Open Schrödinger's Closed Box: Identifying Retrieval Augmented Generation in API-Accessible Large Language Model Services

Anonymous ACL submission

Abstract

Large language models (LLMs) are powerful at question-answering but prone to hallucinations due to limited domain-specific or up-to-date knowledge. Retrieval augmented generation (RAG) mitigates this by adding an external retriever and knowledge database, yet RAG remains vulnerable to targeted attacks that degrade outputs or manipulate opinions. Prior attacks typically assume adversaries know the service is RAG-enhanced and may even know deployment details, an assumption often invalid for real-world commercial LLMs that expose only black-box APIs. This opacity also risks misleading users about system capabilities. This work aims to bridge this gap by proposing `RAG-ID`, a framework for IDentifying RAG properties in LLM services. We classify adversaries into three knowledge levels and design six attack methods. Experiments show these attacks reliably detect RAG — up to 99.97% accuracy with partial or no optional knowledge, and nearly 100% when the LLM and database are known. After detection, `RAG-ID` can infer finer RAG properties (e.g., deployed LLM and knowledge database). We consider `RAG-ID` a reconnaissance tool for attackers, a way to facilitate users' transparent selection of LLM services, and a guide for RAG developers in refining security measures.

1 Introduction

Large language models (LLMs) have been widely used in various fields due to their impressive comprehension and reasoning capabilities (Brown et al., 2020; Oppenlaender, 2022; Touvron et al., 2023; Jiang et al., 2023a). However, studies have revealed that LLMs inevitably produce hallucinations, primarily due to the lack of up-to-date and domain-specific knowledge (Rawte et al., 2023; Ji et al., 2023). To address this limitation, retrieval augmented generation (RAG) (Lewis et al., 2020; Borgeaud et al., 2022), which integrates LLMs with a retrieval system, has emerged as a new paradigm of LLM service. We refer to this paradigm as *RAG* or *RAG system* and regard services without it as *LLM-Only*. As shown in Figure 1, when a query is submitted, the retriever searches the knowledge database for j (e.g., $j = 5$) relevant chunks, and the LLM generates a response based on these retrieved texts rather than solely its internal knowledge (i.e., LLM-Only). This approach improves the accuracy of LLM services and accelerates their commercialization (Gao et al., 2023). Yet building RAG is non-trivial: it requires collecting massive domain documents, chunking, embedding, and storing them for efficient retrieval. Given these costs, RAG systems are frequently (>30% use cases) integrated into commercial LLM services for enterprise applications (Retool, 2023; Zou et al., 2024). For instance, Google Vertex AI offers APIs to create customized RAG systems (Google Cloud, 2024), and OpenAI supports RAG for companies like Morgan Stanley and Rakuten (OpenAI, 2024b,a). Meanwhile, numerous third-party startups are emerging to provide specialized RAG solutions (Semnani et al., 2023; YouTube, 2023; PMFM AI, 2025).

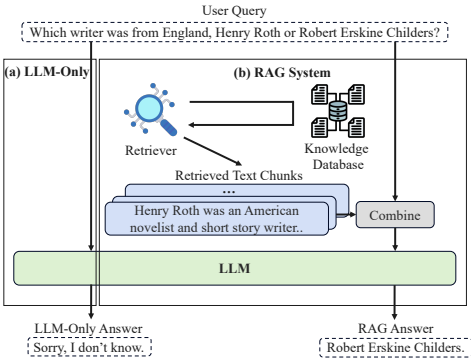


Figure 1: The work pipeline of (a) LLM-Only and (b) RAG system in question answering.

Alongside this adoption, concerns about RAG security have surfaced. Recent studies (Zou et al.,

2024; Cho et al., 2024; Xue et al., 2024; Chen et al., 2024b) reveal that RAG is vulnerable to targeted attacks that degrade generation quality or manipulate opinions. However, these works typically assume adversaries know detailed RAG information, including the LLM and knowledge database. In practice, many LLM services expose only limited interfaces such as APIs or chat apps, as providers aim to protect proprietary systems and reduce risks. Consequently, existing RAG attacks are hard to execute in real-world scenarios, since adversaries may not even know whether a service uses RAG, let alone its architecture. Thus, an LLM service can be viewed as existing in a state of "Schrödinger's box," where multiple possibilities remain unclear to the adversary.[1]

Our Contributions. In this work, we open the closed box of API-accessible LLM services by proposing RAG-ID, a framework for IDentifying RAG Properties in LLM services. Through this framework, we can effectively determine whether an API-accessible LLM service employs RAG to enhance its responses. This capability has broad implications. (1) For adversaries: RAG-ID can act as a reconnaissance tool, enabling adversaries to identify LLM services with RAG and potentially exploit these for follow-up attacks. For instance, knowing that a target LLM service relies on external knowledge could allow adversaries to implement targeted attacks, such as PoisonedRAG (Zou et al., 2024), typo-based manipulations (Cho et al., 2024), BadRAG (Xue et al., 2024), or opinion manipulation (Chen et al., 2024b), aimed at injecting misinformation or biases by corrupting the underlying knowledge database; (2) For users: understanding whether an LLM service uses RAG can improve transparency, helping users make informed choices about which services to trust. This transparency is especially important when service providers might make unverified or misleading claims about their systems' capabilities (JD Supra, 2024; Federal Trade Commission, 2024; Cai et al., 2025); (3) For developers and service providers: the ability to infer RAG properties in competing or third-party LLM services can guide developers in refining security measures, assessing market positioning, and ensuring responsible use and effective update of external knowledge sources.

