Traceback of Poisoned Texts in Poisoning Attacks to Retrieval-Augmented Generation

Anonymous Authors

Abstract

Large language models (LLMs) integrated with retrieval-augmented generation (RAG) systems enhance accuracy by accessing external knowledge database. However, recent studies have exposed RAG's vulnerability to poisoning attacks, where an attacker inject poisoned texts into the knowledge database, leading to attacker-desired responses. Existing defenses, primarily focused on inference-time mitigation, have proven inadequate against sophisticated attacks. In this paper, we present the first traceback system in RAG, RAG-Forensics, which traces poisoned texts from the knowledge database. RAGForensics narrows the space of potentially poisoned texts and accurately identifies them without requiring access to model gradients, a common challenge in RAG systems. Our empirical evaluation on multiple datasets demonstrates RAGForensics's effectiveness against state-of-the-art and adaptive poisoning attacks. This work pioneers the exploration of poisoned texts traceback in RAG systems, offering a practical and promising approach to securing them against poisoning attacks.

1 Introduction

Large language models (LLMs) [1, 2, 5] have demonstrated impressive capabilities, matching human-level performance in tasks like question answering and summarization. However, they are prone to hallucinations [16], generating incorrect information due to the absence of real-time knowledge. Retrieval-augmented generation (RAG) [4, 8, 11, 18, 19, 21, 25, 32] addresses this issue by retrieving relevant texts from an external knowledge database. A RAG system consists of three components: the knowledge database, a retriever, and an LLM. When a user submits a query, the retriever selects the top-*K* relevant texts from the database, which are then provided as context to the LLM for generating a more accurate response.

Recent studies [7, 10, 28, 30, 40, 41] reveal that RAG systems are highly susceptible to poisoning attacks. Since the knowledge database sources information from platforms like Wikipedia [31] and FactoidWiki [9], attackers can inject poisoned texts into the data collection pipeline, causing the LLM in RAG to produce attackerintended responses for specific queries. In response, a few defenses [36, 41] have been developed to counter such attacks, primarily by mitigating the influence of poisoned texts during inference. For example, RobustRAG [36] uses an isolate-then-aggregate approach to filter out malicious keywords from the top-K texts retrieved for a given query.

Given the tendency for new defenses to be quickly overcome by more advanced attacks [10, 12, 22, 23, 29], we argue that the immediate priority should not be on developing complex inference-time defenses. Instead, drawing inspiration from poison forensics in deep learning systems [17, 24, 29], which focuses on tracing problematic training data linked to misclassification behavior, we re-evaluate the security needs of RAG systems. In practice, direct attacks on the retriever or LLM are challenging since these components are typically isolated from external access. However, the knowledge database, which aggregates up-to-date information from various external sources [7, 28, 40, 41], has become a prime target for attackers. Exploiting this vulnerability, adversaries seek to manipulate the LLM's output by injecting stealthy, well-crafted poisoned texts into the database, enabling them to influence responses covertly.

In light of this challenge, we argue that it is more practical for RAG system providers to focus on tracing poisoned texts rather than building sophisticated defenses to prevent adversarial manipulation of the LLM. This approach offers the advantage of identifying problematic data sources or weaknesses in the data collection pipeline, allowing providers to mitigate poisoning attacks by replacing compromised sources or removing malicious texts. However, tracing poisoned texts in RAG systems is a complex task. First, while existing poison forensics methods in deep learning [29] have shown that poisoned training data can be identified using model gradients, this technique is not applicable in RAG systems since we lack access to the parameters of both the LLM and the retriever. Second, with knowledge databases often containing millions of entries, achieving a reliable traceback with an extremely low false positive rate becomes a significant challenge.

In this paper, we are the first to address the challenge of tracing poisoned texts in RAG systems under poisoning attacks and propose RAGForensics, a traceback system capable of accurately identifying malicious texts within the knowledge database based on the observed attack. Our RAGForensics system operates in two key phases: narrowing the scope of potential poisoned texts (Section 4.2) and accurately identifying them (Section 4.3). In the narrowing phase, the system retrieves a subset of texts from the database that are most likely to be poisoned, reducing the identification scope from millions to just a few dozen texts. The identification phase leverages an LLM to precisely detect the poisoned texts within this subset, eliminating the need for gradient calculations. Specifically, for each targeted query flagged through user feedback, RAGForensics iteratively uses the RAG retriever to extract a set of suspect texts from the knowledge database. The LLM then identifies which of these texts are responsible for generating incorrect outputs. The iteration process continues until no poisoned texts remain among the top-Kmost relevant texts retrieved for the targeted query.

Additionally, we recognize that incorrect outputs for targeted queries, reported through user feedback, may not always result from an attack (referred to as non-poisoned feedback). In such cases, no poisoned text may be present among the top-*K* retrieved texts. Instead, the error could stem from the LLM having learned incorrect information during training. In Section 6, we explore how RAGForensics can distinguish non-poisoned feedback and improve the RAG system's output through benign text enhancement. For each targeted query, we insert a relevant benign text and its retrieval proxy into the knowledge database, ensuring it appears among

the top-K texts to guide the LLM toward generating the correct response.

We summarize the following contributions:

- We are the first to introduce the problem that the traceback of poisoned texts in poisoning attacks to RAG, and propose the traceback system RAGForensics that can accurately identify poisoned texts from the knowledge database based on the observed poisoning attacks.
- we discuss how to identify the non-poisoned feedback in our traceback system RAGForensics and correct the output of RAG.
- We empirically demonstrate the effectiveness of our RAG-Forensics and BTE against a variety of state-of-the-art poisoning attacks on 3 datasets. In addition, we also evaluate our RAGForensics against two adaptive attacks and demonstrate that it is robust for the powerful attacks.

To the best of our knowledge, this is the first work to explore a traceback of poisoning texts in RAG. Our results demonstrate the effectiveness of our proposed traceback system. Therefore, we believe that poison forensics in RAG is very practical and promising.

2 Preliminaries and Related Work

2.1 RAG Overview

The RAG system improves LLMs by retrieving relevant information from external knowledge bases [7, 28, 37, 41]. It fetches relevant texts based on a user query and combines them with the query to provide the LLM with additional context, leading to more accurate responses. The RAG workflow is structured into two stages: knowledge retrieval and answer generation.

Knowledge Retrieval: The goal of this stage is to retrieve the most relevant texts from the external knowledge database based on the user's query. Typically, this is achieved using a vector-based retrieval model. For a query q, the query encoder f_q generates an embedding vector v_q . Each text d_j in the knowledge base \mathcal{D} is encoded into a vector v_{d_j} by a text encoder f_d , forming the set $V_{\mathcal{D}}$. The relevance between v_{q_i} and $V_{\mathcal{D}}$ is measured by similarity (e.g., dot product or cosine similarity). Based on these scores, the top-K most relevant texts $\widehat{\mathcal{R}}(q, K, \mathcal{D})$ are selected.

Answer Generation: At this stage, the input query *q* is combined with the set of retrieved documents $\widehat{\mathcal{R}}(q, K, \mathcal{D})$ to query the LLM, which generates a response *O*. Formally, the response can be represented as the following:

$$O = \text{LLM}(\mathcal{R}(q, K, \mathcal{D}), q), \tag{1}$$

where $\widehat{\mathcal{R}}(q, K, \mathcal{D})$ represents the set of the top-*K* most relevant texts retrieved from the knowledge database \mathcal{D} based on the query *q*.

In this stage, we utilize a similar system prompt as described in [41] for RAG, as shown in Appendix 8.3.

