert really robust? a strong baseline for natural language attack on text classification and entailment,” in *AAAI*, 2020.

[81] J. Li, S. Ji, T. Du, B. Li, and T. Wang, "Textbugger: Generating adversarial text against real-world applications," in *NDSS*, 2019.

[82] L. Li, R. Ma, Q. Guo, X. Xue, and X. Qiu, "Bert-attack: Adversarial attack against bert using bert," in *EMNLP*, 2020.

[83] J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi, "Black-box generation of adversarial text sequences to evade deep learning classifiers," in *SPW*, 2018.

[84] "Ms-marco microsoft bing," https://microsoft.github.io/msmarco/.

[85] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing, "Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality," 2023.

[86] N. Carlini and A. Terzis, "Poisoning and backdooring contrastive learning," in *ICLR*, 2022.

[87] N. Carlini, "Poisoning the unlabeled dataset of semi-supervised learning," in *USENIX Security*, 2021.

[88] H. Liu, J. Jia, and N. Z. Gong, "Poisonedencoder: Poisoning the unlabeled pre-training data in contrastive learning," in *USENIX Security Symposium*, 2022.

[89] M. R. Rizqullah, A. Purwarianti, and A. F. Aji, "Qasina: Religious domain question answering using sirah nabawiyah," in *ICAICTA*, 2023.

[90] Y. Huang, S. Gupta, M. Xia, K. Li, and D. Chen, "Catastrophic jailbreak of open-source llms via exploiting generation," *arXiv*, 2023.

[91] Y. Du, A. Bosselut, and C. D. Manning, "Synthetic disinformation attacks on automated fact verification systems," in *AAAI*, 2022.

[92] Y. Pan, L. Pan, W. Chen, P. Nakov, M.-Y. Kan, and W. Y. Wang, "On the risk of misinformation pollution with large language models," in *EMNLP*, 2023.

[93] S.-Q. Yan, J.-C. Gu, Y. Zhu, and Z.-H. Ling, "Corrective retrieval augmented generation," *arXiv*, 2024.

[94] O. Yoran, T. Wolfson, O. Ram, and J. Berant, "Making retrieval-augmented language models robust to irrelevant context," in *ICLR*, 2023.

[95] H. Luo, T. Zhang, Y.-S. Chuang, Y. Gong, Y. Kim, X. Wu, H. Meng, and J. Glass, "Search augmented instruction learning," in *EMNLP*, 2023, pp. 3717–3729.

[96] T. Zhang, S. G. Patil, N. Jain, S. Shen, M. Zaharia, I. Stoica, and J. E. Gonzalez, "Raft: Adapting language model to domain specific rag," *arXiv*, 2024.

[97] J. Steinhardt, P. W. W. Koh, and P. S. Liang, "Certified defenses for data poisoning attacks," *NeurIPS*, 2017.

[98] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *IEEE S& P*, 2019.

[99] J. Jia, X. Cao, and N. Z. Gong, "Intrinsic certified robustness of bagging against data poisoning attacks," in *AAAI*, 2021.

[100] J. Jia, Y. Liu, X. Cao, and N. Z. Gong, "Certified robustness of nearest neighbors against data poisoning and backdoor attacks," in *AAAI*, 2022.

[101] J. Jia, Y. Liu, Y. Hu, and N. Z. Gong, "Pore: Provably robust recommender systems against data poisoning attacks," in *USENIX Security Symposium*, 2023.

[102] Y. Wang, W. Zou, and J. Jia, "Fcert: Certifiably robust few-shot classification in the era of foundation models," in *IEEE S & P*, 2024.

[103] S. Shan, A. N. Bhagoji, H. Zheng, and B. Y. Zhao, "Poison forensics: Traceback of data poisoning attacks in neural networks," in *USENIX Security Symposium*, 2022.

[104] F. Jelinek, "Interpolated estimation of markov source parameters from sparse data," in *Proc. Workshop on Pattern Recognition in Practice*, 1980.

[105] "Tiktoken," https://github.com/openai/tiktoken.

[106] J. Thorne, A. Vlachos, C. Christodoulopoulos, and A. Mittal, "Fever: a large-scale dataset for fact extraction and verification," *arXiv*, 2018.

[107] W. Zou, R. Geng, B. Wang, and J. Jia, "Poisonedrag: Knowledge corruption attacks to retrieval-augmented generation of large language models," *arXiv preprint arXiv:2402.07867*, 2024.

[108] E. Kortukov, A. Rubinstein, E. Nguyen, and S. J. Oh, "Studying large language model behaviors under realistic knowledge conflicts," *arXiv*, 2024.

Appendix. For space reasons, we put Appendix in our technical report [107] on arXiv.