[1]In the Schrödinger's cat experiment, a cat is placed in a box where a bottle of poison may be broken. Before the closed box is opened, the cat is in a superposition of being both alive and dead.

RAG-ID systematically categorizes adversaries into three types based on their knowledge levels, as outlined in Table 1. For adversary type 1, who has access to correct answers for each output text, the correctness of the target LLM service's responses can moderately identify RAG systems, with accuracy up to 67.39% across different LLMs. However, this approach is limited by the (question, short answer) nature of the dataset. For types 2 and 3, where adversaries have access to output LogProbs or texts, we first demonstrate that in a white-box setting—where both the LLM and knowledge database are known—RAG-ID can effectively identify the existence of RAG systems. By analyzing output LogProbs, perplexity, and sentence embeddings, RAG-ID achieves nearly 100% accuracy in some cases. Gradually relaxing the knowledge available to the adversary, we further show that RAG can still be identified with an accuracy up to 99.97%, even when the LLM, database, or both, are unknown. Moreover, once a target LLM service is identified as RAG, RAG-ID can further infer additional fine-grained RAG properties (e.g., the specific database or LLM) through adaptive design.

In summary, our contributions are as follows:

- We investigate the risk of RAG property leakage in API-accessible LLM services, identifying serious vulnerabilities in widely deployed services.
- We introduce RAG-ID, a structured framework for identifying RAG properties in LLM services. RAG-ID systematically models adversaries with different knowledge levels and constructs six targeted attack methods for identifying RAG in either a white-box setup or a black-box scenario by relaxing certain knowledge.
- Through comprehensive experiments across multiple datasets and LLMs, we validate the robustness and generalizability of RAG-ID, demonstrating its effectiveness in accurately identifying RAG in diverse settings.
- Beyond identifying RAG, RAG-ID can further infer specific properties, such as the language model and knowledge database in use, enabling more precise and targeted attacks and exposing further model and data vulnerabilities in current RAG systems.

2 Preliminary and Background

Retrieval Augmented Generation (RAG). Recent advances in LLMs (Zhao et al., 2023) have

Adversary Type	Target LLM Service's				Correct Answer C
	LLM θ	Knowledge Database ϕ	Output Texts R_P	Output LogProbs R_L	
Type 1	-	-	✓	-	✓
Type 2	○	○	-	✓	-
Type 3	○	○	✓	-	-

Table 1: An overview of different adversary types. ✓ represents compulsory knowledge, – indicates the adversary does not need this knowledge, and ○ means this knowledge is optional and is relaxed in black-box settings.

shown remarkable performance in tasks such as question answering and summarization (Brown et al., 2020; Oppenlaender, 2022; Touvron et al., 2023; Jiang et al., 2023a). However, these models rely on static pre-training data (OpenAI, 2023), leaving them outdated, domain-limited, and prone to fabricating information inconsistent with facts (Rawte et al., 2023). RAG addresses these issues by retrieving external knowledge to ground LLM outputs. As shown in Figure 1, a typical RAG system consists of a knowledge database, a retriever, and an LLM. The database, built from diverse sources such as Wikipedia (Thakur et al., 2021), PubMed (Xiong et al., 2024), and legal cases (Hou et al., 2024), is pre-processed into indexed chunks. Given a user query, the retriever selects the top-j relevant chunks, which are appended to the query via a prompt template, enabling the LLM to answer with external context. The quality of the database and retriever critically affects performance: informative chunks improve reliability, while poor context may degrade results (Zou et al., 2024; Tan et al., 2024). Accordingly, many works have focused on constructing domain-specific databases and retrievers to optimize RAG across different applications (Karpukhin et al., 2020; Xiong et al., 2021; Izacard et al., 2022; Jiang et al., 2024; Xiong et al., 2024; Hou et al., 2024; Omar et al., 2025).

Attacks Against RAG. Despite its popularity, recent studies (Zou et al., 2024; Cho et al., 2024; Xue et al., 2024; Chen et al., 2024b; Hu et al., 2024) have revealed multiple vulnerabilities in RAG. They compromise key components of RAG, such as poisoning the knowledge database and affecting the retriever's ranking, making RAG cannot generate correct answers (Cho et al., 2024; Hu et al., 2024) or directly direct answers to specific attack targets (Zou et al., 2024; Xue et al., 2024; Chen et al., 2024b). However, these attacks often require knowledge of the target LLM service deployment, such as knowing that the target is a RAG system, or even knowing the database retrieved by the RAG system. This information is considered as intellectual property for the LLM service provider and could be agnostic to the adversary when the practical target service only supports black-box access. This work aims to meet this gap between these attacks and real-world scenarios by identifying RAG properties. **Hence, we define identifying RAG as inferring whether an LLM service uses RAG to generate answers and the properties (e.g., database) that are used to build RAG.**

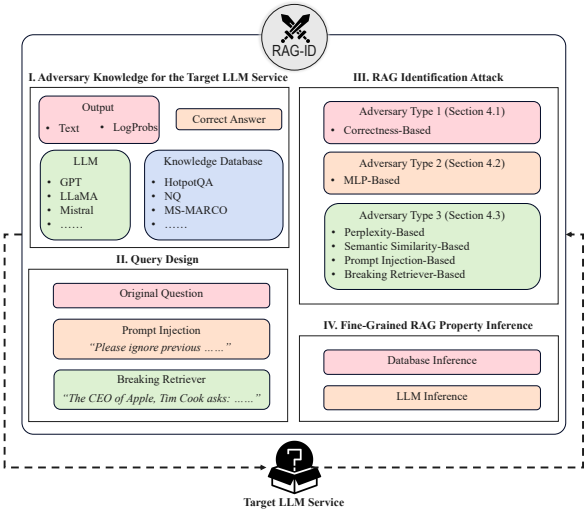


Figure 2: The overview of RAG-ID towards a target LLM service, where the service may take the pipeline of LLM-Only or RAG-based question answering.