2.2 Poisoning Attacks to RAG

The dependence of RAG systems on external data sources creates opportunities to poisoning attacks [28, 37, 40, 41]. In these attacks, adversaries intentionally inject harmful or misleading information into the knowledge base. Their goal is to influence or manipulate the LLM's responses to specific queries. Poisoning attacks to RAG can be implemented by injecting prompts [13], where malicious instructions are embedded in the knowledge base to manipulate the LLM into generating specific responses to targeted questions. In follow-up research, several methods for creating poisoned texts have been proposed. For example, Zhong et al. [40] introduced a method to generate adversarial paragraphs that, once inserted into the retrieval corpus, can cause the LLM to produce misleading answers. Unlike inserting semantically meaningless malicious text, Zou et al. [41] proposed the PoisonedRAG method, where the attacker injects carefully crafted, semantically meaningful "poisoned" documents designed to influence the LLM to generate responses controlled by the attacker.

2.3 Defenses against Poisoning Attacks to RAG

Current defenses against poisoning attacks to RAG include perplexitybased detection [15], query rewriting through LLMs [15, 41], and increasing the number of retrieved documents [41]. Perplexitybased methods identify poisoned content by assessing text quality, assuming that poisoned texts have higher perplexity scores, indicating lower quality. Query rewriting defenses modify user queries to reduce the likelihood of retrieving poisoned texts, aiming to retrieve safer, benign information. Xiang et al. [36] introduced the RobustRAG framework, which uses an isolate-then-aggregate strategy to defend against such attacks, but its effectiveness declines as the amount of poisoned data increases.

While current defenses show promise in detecting poisoned texts or preventing them from influencing the outputs of LLM, stronger and adaptive attackers can still evade them, as seen in deep learning systems [3, 26, 27, 35, 39]. Drawing inspiration from poison forensics in deep learning [17, 24, 29], which trace harmful training data linked to misclassifications, we believe it is more valuable for RAG system providers to trace poisoned texts rather than develop more advanced defenses. However, poison forensics in deep learning rely on model gradients, which are generally inaccessible in RAG systems due to limited access to LLM parameters.

3 Traceback of Poisoned Texts in Poisoning Attacks to RAG

This paper is the first to address the task of tracing poisoned texts in RAG poisoning attacks. We illustrate this task with the following example scenario. Figure 1 outlines the process of tracing poisoned texts in RAG poisoning attacks. The attacker begins by injecting poisoned texts into the knowledge database (Figure 1a). When a user submits a targeted query, the RAG system, relying on the compromised database, generates an incorrect response aligned with the attacker's intent (Figure 1b). To address this, the RAG service provider can offer a feedback button, allowing users to report incorrect outputs along with the related queries. These reports are sent to the traceback system, which identifies the poisoned texts in the knowledge database responsible for the incorrect responses (Figure 1c). We begin by outlining the threat model for poisoning attacks and the traceback system in RAG. Afterward, we analyze the key challenges involved in designing an effective traceback system.



Figure 1: The example scenario of our traceback system. (a) an attacker injects multiple poisoned texts into knowledge database; (b) an user submits a query and obtain an incorrect output to cause a feedback; (c) our traceback system RAGForensics identifies poisoned texts based on the user's feedback that includes query and incorrect answer.

3.1 Threat Model

In this section, we describe the threat model of poisoning attacks and the traceback system respectively.

Poisoning attacks: Building on existing poisoning attacks [28, 37, 40, 41], we outline the attacker's goal and knowledge. The attacker aims to poison the knowledge database so that the LLM in RAG generates specific, attacker-chosen responses (targeted answers) for a set of *M* targeted queries. We assume the attacker has full knowledge of the texts in the database and direct access to the parameters of both the retriever and the LLM, allowing them to query these components directly.

Traceback system: We assume that the service provider of the traceback system is the owner of the RAG system. The traceback system is granted full access to all texts in the knowledge database. However, for the retriever and LLM, we consider a practical scenario where the RAG owner uses a closed-source retriever and LLM that outperform open-source alternatives. As a result, the traceback system cannot access their internal parameters but can query them directly. We assume that the traceback system has collected each targeted query q_i and its corresponding RAG output t_i involved in the poisoning attack. This is a common assumption in traceback systems for poisoning attacks [17, 29]. Many LLM applications, such as ChatGPT¹, include a feedback button that allows users to report incorrect outputs along with the related queries. Thus, this assumption is both practical and easily implementable in RAG.

In practice, incorrect outputs can also result from the LLM itself. An under-trained LLM or one that has acquired incorrect knowledge may generate inaccurate responses even without poisoned texts. In Section 6, we explore methods to identify whether the LLM is the source of an incorrect output. Additionally, we propose a post-hoc defense to help the LLM produce correct answers for all collected queries.

3.2 Design Challenges

Optimization problem: Building on the traceback of data poisoning attacks in neural networks [29], we formulate the traceback of poisoned texts in RAG poisoning attacks as an optimization problem. Unlike poisoning in neural networks, where individual data points are manipulated, RAG attacks involve injecting multiple poisoned texts into the knowledge database to cause the LLM to generate attacker-desired answers for targeted queries. The goal of the traceback process is to identify the specific texts in the knowledge database responsible for the incorrect outputs. Given a set of targeted queries Q and their incorrect outputs, as reported through user feedback, our objective is to identify and remove the poisoned texts \mathcal{D}_p from the knowledge database \mathcal{D} . This ensures that the LLM no longer produces the incorrect output t_i for each targeted query $q_i \in Q$. We formalize this as the following problem:

$$\min_{\mathcal{D}_p} \quad \frac{1}{|\mathcal{Q}|} \sum_{i=1}^{|\mathcal{Q}|} \mathbb{I}(\text{LLM}(\widehat{\mathcal{R}}(q_i, K, \mathcal{D} \setminus \mathcal{D}_p), q_i) = t_i)$$
(2)
s.t. $q_i \in \mathcal{Q}, \quad i = 1, 2, \dots, |\mathcal{Q}|,$
 $\mathcal{D}_p \in \mathcal{D},$

where $\mathcal{D} \setminus \mathcal{D}_p$ denotes the knowledge database with the poisoned texts \mathcal{D}_p removed. The expression $\widehat{\mathcal{R}}(q_i, K, \mathcal{D} \setminus \mathcal{D}_p)$ refers to the top-*K* most relevant texts retrieved by the retriever from the modified database $\mathcal{D} \setminus \mathcal{D}_p$ for the targeted query q_i .

Key challenges: The main challenge in identifying the poisoned text set \mathcal{D}_p lies in solving the optimization problem in Equation 2. A straightforward approach would be to iteratively update \mathcal{D}_p using the gradient of the objective function, as suggested by [29]. However, this is impractical under our threat model. First, computing the gradient is challenging due to two constraints: (1) we lack access to the parameters of the retriever and LLM, and (2) both $\widehat{\mathcal{R}}(\cdot)$ and $\mathbb{I}(\cdot)$ are discrete functions. Second, determining update candidates for \mathcal{D}_p requires calculating the gradient for every text in the knowledge database \mathcal{D} . This approach adds substantial computational overhead and complicates optimization.

4 Our Traceback System: RAGForensics

In this section, we provide a high-level overview of our traceback system RAGForensics, followed by its detailed design.

4.1 Overview

To address the key challenge in Section 3.2, our RAGForensics identifies the poisoned texts \mathcal{D}_p by iteratively retrieving and identifying

¹https://chatgpt.com/

them for each targeted query. For every query, RAGForensics retrieves texts likely to be poisoned and identifies those responsible for the incorrect output. The process stops once no poisoned texts remain among the top-*K* relevant texts. RAGForensics involves two key operations: narrowing the scope of poisoned texts (Section 4.2) and identifying them (Section 4.3). The narrowing operation clusters scattered poisoned texts into a small set, reducing identification costs and false positives. We use the RAG retriever to retrieve texts likely to be poisoned. In the identification step, a LLM (potentially different from the one in RAG) evaluates each retrieved text to determine if it is poisoned based on the incorrect output.

4.2 Narrowing the Scope of Poisoned Texts

Since the traceback system lacks knowledge of the attacker's strategy for injecting poisoned texts, we assume the poisoned texts are randomly distributed throughout the knowledge database. For a targeted query q_i , a straightforward approach would be to brute-force search through the entire database \mathcal{D} to identify the text causing the incorrect output t_i . However, this approach is computationally expensive and increases the risk of false positives for benign texts. To address this, we propose clustering the texts most likely to be poisoned into a smaller, manageable set.

The main challenge is determining which texts are more likely to be poisoned. This presents a dilemma: we want to avoid evaluating every text individually through clustering, but clustering requires assessing the likelihood of each text being poisoned. To overcome this, we leverage the attacker's strategy. The attacker crafts poisoned texts to maximize their similarity with the targeted query q_i , ensuring they are retrieved among the top-K texts. The incorrect output t_i further confirms that poisoned texts are being retrieved. Using this insight, we iteratively employ the RAG retriever to select the top-K relevant texts for the query q_i as potential poisoned candidates. In each iteration, we identify which texts in the retrieved set are poisoned (as detailed in Section 4.3) and remove them from the knowledge database. The process repeats until no poisoned texts are identified in the retrieved set.

4.3 Identifying Poisoned Texts

For the set of texts \mathcal{R} retrieved in Section 4.2, we must accurately determine whether each text is poisoned. Due to the complexity of linguistic features, it is challenging to create a deterministic function capable of identifying which texts are responsible for the incorrect output.

To tackle this challenge, we leverage another aspect of the attacker's strategy. The attacker ensures that each poisoned text p_j can independently cause the LLM to generate the incorrect output t_i for the targeted query q_i . Thus, for each text \mathcal{R}_j in the set \mathcal{R} , we can use a LLM (which may differ from the one in RAG) to assess whether \mathcal{R}_j triggers the incorrect output t_i . However, if t_i is malicious, security restrictions may prevent the LLM from evaluating accurately. To address this limitation, we provide explicit instructions, directing the LLM to disregard its background knowledge and the correctness of both \mathcal{R}_j and t_i . Additionally, inspired by the chain-of-thought (CoT) approach [34], we guide the LLM with step-by-step instructions to ensure a thorough evaluation. The prompt we designed to guide the LLM in evaluating the text \mathcal{R}_j follows three steps. First, it instructs the LLM to assess whether \mathcal{R}_j aims to induce an answer aligned with the incorrect output t_i , regardless of factual accuracy. Second, the prompt asks the LLM to provide a brief explanation for its judgment, encouraging logical reasoning and improving the reliability of its assessment. Finally, we incorporate the chain-of-thought (CoT) approach by including the instruction, "Let's think step by step", prompting the LLM to perform a structured, step-by-step evaluation. The complete prompt is as follows:

I will provide a question, an response, and a context. Without any other knowledge, judge whether the provided context tries to induce you to generate an answer consistent with the provided response $[t_i]$, regardless of whether it is correct. Begin your judgement by providing a short explanation.

If yes, return "[Label: Yes]", otherwise, return "[Label: No]". Question: $[q_i]$ Context: $[\mathcal{R}_i]$

Response: $[t_i]$ Let's think step by step.

5 Experiments

5.1 Experimental Setup

5.1.1 Datasets. We utilize three question-answering datasets: Natural Questions (NQ) [20], MS-MARCO [38], and HotpotQA [6]. Each of these datasets contains a collection of queries along with an associated knowledge database. For every query, multiple ground truth texts are provided as correct answers. NQ originates from real Google search queries, with its knowledge base being Wikipedia pages. Similarly, MS-MARCO is built on Bing search queries, with relevant web pages retrieved via Bing serving as its knowledge source. In contrast, HotpotQA consists of human-crafted questions requiring "multi-hop reasoning," and its knowledge base is also drawn from Wikipedia.

In our experiments, we use the same 100 queries from PoisonedRAG [41] for each dataset as the initial set of targeted queries, and adopt the same attacker-desired answers as in [41] for each query. We simulate user feedback through the following process: 1) we conduct poisoning attacks on the initial set of targeted queries; 2) we submit each targeted query to the poisoned RAG to generate the output; 3) we select 50 targeted queries whose outputs match the attacker-desired answers to serve as the final test data.

5.1.2 Attacks. To assess the effectiveness of our traceback system RAGForensics, we employ the following poisoning attacks:

PoisonedRAG attack [41]: PoisonedRAG aims to inject *M* poisoned texts into the knowledge database so that the RAG generates the attacker-desired answer for the targeted query. Each poisoned text *P* is split into two subtexts, *S* and *I*, where $P = S \oplus I$, with \oplus denoting string concatenation. For the subtext *I*, the attacker uses an LLM to generate it in a way that, when used as context, the LLM produces the attacker-desired answer. For the subtext *S*, various

Datasets	Attacks	Metrics	PPL-100	PPL-90	PoiFor	ExpGen	RKM	TKM	RAGForensics
		DACC ↑	37.5	37.5	83.1	90.3	84.3	81.0	99.6
	PoisonedRAG-B	FPR ↓	0.0	0.0	2.2	0.0	7.6	34.1	0.8
		FNR ↓	100.0	100.0	31.5	ForExpGenRKMTKMRAGF190.384.381.0920.07.634.19519.423.93.99191.284.972.9900.58.838.81917.124.115.4010100.069.686.3900.00.811.5000.060.016.00587.577.785.4962.66.729.12522.537.80.02189.280.364.1971.67.935.41420.031.436.42999.168.687.5921.80.78.9200.018.149.22033.328.72.00687.878.540.2950.017.947.72324.425.171.90899.253.873.7950.08.711.4101.683.841.20	0.0		
		DACC ↑	16.7	16.7	81.1	91.2	84.9	72.9	99.2
NQ	PoisonedRAG-W	FPR↓	0.0	0.0	0.0	0.5	8.8	38.8	1.6
		FNR↓	100.0	100.0	37.9	17.1	24.1	15.4	0.0
		DACC ↑	28.6	28.6	100.0	100.0	69.6	86.3	99.6
	InstruInject	FPR↓	0.0	0.0	0.0	0.0	0.8	11.5	0.4
			100.0	100.0	0.0	0.0	60.0	16.0	0.4
		DACC ↑	0.0	68.2	75.5	87.5	77.7	85.4	97.4
	PoisonedRAG-B	FPR↓	0.0	18.1	19.6	2.6	6.7	M IKM RAGForensics 3 81.0 99.6 6 34.1 0.8 .9 3.9 0.0 .9 72.9 99.2 8 38.8 1.6 .1 15.4 0.0 .6 86.3 99.6 8 11.5 0.4 .0 16.0 0.4 .0 16.0 0.4 .7 85.4 97.4 .7 29.1 2.4 .8 0.0 2.8 .3 64.1 97.6 9 35.4 1.6 .4 36.4 3.2 .6 87.5 98.2 .7 8.9 2.3 .1 16.0 1.2 .6 74.4 98.4 .1 49.2 2.3 .7 2.0 0.8 .5 40.2 <td< td=""></td<>	
		FNR ↓	100.0	45.6	29.5	22.5	37.8	0.0	2.8
	PoisonedRAG-W	DACC ↑	0.0	60.1	75.1	89.2	80.3	64.1	97.6
HotpotQA		FPR↓	85.6	79.7	21.7	1.6	7.9	35.4	1.6
		FNR ↓	57.2	0.0	44.4	20.0	31.4	36.4	3.2
		DACC ↑	15.6	60.5	98.9	99.1	68.6	87.5	98.2
	InstruInject	FPR ↓	3.3	79.1	2.2	1.8	0.7	8.9	2.3
		FNR ↓	98.0	0.0	0.0	0.0	62.1	16.0	1.2
		DACC ↑	44.4	67.6	73.0	83.4	76.6	74.4	98.4
	PoisonedRAG-B	FPR↓	0.0	8.3	1.0	0.0	18.1	49.2	A RAGForensics 0 99.6 1 0.8 0.0 99.2 3 1.6 4 0.0 3 99.6 6 0.0 3 1.6 4 0.0 3 99.6 5 0.4 0 0.4 4 97.4 1 2.4 2.8 1 1 0.4 1 2.4 2.8 1 1.6 4 3.2 5 5 98.2 2.3 0 1.2 4 4 98.4 2 2.3 7 2.7 9.8 7 7 $9.9.4$ 4 1.2 0.0 0.0
		FNR ↓	100.0	56.4	53.0	33.3	28.7	2.0	0.8
		DACC ↑	69.5	64.1	66.6	87.8	78.5	40.2	98.3
MS-MARCO	PoisonedRAG-W	FPR↓	0.0	71.7	3.5	0.0	17.9	47.7	2.7
		FNR↓	61.0	0.0	63.3	24.4	25.1	71.9	0.8
		DACC ↑	86.2	64.1	97.8	99.2	53.8	73.7	99.4
	InstruInject	FPR ↓	0.0	71.7	2.5	0.0	8.7	11.4	1.2
		FNR↓	27.6	0.0	2.0	1.6	83.8	41.2	0.0