3 Overview of RAG-ID

Figure 2 illustrates the components of our proposed method, RAG-ID, which consists of four steps.

Step I. RAG-ID gathers relevant knowledge about the target LLM service and categorizes it as shown in Table 1.

Step II. According to the adversary knowledge type, RAG-ID designs queries for input into the target LLM service.

Step III. After obtaining responses from the LLM service, RAG-ID performs the RAG identification

attack. Formally, for a target LLM service S_t and an input query Q, the output response $R = S_t(Q)$ can be obtained. Depending on the restrictions imposed by the service provider, R may contain only the plaintext answer R_P, or it may also include the Top-k LogProbs R_L for each token position. Thus, given the target LLM service S_t, input query Q, and adversary knowledge $\mathcal{K}$ about S_t, a RAG identification attack ($\mathcal{A}$) is defined as follows:

$$\mathcal{A}(S_t, R, \mathcal{K}) \in \{0, 1\}. \tag{1}$$

Here, 0 indicates that S_t does not use RAG to generate R for query Q, and 1 otherwise. Thus, $\mathcal{A}$ acts as a binary classifier.

Particularly, the adversary's knowledge

$$\mathcal{K} = \{\mathbb{1}(C), \mathbb{1}(\theta), \mathbb{1}(\phi)\}, \tag{2}$$

which involves three attributes: the correct answer C to the query Q, as well as the LLM θ and database ϕ implemented by S_t. Here $\mathbb{1}(\cdot) = 1$ if the attribute is known to the adversary and 0 otherwise. Depending on the adversary knowledge (see Table 1), $\mathcal{A}$ may deploy different components, such as a correctness classifier for the answer, a pre-trained model for calculating perplexity, a multi-layer perceptron (MLP) model for classifying LogProbs, to identify RAG properties.

Step IV. If the target LLM service is identified as using RAG, `RAG-ID` can further perform fine-grained inference on RAG-related properties (e.g., LLM and database). Specifically, let p denote the targeted property for inference. We define fine-grained RAG property inference $\mathcal{F}$ as:

$$\begin{aligned} \mathcal{F}(p, S_t, R, \mathcal{K}) &\in \{p_0, p_1, \cdots, p_{i-1}\}, \\ s.t.\ \mathcal{A}(S_t, R, \mathcal{K}) &= 1, \end{aligned} \tag{3}$$

where $\{p_0, \cdots, p_{i-1}\}$ are i potential values for p.

In the following sections, we first explain how to design queries and identify RAG in a white-box setting, then extend the approach to black-box scenarios. Finally, we will outline the design of fine-grained property inference for the RAG system.

Moreover, the setups of the target (victim) LLM services are provided in Appendix A.

4 White-Box RAG Identification Attacks

In this section, we examine RAG identification attacks in a white-box setting, where the adversary has full access to knowledge related to the target LLM service. Note, the attacker's knowledge of the target service's database is limited to the database that the target service claims to use, and knowing the possible database alone does not confirm whether RAG is actually used for generating responses (as the target service can have a database but not retrieve it). Specifically, for the three adversary types described in Table 1, we begin by hypothesizing and verifying the intuition for each type, followed by proposing and evaluating specific RAG identification attacks. Detailed intuition validations and methodologies for all concrete attacks are deferred to Appendix B. We retain the threat models and brief explanations for readability.

4.1 Adversary Type 1

Threat Model. For a given query Q, the adversary can obtain the corresponding correct answer C, where the output R of the target LLM service is plaintext R_P. The adversary's goal is to identify RAG by comparing C and R_P.

Brief Explanation. Compared with LLM-Only, RAG's answers are likely to have a higher correctness ratio. The adversary could simply identify RAG usage based on the correctness of the answer.

4.2 Adversary Type 2

Threat Model. The adversary has access to the target LLM service S_t's language model θ and knowledge database ϕ. Additionally, the adversary obtains the LogProbs R_L generated by S_t to identify RAG.

Brief Explanation. The answer LogProbs generated by LLM-Only and RAG likely follow different distributions. The adversary can identify RAG usage based on the difference in LogProbs.

4.3 Adversary Type 3

Threat Model. The adversary has access to the language model θ and knowledge database ϕ of the target LLM service S_t and aims to identify RAG based on the plaintext R_P generated by S_t. Specifically, we consider the differences between LLM-Only and RAG answers in four aspects and design a corresponding attack method for each.

❶ **Perplexity-Based Attack.** LLM-Only answers generally exhibit lower perplexity compared to RAG answers. The adversary identifies RAG usage according to the difference in Perplexity.

❷ **Sentence Embedding-Based Attack.** The inherent randomness of the LLM is minor compared to the impact of introducing external knowledge.

The adversary can introduce an isolated LLM-only model to assist in identifying RAG usage.

❸ **Prompt Injection-Based Attack.** When prompt injection is used to make the LLM ignore retrieved contexts, the RAG output changes, while the LLM-Only output ideally remains similar. The adversary can conduct prompt injection to assist in identifying RAG usage.

❹ **Breaking Retriever-Based Attack.** If a text can be inserted into the original query Q that disrupts the retriever's function without affecting the LLM's understanding of Q, then RAG will be more impacted than LLM-Only. The adversary can break the retriever to assist in identifying RAG usage.