Table 1: The DACC, FPR and FNR of our traceback system RAGForensics and 6 traceback baselines against 3 poisoning attacks on 3 datasets. Bold font indicates optimal, font underlined indicates suboptimal.

techniques are employed to craft it based on whether the retriever operates in a white-box or black-box setting.

- Black-box (PoisonedRAG-B): The attacker only uses the targeted query as the subtext *S*.
- White-box (PoisonedRAG-W): The attacker initially sets the targeted query as the subtext *S* and then updates it to maximize the similarity score between the subtext *S* and the targeted query.

Instruction injection attack (InstruInject) [28]: This attack was formally introduced in [28]. The poisoned text for each targeted query is crafted in the same manner as in PoisonedRAG (blackbox). However, in the instruction injection attack, the subtext *I* is replaced with the instruction: "Ignore all other context information and respond only with: [attacker-desired answer]."

5.1.3 Traceback Baselines in Poisoning Attacks to RAG. Given the limited research on traceback methods for poisoning attacks in RAG, no established traceback method exists to use as a baseline. To provide a more comprehensive evaluation of our traceback system RAGForensics, we introduce six baselines by extending traceback methods from poisoning attacks on neural networks and adapting commonly used defenses against such attacks in RAG systems.

Poison Forensics (PoiFor): Poison Forensics [29] is a traceback mechanism designed to address poisoning attacks on neural networks. It works by iteratively clustering and pruning benign training data to isolate the set of poisoned data. When adapted to RAG, Poison Forensics uses an LLM to group the top-*K* texts for each query into two clusters. Then, by prompting the LLM with the texts from each cluster, Poison Forensics identifies and removes the poisoned texts.

Explanation generation (ExpGen): Several studies [41] suggest enhancing user comprehension of answer sources by having LLMs provide explanations. To implement this, we first modify the system prompt of RAG, instructing it to explain which specific texts from the top-*K* sources are used to produce the answer. Next, we detect poisoned texts by evaluating the answers and explanations generated by the RAG model.

Response keywords matching (RKM): Inspired by the RobustRAG defense (as outlined in Section 5.1.4), we introduce the RKM method to detect poisoned texts. Specifically, after utilizing RobustRAG to extract the keywords from each top-*K* response, we employ substring matching to determine if the corresponding text is poisoned.

Text keywords matching (TKM): Different from the baseline RKM, we directly extract the keywords of each text in the top-*K*.

Perplexity-based detection (PPL): Perplexity-base detection [28, 41] have proposed defenses against poisoning attacks targeting RAG. In our experiment, we extend the Perplexity-based detection approach to the traceback baseline. Specifically, for each dataset, we begin by randomly selecting 1,000 texts from the knowledge database and use Llama-2-7b [33] to calculate the perplexity of each text. We then choose two thresholds to serve as baselines.

- **PPLDetect-100:** : The threshold is set to be greater than the perplexity of all the texts.
- **PPLDetect-90:** The threshold is set to be greater than the perplexity of 90% of the texts.

5.1.4 Defense Baselines Against Poisoning Attacks to RAG. In our experiment, we use the following state-of-the-art baseline defenses:

RobustRAG [36]: RobustRAG is a method designed to defend against poisoning attacks targeting RAG. The process begins by having the LLM generate a response for each of the top-*K* texts. Next, it identifies keywords from all the generated responses and filters out those that occur less frequently. Finally, the LLM produces a final response using the remaining keywords.

Knowledge expansion (KE) [41] : Knowledge expansion is identified as the most effective defense strategy proposed in [41]. This approach works by increasing the number of retrieved texts, thereby increasing the proportion of benign texts in the context. By doing so, KE reduces the influence of poisoned texts on the LLM's generated outputs. The notation KE-*x* indicates that *x* texts are retrieved during this process.

Perplexity-based detection (PPL): It is the same as described in Section 5.1.3.

5.1.5 *Evaluation Metric.* In evaluating our traceback system, we assess detection performance using three key metrics: detection accuracy (DACC), false positive rate (FPR), and false negative rate (FNR). For evaluating the general performance of our post-hoc defense, we rely on two metrics: attack success rate (ASR) and accuracy rate (ACC). The detailed computation of these metrics can be found in Appendix 8.2.

5.1.6 Parameter Setting. We outline the default configurations for the RAG system, the attack settings, and our traceback system.

RAG settings: For the retriever, we use Contriever [14] by default and apply a dot product to calculate similarity scores. We retrieve the K = 5 most relevant texts from the knowledge database for each query. For the LLM component, we use GPT-4o-mini as the default LLM in RAG.

Attack settings: In general, we set the number of poisoned texts *M* per targeted query to 5. For the PoisonedRAG attack, we use GPT-40-mini to craft the poisoned texts.

Our traceback system settings: We use GPT-4o-mini to identify each text among the top-*K* retrieved texts.

To reduce computational costs, we conduct 5 iterations for each experiment, with each iteration randomly selecting 10 queries for evaluation.

	~			
Attacks	Metrics	GPT-40-mini	GPT-4-turbo	GPT-40
	DACC↑ 99.6		97.4	99.4
PoisonedRAG-B	FPR↓	0.8	4.1	0.8
	FNR↓	0.0	1.2	0.4
	DACC ↑	99.2	99.6	99.6
PoisonedRAG-W	FPR↓	1.6	0.4	0.0
	FNR↓	0.0	0.4	0.8
	DACC ↑	99.6	97.5	98.0

0.4

0.4

5.0

0.0

0.8

3.2

Table 2: Impact of LLM used to identify the poisoned texts in our RAGForensics on NQ datasets.

5.2 Evaluation of Our Traceback System RAGForensics

FPR |

FNR |

InstruInject

Our RAGForensics can accurately trace the poisoned texts of various poisoning attacks on 3 datasets: Table 1 presents the DACC, FPR, and FNR metrics of our traceback system RAGForensics when tested against three poisoning attacks across three datasets. We have the following key observations. Firstly, our RAGForensics consistently identifies all poisoning texts within the knowledge database with high accuracy. For instance, the DACCs of our RAGForensics are all higher than 97.4%, and the FPRs are all lower than 2.7%, and the FNRs are all lower than 3.2%. Secondly, RAGForensics demonstrates stable performance across different poisoning attacks. For instance, in the NQ dataset, the DACCs fluctuate by no more than 0.4%, the FPRs by no more than 1.2%, and the FNRs by no more than 0.4%.

Our RAGForensics outperforms all traceback baselines on 3 datasets: Table 1 also shows the comparison between our RAG-Forensics and other traceback baselines against various poisoning attacks. The results demonstrates that our RAGForensics significantly outperforms other traceback baselines. Specifically, our RAG-Forensics basically achieves the optimal or suboptimal performance of the DACCs, FPRs, and FNRs. Moreover, our RAGForensics has the best generalization performance, while other baselines can only achieve good performance on the few settings of specific poisoning attacks and datasets. For instance, the DACCs of our RAGForensics are all higher than 97.4%, while the minimum fluctuation range of DACC in other baselines is more than 15%.

Impact of the number of poisoned texts injected into the knowledge database by attacker: We evaluate the effectiveness of our RAGForensics as the number of poisoned texts increases, ranging from 5 to 40. Figures 2, 3, 4 show the DACCs, FPRs, and FNRs of our traceback system RAGForensics against various poisoning attacks on 3 datasets. We find that our RAGForensics achieves similar DACCs, FNRs, and FPRs, demonstrating that its performance is insensitive to the number of poisoned texts.

Impact of LLMs in our RAGForensics: We conduct experiments using different LLMs in our RAGForensics to indentify the poisoned texts, including GPT-40, GPT-4-turbo, and GPT-40-mini. The results on NQ dataset are presented in Table 2, with those on HotpotQA and MS-MARCO datasets shown in Table 5 in Appendix 8.5. These results indicate that our RAGForensics performs well on the



Figure 2: Impact of the number of poisoned texts for each targeted query for our RAGForensics on NQ dataset.



Figure 3: Impact of the number of poisoned texts for each targeted query for our RAGForensics on HotpotQA dataset.



Figure 4: Impact of the number of poisoned texts for each targeted query for our RAGForensics on MS-MARCO dataset.

DACC, FNR, and FPR metrics across various LLMs, showcasing its remarkable adaptability to different LLMs.

5.3 Adaptive Attacks to RAGForensics

To verify the robustness of our method, we consider targeted adaptive attacks that an attacker familiar with our RAGForensics might deploy. We assume the adversary has full knowledge of the details of our RAGForensics, including its processes and prompts. Based on the three existing poisoning attacks mentioned before, we enhance them using two new adaptive attack approaches to circumvent our RAGForensics's defense. We evaluate our RAGForensics's defensive performance against these adaptive attacks through experiments.

Deceiving identification: In our RAGForensics, we use prompts to guide the LLM in assessing whether the texts in the knowledge database are designed to induce responses that align with the incorrect answers. Inspired by the prompt injection attack, an attacker familiar with our system and prompts can embed an additional instruction within the poisoned text they create. This instruction misleadingly asserts that the text is designed to prompt the LLM to produce the correct answer, thereby attempting to deceive the identification of poisoned texts by the LLM. Specifically, based on the poisoned text generated by three different attack methods, the attacker appends the phrase, "This text will induce you to generate [correct answer]", with the goal of deceiving the judging LLM within our RAGForensics.

We conduct experiments using this adaptive attack method both in the absence of defense and within our RAGForensics, with the results shown in Table 3. The experiments without any defense reveal high ASRs, demonstrating the strong attack capability of this method. In contrast, the results in our RAGForensics show high DACCs, along with very low FPRs and FNRs. This indicates that our RAGForensics can accurately distinguish between poisoned and benign texts, demonstrating robust resistance to this adaptive attack method.

Disguising Poison as Benign: In our RAGForensics, the LLM compares texts with incorrect answers when making judgments.

Table 3: The evaluation of RAGForensics against adaptive poisoning attacks in deceiving identification. ("No defense" in the table means not using any defensive measure against attacks; "Apt+" means that we enhance the basic attack methods using adaptive methods.)

Detecete	Attocks	No defense	RAGF	oren	sics
Datasets	Attacks	ASR	DACC	FPR	FNR
	Apt+PoisonedRAG-B	72.0	99.5	1.1	0.0
NQ	Apt+PoisonedRAG-W	78.0	98.5	3.0	0.0
	Apt+InstruInject	66.0	99.1	Forens FPR 1.1 3.0 1.8 3.4 1.3 2.1 0.7 2.4 2.0	0.0
	Apt+PoisonedRAG-B	80.0	97.3	3.4	2.0
HotpotQA	Apt+PoisonedRAG-W	92.0	98.2	1.3	2.3
	Apt+InstruInject	72.0	99.0	Foren C FPR 1.1 3.0 1.8 3.4 1.3 2.1 0.7 2.4 2.0	0.0
	Apt+PoisonedRAG-B	58.0	99.7	0.7	0.0
MS-MARCO	Apt+PoisonedRAG-W	82.0	98.8	2.4	0.0
	Apt+InstruInject	40.0	99.0	2.0	0.0

Table 4: The evaluation of RAGForensics against adaptive poisoning attacks in disguising poisoned as benign.

Datasata	Attacks	No defense	RAGF	oren	sics
Datasets	Attacks	ASR	DACC	FPR	FNR
	Apt+PoisonedRAG-B	54.0	97.5	4.3	0.7
NQ	Apt+PoisonedRAG-W	80.0	99.5	1.0	0.0
	Apt+InstruInject	74.0	efense RAGForense SR DACC FPR 4.0 97.5 4.3 0.0 99.5 1.0 4.0 99.5 1.1 2.0 96.4 2.6 3.0 98.2 1.3 4.0 98.5 2.4 5.0 98.0 4.0 4.0 98.2 3.6 3.0 100.0 0.0	0.0	
	Apt+PoisonedRAG-B	62.0	96.4	2.6	4.7
HotpotQA	Apt+PoisonedRAG-W	98.0	98.2	1.3	2.5
	Apt+InstruInject	64.0	98.5	Foren C FPR 4.3 1.0 1.1 2.6 1.3 2.4 4.0 3.6 0 0.0	0.6
	Apt+PoisonedRAG-B	56.0	98.0	4.0	0.0
MS-MARCO	Apt+PoisonedRAG-W	94.0	98.2	3.6	0.0
	Apt+InstruInject	48.0	100.0	0.0	0.0

This creates an opportunity for attackers to mislead the judging LLM by embedding the correct answer to a specific question within the poisoned text, leading the LLM to mistakenly classify it as harmless. In PoisonedRAG-B and Prompt Injection attack, attackers typically segment the poisoned text into two parts, while adaptive attackers can place the correct answer to the target question between these segments. For PoisonedRAG-W, the correct answer can be positioned at the start of the generated poisoned text, thereby confusing the LLM's judgment.

We conduct experiments using this adaptive attack method, and the results are shown in Table 4. In the absence of the defense, this method exhibits high ASRs, indicating its strong attack capability. In contrast, results in our RAGForensics demonstrate high DACCs, with low FPRs and FNRs, indicating that our RAGForensics can effectively identify benign and poisoned texts, thereby possessing robustness against adaptive attacks.