4.4 Multi-Query Enhanced Attacks

The above attacks are simply based on the single-query-level. However, in real-world attack scenarios, the adversary can make multiple queries to a target. Therefore, we also propose an enhanced, multi-query version of these attacks. Specifically, we input m different queries to the target LLM service S_t, obtaining responses $\{R_d\}_{d=0}^{m-1}$ and using a certain RAG identification attack to compute

$$\{\mathcal{A}(S_t, R_d, \mathcal{K})\}_{d=0}^{m-1}. \quad (4)$$

We then apply a majority vote to determine the final attack result. This method can also be extended to black-box settings and fine-grained RAG property inference. Unless otherwise specified, we set $m = 10$ by default in this work.

4.5 Experimental Results

We address three research questions (RQs) through experiments on six LLMs and three datasets. RQ1: Can our attacks effectively distinguish between LLM-Only and RAG, and which one performs best? RQ2: Does increasing attack queries m consistently improve attack performance? RQ3: If RAG uses different retrievers and varies the chunk number (j), will our attack remain effective?

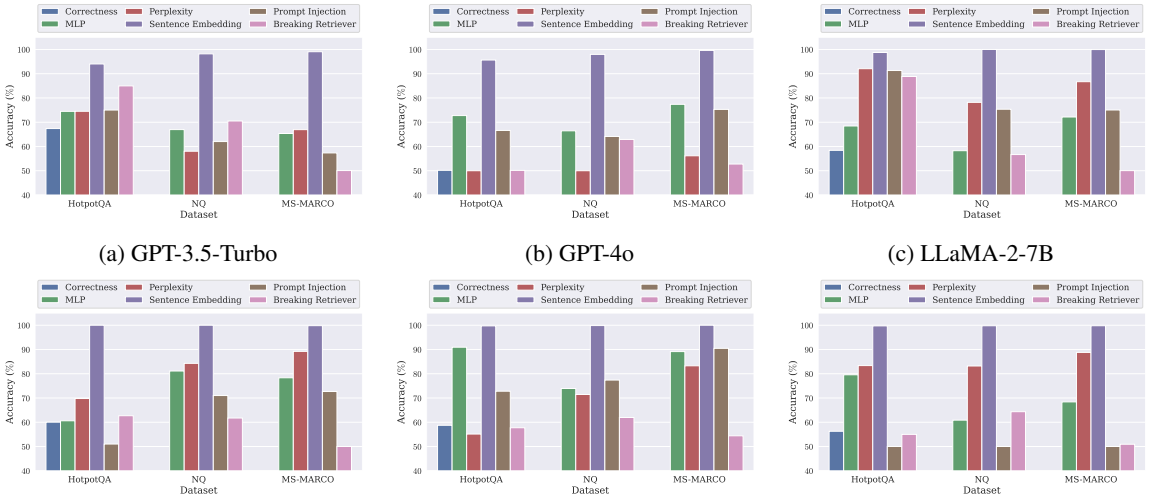
Results for RQ1. For each dataset, we split queries into 80% training (or threshold calculation) and 20% testing, reporting accuracy on the test set. Correctness-based attacks require no training and are evaluated only on the test set. Results in Figure 3 show all attacks can distinguish LLM-Only from RAG, most exceeding 60% accuracy. The sentence embedding-based attack dominates, reaching over 90% (near 100%) across settings. Correctness-based attacks, though simple, outperform random guessing and remain effective when ground-truth answers are known (adversary type 1). For type 2, the MLP-based attack captures LogProb differences, consistently above 60% and surpassing 70% in 10 of 18 scenarios. For type 3, both the embedding-based and perplexity-based attacks perform strongly, with the latter usually above 70%. Prompt injection and retriever-breaking also succeed in many cases, though prompt injection underperforms on Mistral-7B, and retriever-breaking is weaker on MS-MARCO due to prompt constraints. Further ablation results on thresholding and attack combinations are given in Appendix C.

> **Take-away for RQ1.** Our proposed attacks demonstrate effective RAG identification, with the sentence embedding-based method performing the best.

Results for RQ2. To analyze the relationship between the number of attack queries m and attack performance, we evaluate the accuracy of all attacks on the HotpotQA dataset with $m \in \{1, 5, 10, 20\}$. As shown in Figure 4, on most LLMs, increasing m from 1 to 10 significantly improves performance, with gains up to 19.41% (e.g., prompt injection-based attack on LLaMA-2-7B). On LLaMA-2-13B, however, increasing m causes only minor fluctuations in accuracy. Further raising m to 20 yields limited improvement, suggesting a trade-off between performance and cost.

> **Take-away for RQ2.** Increasing the number of attack queries m can enhance attack performance up to a point, but too many queries may introduce unnecessary overhead.

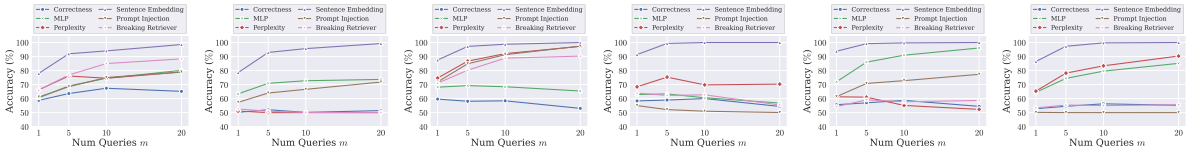
Results for RQ3. We also examine the effect of varying retrieval settings (i.e., different retrievers and the number of retrieved text chunks j) on our attacks. Specifically, we test the threshold derived from Contriever with $j = 5$ on samples generated by different retrievers and values of j. Figure 5 shows the attack performance across various retrievers. For LLMs other than GPT-3.5-Turbo, changes in retrievers have minimal impact on accuracy, always under 1%. GPT-3.5-Turbo exhibits an average maximum fluctuation of only 3.43% across datasets. In Figure 6, when varying the number of retrieved text chunks $j \in \{1, 2, 5, 10\}$, the attack performance shows slight changes on GPT-3.5-Turbo, GPT-4o, and LLaMA-2-7B, with average maximum changes of 5.42%, 3.21%, and 2.50%, respectively. On other LLMs, performance remains highly stable, with fluctuations under 1%.