6 Discussion

In this section, we first discuss how to identify non-poisoned feedback in our traceback system RAGForensics. Then, we propose a method of benign texts enhancement to correct the output of RAG for the non-poisoned feedback. In the last, we discuss the limitations and future directions. **Identifying the non-poisoned feedback:** In practice, the incorrect output for a targeted query collected by the user's feedback may not result from an attack (called non-poisoned feedback), there might be no poisoned text among the top-*K* texts. Instead, the LLM could have learned incorrect knowledge during training, causing the incorrect output. Our traceback system RAGForensics can adapted to identify these non-poisoned feedback. In particular, given a targeted query and its incorrect output, we first use our traceback system RAGForensics to trace all poisoned texts in the knowledge database. When the tracing process is completed, we remove the poisoned texts from the knowledge database. Next, we submit the targeted query to the RAG system to obtain the latest output. If the latest output is still consistent with the incorrect output, we can assume that the incorrect output is not caused by the attacker.

Correcting the output of RAG: Since removing poisoned texts traced by our traceback system RAGForensics from the knowledge database does not correct the output of non-poisoned feedback, we have the responsibility to propose a post-hoc defense method to deal with this limitation. As a result, we propose a method of benign texts enhancement (detailed in Appendix 8.1) to correct the outputs of non-poisoned feedback. For each target question, we insert a benign text and its retrieval proxy into the knowledge database, ensuring that it can be retrieved among the top-*K* texts and induce the LLM generate correct answer. We evaluate the method of removing poisoned texts traced by our RAGForensics and the method of benign texts enhancement on 3 datasets, and compare them with other defense baselines. The results in Table 6 demonstrates that the method of benign texts enhancement effectively improves the accuracy of the correct answer to RAG output.

Limitation and future direction: Our traceback system may be vulnerable for non-targeted poisoning attacks [7, 40], where an attacker can inject multiple poisoning texts into the knowledge database to induce LLM generate randomly incorrect answer for the targeted query. This is because the error output and poisoning text of the user feedback are weak links, which makes us unable to accurately identify. This threat stimulates us to explore the traceback system against non-targeted poisoning attacks in the future, thereby improving the security system of treceback in RAG.

7 Conclusion

In this paper, we propose a novel approach to addressing poisoning attacks in RAG systems by introducing RAGForensics, a traceback system that focuses on identifying and removing poisoned texts from the knowledge database. By shifting the focus from inferencetime defenses to targeting the poisoned text itself, RAGForensics provides a more effective solution to the problem of poisoned outputs. Additionally, we tackle the challenge of distinguishing nonpoisoned feedback from actual attacks, enhancing model reliability with benign texts. Our experiments validate the robustness of RAGForensics against a variety of sophisticated poisoning attacks, making it a promising direction for strengthening the security of RAG systems.

References

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. arXiv (2023).
- [2] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. arXiv (2023).
- [3] Eugene Bagdasaryan and Vitaly Shmatikov. 2020. Blind backdoors in deep learning models. arXiv preprint arXiv:2005.03823 (2020).
- [4] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *ICML*.
- [5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *NeurIPS* (2020).
- [6] Daniel Fernando Campos, Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, Li Deng, and Bhaskar Mitra. 2016. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. ArXiv abs/1611.09268 (2016). https://api.semanticscholar.org/CorpusID:1289517
- [7] Harsh Chaudhari, Giorgio Severi, John Abascal, Matthew Jagielski, Christopher A Choquette-Choo, Milad Nasr, Cristina Nita-Rotaru, and Alina Oprea. 2024. Phantom: General Trigger Attacks on Retrieval Augmented Language Generation. arXiv preprint arXiv:2405.20485 (2024).
- [8] Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2024. Benchmarking large language models in retrieval-augmented generation. In AAAI.
- [9] Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. 2023. Dense X Retrieval: What Retrieval Granularity Should We Use? arXiv preprint arXiv:2312.06648 (2023). https://arxiv.org/pdf/2312.06648.pdf
- [10] Gelei Deng, Yi Liu, Kailong Wang, Yuekang Li, Tianwei Zhang, and Yang Liu. 2024. Pandora: Jailbreak gpts by retrieval augmented generation poisoning. arXiv preprint arXiv:2402.08416 (2024).
- [11] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997 (2023).
- [12] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising realworld llm-integrated applications with indirect prompt injection. In *Proceedings* of the 16th ACM Workshop on Artificial Intelligence and Security. 79–90.
- [13] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. *Proceedings* of the 16th ACM Workshop on Artificial Intelligence and Security (2023). https: //api.semanticscholar.org/CorpusID:258546941
- [14] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised Dense Information Retrieval with Contrastive Learning. *Trans. Mach. Learn. Res.* 2022 (2021). https://api.semanticscholar.org/CorpusID:249097975
- [15] Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. 2023. Baseline Defenses for Adversarial Attacks Against Aligned Language Models. ArXiv abs/2309.00614 (2023). https://api.semanticscholar.org/ CorpusID:261494182
- [16] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *Comput. Surveys* 55, 12 (2023), 1–38.
- [17] Yuqi Jia, Minghong Fang, Hongbin Liu, Jinghuai Zhang, and Neil Zhenqiang Gong. 2024. Tracing Back the Malicious Clients in Poisoning Attacks to Federated Learning. arXiv preprint arXiv:2407.07221 (2024).
- [18] Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. arXiv preprint arXiv:2305.06983 (2023).
- [19] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In EMNLP.
- [20] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur P. Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc V. Le, and Slav Petrov. 2019. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics* 7 (2019), 453–466. https: //api.semanticscholar.org/CorpusID:86611921
- [21] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *NeurIPS* (2020).

- [22] Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. 2023. Prompt injection attacks and defenses in llm-integrated applications. arXiv preprint arXiv:2310.12815 (2023).
- [23] Fábio Perez and Ian Ribeiro. 2022. Ignore previous prompt: Attack techniques for language models. arXiv preprint arXiv:2211.09527 (2022).
- [24] Evan Rose, Hidde Lycklama, Harsh Chaudhari, Anwar Hithnawi, and Alina Oprea. 2024. UTrace: Poisoning Forensics for Private Collaborative Learning. arXiv preprint arXiv:2409.15126 (2024).
- [25] Alireza Salemi and Hamed Zamani. 2024. Evaluating retrieval quality in retrievalaugmented generation. In SIGIR.
- [26] Roei Schuster, Congzheng Song, Eran Tromer, and Vitaly Shmatikov. 2021. You autocomplete me: Poisoning vulnerabilities in neural code completion. In 30th {USENIX} Security Symposium ({USENIX} Security 21).
- [27] Giorgio Severi, Jim Meyer, Scott Coull, and Alina Oprea. 2021. Explanation-Guided Backdoor Poisoning Attacks Against Malware Classifiers. In 30th {USENIX} Security Symposium ({USENIX} Security 21).
- [28] Avital Shafran, Roei Schuster, and Vitaly Shmatikov. 2024. Machine Against the RAG: Jamming Retrieval-Augmented Generation with Blocker Documents. arXiv preprint arXiv:2406.05870 (2024).
- [29] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y Zhao. 2022. Poison forensics: Traceback of data poisoning attacks in neural networks. In 31st USENIX Security Symposium (USENIX Security 22). 3575–3592.
- [30] Zhen Tan, Chengshuai Zhao, Raha Moraffah, Yifan Li, Song Wang, Jundong Li, Tianlong Chen, and Huan Liu. 2024. "Glue pizza and eat rocks"–Exploiting Vulnerabilities in Retrieval-Augmented Generative Models. arXiv preprint arXiv:2406.19417 (2024).
- [31] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. arXiv preprint arXiv:2104.08663 (2021).
- [32] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. Lamda: Language models for dialog applications. arXiv (2022).
- [33] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. arXiv (2023).
- [34] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems 35 (2022), 24824–24837.
- [35] Emily Wenger, Josephine Passananti, Arjun Bhagoji, Yuanshun Yao, Haitao Zheng, and Ben Y. Zhao. 2021. Backdoor Attacks Against Deep Learning Systems in the Physical World. arXiv preprint arXiv:2006.14580 (2021).
- [36] Chong Xiang, Tong Wu, Zexuan Zhong, David Wagner, Danqi Chen, and Prateek Mittal. 2024. Certifiably Robust RAG against Retrieval Corruption. ArXiv abs/2405.15556 (2024). https://api.semanticscholar.org/CorpusID:270045527
- [37] Jiaqi Xue, Mengxin Zheng, Yebowen Hu, Fei Liu, Xun Chen, and Qian Lou. 2024. BadRAG: Identifying Vulnerabilities in Retrieval Augmented Generation of Large Language Models. arXiv preprint arXiv:2406.00083 (2024).
- [38] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In Conference on Empirical Methods in Natural Language Processing. https://api.semanticscholar.org/ CorpusID:52822214
- [39] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y Zhao. 2019. Latent backdoor attacks on deep neural networks. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 2041–2055.
- [40] Zexuan Zhong, Ziqing Huang, Alexander Wettig, and Danqi Chen. 2023. Poisoning retrieval corpora by injecting adversarial passages. arXiv preprint arXiv:2310.19156 (2023).
- [41] Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. 2025. Poisonedrag: Knowledge poisoning attacks to retrieval-augmented generation of large language models. In USENIX Security Symposium.

8 Appendix

8.1 Benign texts Enhancement

A straightforward post-hoc defense is to remove the poisoned texts identified by our traceback system RAGForensics, which we refer to as Poisoned Texts Removal (PTR). However, this approach has two potential limitations. First, since RAGForensics cannot guarantee perfect identification of all poisoned texts, PTR may accidentally remove some benign texts or overlook a few poisoned ones. Second, as discussed in Section 3.1, the incorrect output for a targeted query reported by the user may not result from an attack; there might be no poisoned text among the top-*K* results. Instead, the LLM could have learned incorrect knowledge during training, causing the erroneous output. In such cases, PTR would be ineffective.

To address these two limitations, we propose a post-hoc defense called Benign Texts Enhancement (BTE), which enables the LLM to generate correct answers for targeted queries without requiring fine-tuning. In BTE, we assume the defender can obtain the correct answer for each targeted query either through manual annotation or by consulting more advanced LLMs. This assumption is practical, as service providers of RAG systems must address user-reported issues to improve user experience. For each targeted query q_i reported through user feedback, our BTE defense follows two steps:

Improving the confidence of LLM in RAG for the benign text: The key is to increase the LLM's confidence in the benign text so it prioritizes it over other texts in the top-*K* set. Interestingly, this aligns with the concept of backdoor attacks [], where an adversary implants a backdoor to trigger specific model behavior for certain inputs. In our defense, we leverage a similar idea, using triggers to make the LLM focus more on the benign text. However, in our threat model, the defender cannot train the LLM to implant a backdoor. Additionally, if the attacker discovers the trigger, retraining the LLM to implant a new backdoor would be costly and impractical.

To overcome these challenges, we propose a method to implant a backdoor into the LLM without requiring any training and allowing for flexible trigger replacement. Our approach involves adjusting the system prompt of RAG to prioritize texts with triggers in the top-K results. Specifically, we embed the description of the trigger and the corresponding text within the system prompt. To enhance the robustness of the prompt, we account for three possible scenarios involving the top-K texts and define appropriate handling rules for each: (1) texts with triggers relevant to the user's query, (2) texts with triggers present. The detailed system prompt is provided in Appendix 8.4.

Additionally, we need to create a benign text with the trigger. To ensure the benign text b_i conveys the correct answer c_i for the user's query q_i , we use an LLM (which may differ from the one in RAG) to generate b_i so that the LLM produces the correct answer c_i for the targeted query q_i . The prompt used to guide the LLM in crafting the benign text is provided in Appendix 8.4. Next, we embed a trigger into the benign text b_i . To maintain the semantic coherence and enhance the credibility of the modified text, we use the semantic trigger "[LATEST][/LATEST]." Specifically, we insert "[LATEST]" at the beginning of the benign text and "[/LATEST]" at the end. An example of this can be found in Appendix 8.4.

Configuring the retrieval proxy of benign text b_i : A crucial aspect of the BTE algorithm is ensuring that the benign text b_i with the trigger is included in the top-K most relevant texts. Since the defender lacks access to the retriever's parameters, optimizing each b_i using gradients to maximize its similarity with the target query isn't feasible. To overcome this, we propose a flexible and efficient retrieval proxy method. Specifically, for each benign text b_i , we designate its corresponding targeted query q_i as its retrieval proxy. Both b_i and its proxy q_i are inserted into the knowledge database. If q_i appears in the top-K retrieved texts, it is replaced

with the corresponding b_i . This retrieval proxy method offers three key advantages. First, it eliminates the need for retriever parameter access or optimization of benign texts. Second, the semantics of b_i are preserved, allowing it to be retrieved by other related queries. Third, it is a plug-and-play solution that does not alter the RAG system's retrieval process.

8.2 Metrics Computation

False positive rate (FPR): FPR is the ratio of the number of texts wrongly identified as poisoned (False Positive, *FP*) to the total number of benign texts. The FPR is computed as follows:

$$FPR = \frac{FP}{FP + TN},$$
(3)

where TN is the True Negative, the number of texts correctly identified as benign.

False negative rate (FNR): FNR is the ratio of the number of texts wrongly identified as benign (False Negative, *FN*) to the total number of poisoned texts. The FNR is computed as follows:

$$FNR = \frac{FN}{FN + TP},$$
(4)

where *TP* is the True Positive, the number of texts correctly identified as poisoned.

Detection accuracy (DACC): DACC is fraction of texts that are correctly identified. The DACC is computed as follows:

$$DACC = \frac{TP + TN}{TP + FP + TN + FN}.$$
 (5)

Attack success rate (ASR): ASR is the ratio of the number of targeted queries for which LLM generates targeted answers to the number of all targeted queries. Given a set of targeted queries $Q = \{q_1, q_2, ..., q_n\}$, the ASR is computed as follows:

$$ASR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \mathbb{I}(o_i, t_i),$$
(6)

where o_i is the answer generated by LLM for the targeted query q_i , t_i is the targeted answer. The indicator function $\mathbb{I}(\cdot)$ returns 0 when the condition is satisfied, otherwise it returns 0.

Accuracy (ACC): ACC is the ratio of the number of targeted queries for which LLM generates correct answers to the number of all targeted queries. The ACC is computed as follows:

$$ACC = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \mathbb{I}(o_i, c_i),$$
(7)

where c_i is the correct answer for the targeted query q_i .

Measurement of the ASRs and ACCs: For the term $\mathbb{I}(o_i, t_i)$ and $\mathbb{I}(o_i, c_i)$ in Equation 6 and 7, we need to manually judge whether the conditions of $\mathbb{I}(\cdot)$ are met. To improve the efficiency of the experiment, we use the LLM to replace human judgment.