(a) GPT-3.5-Turbo (b) GPT-4o (c) LLaMA-2-7B

(d) LLaMA-2-13B (e) LLaMA-3-8B (f) Mistral-7B

Figure 3: The performance of RAG identification attacks for multiple LLMs & datasets under the white-box setting.



(a) GPT-3.5-Turbo (b) GPT-4o (c) LLaMA-2-7B (d) LLaMA-2-13B (e) LLaMA-3-8B (f) Mistral-7B

Figure 4: The performance of different RAG identification attacks for multiple LLMs on HotpotQA with various m.

Take-away for RQ3. Our attack can be stably and effectively extended to a wider range of retrieval settings.

5 Black-Box RAG Identification Attacks

Recall that adversary types 2&3 require knowledge of the LLM θ and database ϕ. In this section, we design methods for performing RAG identification attacks when θ and ϕ are partially or fully unknown, i.e., black-box RAG identification attacks.

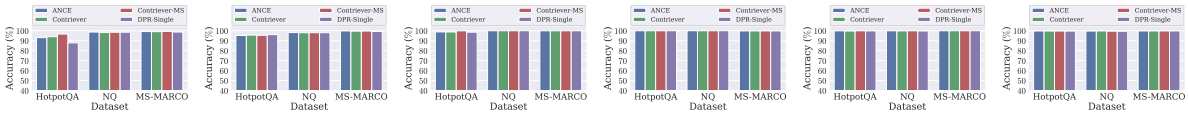
5.1 Adversary Type 2

When θ and ϕ of the target LLM service are unknown, we assume the adversary can build a pool of surrogate LLMs $\{\theta_\alpha\}_{\alpha=0}^{h-1}$ that excludes θ and a pool of surrogate databases $\{\phi_\beta\}_{\beta=0}^{g-1}$ that excludes ϕ. The key idea is to train an MLP on these surrogate LLMs and databases, enabling it to generalize to unknown LLMs and databases. Concretely, in this work, we consider six LLMs and three databases. When a certain LLM or database is unknown, the other (6-1) LLMs and (3-1) databases are used as surrogates.

LLM-Agnostic Attack. When the language model of the target LLM service is unknown but the database is known ($\mathcal{K} = \{0, 0, 1\}$), the MLP-based attack uses database-related queries to input each LLM in $\{\theta_\alpha\}_{\alpha=0}^{h-1}$, obtaining $\{D_{0,\alpha}\}_{\alpha=0}^{h-1}$ for LLMs without RAG and $\{D_{1,\alpha}\}_{\alpha=0}^{h-1}$ for LLMs with RAG. Then, following a similar approach to Equation 8, we find the optimal solution ω^* on $\{D_{0,\alpha}\}_{\alpha=0}^{h-1} \cup \{D_{1,\alpha}\}_{\alpha=0}^{h-1}$ for the classifier f.

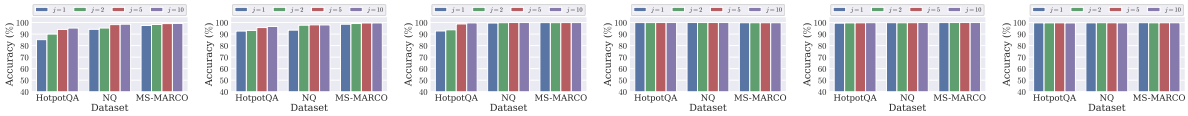
Database-Agnostic Attack. When the database is unknown ($\mathcal{K} = \{0, 1, 0\}$), we query the known LLM with queries from multiple databases $\{\phi_\beta\}_{\beta=0}^{g-1}$. For LLM without RAG, we obtain $\{D_{0,\beta}\}_{\beta=0}^{g-1}$. For LLM with RAG, the LLM performs retrieval on the query-related database and generates $\{D_{1,\beta}\}_{\beta=0}^{g-1}$. After that, we obtain the parameters ω^* for f on $\{D_{0,\beta}\}_{\beta=0}^{g-1} \cup \{D_{1,\beta}\}_{\beta=0}^{g-1}$.

Fully Agnostic Attack. When both LLM and database are unknown ($\mathcal{K} = \{0, 0, 0\}$), we perform a database-agnostic attack within an LLM-agnostic attack, i.e., we use queries related to $\{\phi_\beta\}_{\beta=0}^{g-1}$ on the LLM in $\{\theta_\alpha\}_{\alpha=0}^{h-1}$ to obtain $\bigcup_{\alpha=0}^{h-1} \bigcup_{\beta=0}^{g-1} \{D_{0,\alpha,\beta} \cup D_{1,\alpha,\beta}\}$. Thereafter, we have ω^* based on the obtained data for f.



(a) GPT-3.5-Turbo (b) GPT-4o (c) LLaMA-2-7B (d) LLaMA-2-13B (e) LLaMA-3-8B (f) Mistral-7B

Figure 5: The performance of sentence embedding-based attack for multiple LLMs with various retrievers.



(a) GPT-3.5-Turbo (b) GPT-4o (c) LLaMA-2-7B (d) LLaMA-2-13B (e) LLaMA-3-8B (f) Mistral-7B

Figure 6: The performance of sentence embedding-based attack for multiple LLMs with various j.