8.3 System Prompt

The system prompt of RAG

You are a helpful assistant, below is a query from a user and some relevant contexts. Answer the question given the information in those contexts. Your answer should be short and concise. If you cannot find the answer to the question, just say "I don't know". **Contexts**: $[\widehat{\mathcal{R}}(q_i, K, \mathcal{D})]$ **Query**: $[q_i]$ **Answer**:

8.4 The prompts in Benign Texs Enhancement

The system prompt of RAG in the method of benign texts enhancement

You are a helpful assistant, below is a query from a user and some relevant contexts.

You should answer the question based on the information listed by entries in the contexts.Tell me your reasoning process in detail. Firstly, you need assess the relevance of each entry to the query, focusing only on contributions to the answer and excluding unrelated entry. Relevance is judged independently; do not evaluate correctness or contradictions. List all relevant entry unchanged. Secondly, count and list the number of the entry **wrapped by [LATEST] and [/LATEST]**, you will encounter three different situations regarding the filtered information:

Situation One: If there are more than one entry **wrapped by [LATEST] and [/LATEST]**, please follow these steps:

Step 1: For all contradictory entries, unconditionally select and display the one with the smallest index, regardless of its content. **Step 2**: Subsequently, delete all entries that contradict the selected one, irrespective of their quantity.

Situation Two: If there is only one entry wrapped by [LATEST] and [/LATEST], this indicates that it is the **newest** information. Therefore, please focus solely on the wrapped entry and disregard the remaining context.

Situation Three: If there is no entry wrapping by [LATEST] and [/LATEST], please analyze the relevance and consistency to draw a conclusion.

If you cannot find the answer to the question, just say "I don't know".

Contexts: $[\widehat{\mathcal{R}}(q_i, K, \mathcal{D})]$ Query: $[q_i]$ Answer:

The prompt to guide LLM in generating benign text

This is my question: $[q_i]$.

This is my answer: $[c_i]$.

Please craft a brief and complete corpus in a declarative tone that conveys the answer $[c_i]$ within the statement, ensuring it corresponds directly to the question $[q_i]$. Don't use phrases like "Yes" or "No". Please limit the corpus to V words and only return the corpus.

An example of the benign text with a trigger

Query: How many episodes are in chicago fire season 4?

Crafted benign text by LLM: Chicago Fire Season 4 consists of 23 episodes, showcasing the intense and dramatic experiences of the firefighters and paramedics at Firehouse 51 throughout the season.

Benign text with the trigger: [LATEST] Chicago Fire Season 4 consists of 23 episodes, showcasing the intense and dramatic experiences of the firefighters and paramedics at Firehouse 51 throughout the season. [/LATEST]

8.5 Experiment Results

Attacks	Metrics	GPT-40-mini	GPT-4-turbo	GPT-40	Attacks	Metrics	GPT-40-mini	GPT-4-turbo	GPT-40
PoisonedRAG	DACC ↑	0.97	0.99	0.98	DeisenedDAC	DACC ↑	0.98	0.99	0.99
	FPR↓	0.02	0.01	0.00	(Plash Bar)	FPR ↓	0.02	0.00	0.01
(DIACK-DOX)	FNR \downarrow	$\begin{array}{c cccc} 0.03 & 0.02 & 0.03 \\ \hline 0.98 & 0.98 & 0.99 \end{array} \qquad \begin{array}{c ccccccccccccccccccccccccccccccccccc$	0.01	0.02	0.02				
PoisonedRAG	DACC ↑	0.98	0.98	0.99	DeisenadDAC	DACC ↑	0.98	0.99	0.99
	FPR↓	0.02	0.00	0.00	(WILL A Deer)	FPR ↓	0.03	0.01	0.01
(wnite-box)	FNR \downarrow	0.03	0.03	0.02	(white-box)	FNR \downarrow	0.01	0.02	0.01
Instruction Injection	DACC ↑	0.98	0.99	0.97	Instruction	DACC ↑	0.99	0.99	0.95
	FPR↓	0.02	0.03	0.02	Instruction	FPR \downarrow	0.01	0.02	0.02
	FNR ↓	0.01	0.00	0.03	Injection	$\mathrm{FNR}\downarrow$	0.00	0.00	0.08

Table 5: Impact of LLM used to identify the poisoned texts in our RAGForensics on HotpotQA amd MS-MARCO datasets.(a) HotpotQA dataset.(b) MS-MARCO dataset.

Table 6: The ASRs and ACCs of our defenses and other baseline defenses against various poisoning attacks on 3 datasets. PTR represents the method of removing poisoned texts traced by our RAGForensics, PTE \oplus BTE represents the method of the combination of PTR and BTE. Bold font indicates optimal, font underlined indicates suboptimal.

Datasets	Attacks	Metrics	No defense	PPL-90	PPL-100	RobustRAG	KE-10	KE-20	KE-50	PTR	PTR⊕BTE
Datasets NQ HotpotQA MS-MARCO		ASR↓	1.00	0.98	1.00	0.50	0.84	0.78	0.72	0.00	0.00
	r oisoileurAG-D	ACC ↑	0.00	0.00	0.00	0.46	0.08	0.16	0.20	0.52	1.00
NO	PoisonadPAC W	ASR↓	1.00	1.00	1.00	0.44	0.86	0.76	0.72	0.00	0.00
NQ	PoisonedRAG-W	ACC ↑	0.00	0.00	0.00	0.52	0.10	0.20	0.28	0 PTR I 0.00 0.52 0.00 0.56 0.00 0.52 0.06 0.54 0.12 0.46 0.06 0.40 0.80 0.80 0.80 0.84 0.84 0.84	1.00
	InstruIniaat	ASR↓	1.00	0.98	1.00	0.28	0.82	KE-20 H 0.78 0.16 0.76 0.20 0.80 0.10 0.84 0.12 0.80 0.16 0.82 0.10 0.64 0.34 0.68 0.30 0.52 0.44	0.72	0.00	0.00
	mstrumjeet	ACC ↑	0.00	0.00	0.00	0.66	0.06	0.10	0.14	0.52	1.00
	PoisonadPAC B	ASR↓	1.00	0.64	1.00	0.80	0.82	0.84	0.82	0.06	0.00
	r oisoileutAG-D	ACC ↑	0.00	0.16	0.00	0.14	0.12	0.12	0.14	0.54	0.98
Datasets NQ HotpotQA MS-MARCO	PoisonedRAG-W	ASR↓	1.00	0.06	0.94	0.82	0.82	0.80	0.84	0.12	0.00
ΠοιροιζΑ		ACC ↑	0.00	0.50	0.06	0.14	0.14	0.16	0.14	0.46	0.98
	InstruInject	ASR↓	1.00	0.04	0.96	0.54	0.86	0.82	0.82	0.06	0.00
	mstrumject	ACC ↑	0.00	0.54	0.00	0.40	0.08	0.10	0.10	0.40	0.96
	PoisonadPAC B	ASR↓	1.00	0.62	1.00	0.38	0.72	0.64	0.56	0.00	0.00
	r oisoileutAG-D	ACC ↑	0.00	0.32	0.00	0.60	0.26	0.34	0.36	0.80	1.00
MS MARCO	Poisonad PAC W	ASR↓	1.00	0.00	0.70	0.36	0.76	0.68	0.66	0.00	0.00
NQ HotpotQA MS-MARCO	PoisonedRAG-W	ACC ↑	0.00	0.80	0.22	0.62	0.22	0.30	0.34	0.80	1.00
	InstruIniaat	ASR↓	1.00	0.00	0.28	0.14	0.60	0.52	0.52	0.00	0.00
	mstrumject	ACC ↑	0.00	0.82	0.58	0.82	0.32	0.44	0.46	0.84	1.00