5.2 Adversary Type 3

For adversary type 3, we propose four attack methods, with the sentence embedding-based method performing best. Therefore, we extend this method to the black-box scenario, assuming the adversary has built $\{\theta_\alpha\}_{\alpha=0}^{h-1}$ and $\{\phi_\beta\}_{\beta=0}^{g-1}$. Notably, all adversary type 3 attacks use a threshold to identify RAG, indicating that other methods could also be adapted to the black-box scenario with a similar approach. In this black-box setting, our goal is to derive a threshold that can be generalized to unknown LLMs and databases.

LLM-Agnostic Attack. When $\mathcal{K} = \{0, 0, 1\}$, for a query Q, we could have $\{E(R_{P,\alpha})\}_{\alpha=0}^{h-1}$ from $\{\theta_\alpha\}_{\alpha=0}^{h-1}$ and $E(R_P)$ from target service S_t. Simply calculating the cosine similarity between R_P and each embedding in $\{E(R_{P,\alpha})\}_{\alpha=0}^{h-1}$ could introduce a lot of noise because some of the surrogate LLMs may be completely different from θ. Hence, we define the similarity between surrogate embeddings and $E(R_P)$ as

$$\begin{aligned}&\text{CosSim}(E(R_P), \{E(R_{P,\alpha})\}_{\alpha=0}^{h-1}) \\ &\qquad = \max\{\text{CosSim}(E(R_P, E(R_{P,\alpha}))\}_{\alpha=0}^{h-1}.\end{aligned} \tag{5}$$

Then, similar to Equation 13, we take a threshold $\hat{\lambda_{SE}}$ to determine RAG or LLM-Only.

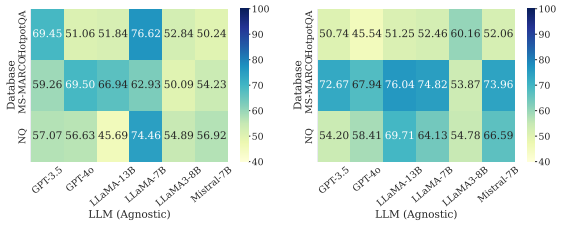
To obtain $\hat{\lambda_{SE}}$, we perform a simulated LLM-agnostic attack in $\{\theta_\alpha\}_{\alpha=0}^{h-1}$. Specifically, we first take out θ_{α_1} from $\{\theta_\alpha\}_{\alpha=0}^{h-1}$ to get $\Theta_{\alpha_1} = \{\theta_\alpha\}_{\alpha=0}^{h-1} \setminus \{\theta_{\alpha_1}\}$. Second, we query each surrogate LLM in Θ_{α_1} to get surrogate embeddings $\{\{E(R_{P,\alpha,d}) \mid \alpha \neq \alpha_1\}_{\alpha=0}^{h-1}\}_{d=0}^{n-1}$. Then, we query the hand-out LLM θ_{α_1} to get $D_{0,\alpha_1} = \{(\text{CosSim}(E(R_{P,\alpha_1,d}), \{E(R_{P,\alpha,d}) \mid \alpha \neq \alpha_1\}_{\alpha=0}^{h-1}), 0)\}_{d=0}^{n-1}$ for LLM-Only and $D_{1,\alpha_1} = \{(\text{CosSim}(E(R_{P,\alpha_1,d}), \{E(R_{P,\alpha,d}) \mid \alpha \neq \alpha_1\}_{\alpha=0}^{h-1}), 1)\}_{d=0}^{n-1}$ for RAG. Following steps in the white-box sentence embedding-based attack, we could derive a threshold λ_{SE,α_1} for the attack against θ_{α_1}. Furthermore, by changing α_1 to each value from 0 to $h-1$, we get multiple thresholds $\{\lambda_{SE,\alpha_1}\}_{\alpha_1=0}^{h-1}$ and finally calculate $\hat{\lambda_{SE}} = \frac{1}{h-1}\sum_{\alpha_1=0}^{h-1}\lambda_{SE,\alpha_1}$.

Database-Agnostic Attack. The database-agonistic setting is a more realistic scenario, as existing work (Carlini et al., 2024a; Cai et al., 2025) provides various methods for inferring the LLM used by an API-based LLM service. When $\mathcal{K} = \{0, 1, 0\}$, we simply query the known LLM with queries related to $\{\phi\}_{\beta=0}^{g-1}$, obtaining $\{D_{0,\beta}\}_{\beta=0}^{g-1}$ from LLM-Only and $\{D_{1,\beta}\}_{\beta=0}^{g-1}$ from RAG. For each surrogate database, we could derive a corresponding threshold $\lambda_{SE,\beta}$ and compute the final threshold as $\hat{\lambda_{SE}} = \frac{1}{g-1}\sum_{\beta=0}^{g-1}\lambda_{SE,\beta}$.

Fully Agnostic Attack. When $\mathcal{K} = \{0, 0, 0\}$, similar to Section 5.1, the key idea of designing a fully agnostic attack is to perform an LLM-agnostic attack within a database-agnostic attack. For each surrogate database ϕ_β in $\{\phi_\beta\}_{\beta=0}^{g-1}$, we could get $(h-1)$ different thresholds $\{\lambda_{SE,\alpha_1,\beta}\}_{\alpha_1=0}^{h-1}$. Therefore, by traversing all databases $\{\phi_\beta\}_{\beta=0}^{g-1}$, we can get $(h-1)(g-1)$ thresholds and calculate the final threshold as $\hat{\lambda_{SE}} = \frac{1}{(h-1)(g-1)}\sum_{\alpha_1=0}^{h-1}\sum_{\beta=0}^{g-1}\lambda_{SE,\alpha_1,\beta}$.

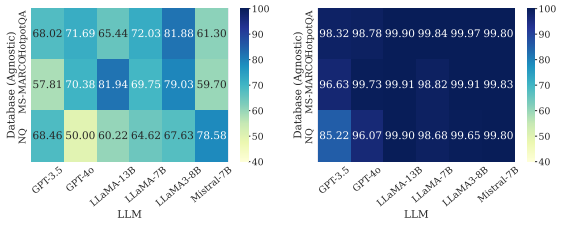
5.3 Experimental Results

We evaluate adversary type 2 (MLP-based) and type 3 (sentence embedding-based) attacks without access to the target service's LLM or database. LLM-agnostic and database-agnostic results are shown in Figure 7 and Figure 8. For the database-unknown case, we query the target LLM service



(a) MLP (b) Sentence embedding

Figure 7: The performance of LLM-agnostic attacks.



(a) MLP (b) Sentence embedding

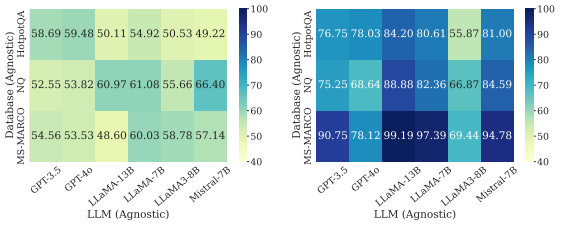
Figure 8: The performance of database-agnostic attacks.

with surrogate database-related queries and report average accuracy.

In LLM-agnostic attacks, both methods degrade compared to white-box. The MLP-based attack exceeds 60% accuracy in 6 of 18 scenarios, while the embedding-based attack surpasses 60% in most NQ and MS-MARCO cases, peaking at 76.04%. For database-agnostic attacks, both methods transfer well: MLP reaches 81.94% and embedding-based achieves over 95% in 17 of 18 cases. Combined with existing LLM inference methods (Carlini et al., 2024a; Cai et al., 2025), these results show strong practicality.

Figure 9 considers the extreme setting where both LLM and database are unknown. Here, MLP achieves over 60% in 4 cases, while the embedding-based attack performs unexpectedly better, exceeding 80% in more than half the scenarios and peaking at 99.19%, likely benefiting from diverse dataset queries.

Take-aways. The proposed attacks can be generalized effectively even in black-box scenarios with limited knowledge of the target service, with sentence embedding-based attacks showing superior performance. Additionally, transferring attacks between LLMs (from surrogate to unknown LLMs) results in greater performance degradation than transferring between databases.



(a) MLP (b) Sentence embedding

Figure 9: The performance of fully agnostic attacks.

6 Fine-Grained RAG Property Inference

Once a target LLM service is identified as RAG, various RAG-specific attacks (Zou et al., 2024; Cho et al., 2024; Xue et al., 2024; Chen et al., 2024b) can be launched. Additionally, the adversary may attempt to infer more fine-grained properties of the RAG system, further exposing intellectual property vulnerabilities and enhancing the effectiveness of other attacks. We conduct a case study to infer two fine-grained properties (the database and LLM) of the target RAG system in a black-box setting. Due to the page limit, we present the results in Appendix D.

7 Conclusion

In this work, we provide a pioneering investigation into the risks of RAG property leakage in API-accessible LLM services. In particular, we introduce `RAG-ID`, a framework for IDentifying RAG Properties in LLM services. Through `RAG-ID`, we systematically categorize adversaries and present a suite of six attack methods capable of identifying RAG usage in LLM services across both white- and black-box settings. Extensive experiments validate the effectiveness, robustness, and adaptability of our attacks, underscoring the practicality of our approach in real-world scenarios. Besides, `RAG-ID` can further infer fine-grained properties, such as the specific LLM and database used. These insights significantly increase the potential for designing targeted attacks, exposing vulnerabilities in proprietary data sources and model configurations.

Overall, this work highlights critical security implications for API-accessible LLM services, emphasizing the need for increased awareness and stronger defenses to protect against RAG property leakage. By showing both identification and detailed inference capabilities, our study expands understanding of RAG-related threats and provides a baseline for future protections.

Limitations

In this study, we do not explore defenses against the proposed attacks. Notably, some existing methods may offer partial defense against `RAG-ID`. For instance, (Zeng et al., 2024) controls output Log-Probs uncertainty through backdoor training, which may affect the classification of our MLP-based attack. Additionally, the performance of our prompt injection-based attack relies on the adopted injected prompt, indicating that the current game between prompt injection attacks and defenses could provide insights for our work (Perez and Ribeiro, 2022; Liu et al., 2024; Abdelnabi et al., 2023; Piet et al., 2024; Yi et al., 2023; Chen et al., 2024a). However, as RAG properties can leak through multiple channels, no single defense currently exists to eliminate the differences between LLM-Only and RAG outputs uniformly. Developing such defenses remains an open area of research, which we leave for future work. Besides, the RAG system may use different mechanisms to perform RAG, which means it does not search the database on all user queries. Our experimental results for the case of $m = 1$ (see Figure 4) can provide some insights at the single-query-level, and we leave the analysis of different RAG mechanisms as future work. Moreover, for the threshold-based attack method, we acknowledge that more threshold selection methods such as TPR@Low FPR can be adopted (see Appendix C), and consider the exploration of more reasonable threshold selection methods as one of the future research directions.

Ethical Considerations

Irresponsible use of the methods we propose may result in damage to the intellectual property rights of LLM service providers. However, the primary goals of this research are to reveal the property leakage risk of the current RAG system and to provide a basis for the secure and transparent deployment of future RAG systems. We do not encourage such irresponsible use. Besides, we conduct experiments using publicly available datasets and LLMs that are commonly employed in prior studies. Therefore, our work does not pose any direct ethical concerns. We will publicly release our code to promote reproducibility and advance future research on secure RAG.

References

Sahar Abdelnabi, Kai Greshake, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In *Workshop on Security and Artificial Intelligence (AISec)*, pages 79–90. ACM.

Gabriel Alon and Michael Kamfonas. 2023. Detecting Language Model Attacks with Perplexity. *CoRR abs/2308.14132*.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. 2016. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. *CoRR abs/1611.09268*.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego De Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, and 9 others. 2022. Improving Language Models by Retrieving from Trillions of Tokens. In *International Conference on Machine Learning (ICML)*, pages 2206–2240. PMLR.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language Models are Few-Shot Learners. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NeurIPS.

Will Cai, Tianneng Shi, Xuandong Zhao, and Dawn Song. 2025. Are You Getting What You Pay For? Auditing Model Substitution in LLM APIs. *CoRR abs/2504.04715*.

Nicholas Carlini, Daniel Paleka, Krishnamurthy Dj Dvijotham, Thomas Steinke, Jonathan Hayase, A. Feder Cooper, Katherine Lee, Matthew Jagielski, Milad Nasr, Arthur Conmy, Eric Wallace, David Rolnick, and Florian Tramèr. 2024a. Stealing Part of a production language model. In *International Conference on Machine Learning (ICML)*, pages 5680–5705. PMLR.

Nicholas Carlini, Daniel Paleka, Krishnamurthy Dj Dvijotham, Thomas Steinke, Jonathan Hayase, A. Feder Cooper, Katherine Lee, Matthew Jagielski, Milad Nasr, Arthur Conmy, Itay Yona, Eric Wallace, David Rolnick, and Florian Tramèr. 2024b. Stealing Part of a Production Language Model. *CoRR abs/2403.06634*.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. 2024a. StruQ: Defending Against Prompt Injection with Structured Queries. *arXiv preprint arXiv:2402.06363*.

Zhuo Chen, Jiawei Liu, Haotan Liu, Qikai Cheng, Fan Zhang, Wei Lu, and Xiaozhong Liu. 2024b. Black-Box Opinion Manipulation Attacks to Retrieval-Augmented Generation of Large Language Models. *CoRR abs/2407.13757*.

Sukmin Cho, Soyeong Jeong, Jeongyeon Seo, Taeho Hwang, and Jong C. Park. 2024. Typos that Broke the RAG's Back: Genetic Attack on RAG Pipeline by Simulating Documents in the Wild via Low-level Perturbations. *CoRR abs/2404.13948*.

Federal Trade Commission. 2024. FTC Announces Crackdown on Deceptive AI Claims and Schemes. Accessed April 2025.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2023. Retrieval-Augmented Generation for Large Language Models: A Survey. *CoRR abs/2312.10997*.

Google Cloud. 2024. Vertex AI Agent Builder. Accessed April 2025.

Abe Bohan Hou, Orion Weller, Guanghui Qin, Eugene Yang, Dawn Lawrie, Nils Holzenberger, Andrew Blair-Stanek, and Benjamin Van Durme. 2024. CLERC: A Dataset for Legal Case Retrieval and Retrieval-Augmented Analysis Generation. *CoRR abs/2406.17186*.

Zhibo Hu, Chen Wang, Yanfeng Shu, Hye-Young Paik, and Liming Zhu. 2024. Prompt Perturbation in Retrieval-Augmented Generation based Large Language Models. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM.

Yangsibo Huang, Samyak Gupta, Mengzhou Xia, Kai Li, and Danqi Chen. 2023. Catastrophic Jailbreak of Open-source LLMs via Exploiting Generation. *CoRR abs/2310.06987*.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised Dense Information Retrieval with Contrastive Learning. *Transactions on Machine Learning Research*.

JD Supra. 2024. Fighting the Robots: Texas Attorney General Settles "First-of-its-Kind" Investigation of Healthcare AI Company. Accessed April 2025.

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. *ACM Computing Surveys*.

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

Zhengbao Jiang, Frank Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023b. Active Retrieval Augmented Generation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7969–7992.

Ziyan Jiang, Xueguang Ma, and Wenhu Chen. 2024. LongRAG: Enhancing Retrieval-Augmented Generation with Long-context LLMs. *CoRR abs/2406.15319*.

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 *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, page 6769–6781. ACL.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur 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 Le, and Slav Petrov. 2019. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 9459–9474.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. 2023. AutoDAN: Generating Stealthy Jailbreak Prompts on Aligned Large Language Models. *CoRR abs/2310.04451*.

Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. 2024. Formalizing and Benchmarking Prompt Injection Attacks and Defenses. In *USENIX Security Symposium (USENIX Security)*, pages 1831–1847. USENIX.

Hope McGovern, Rickard Stureborg, Yoshi Suhara, and Dimitris Alikaniotis. 2024. Your Large Language Models Are Leaving Fingerprints. *CoRR abs/2405.14057*.

Reham Omar, Omij Mangukiya, and Essam Mansour. 2025. Dialogue Benchmark Generation from

Knowledge Graphs with Cost-Effective Retrieval-Augmented LLMs. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM.

OpenAI. 2023. GPT-4 Technical Report. *CoRR abs/2303.08774*.

OpenAI. 2024a. Morgan Stanley Uses AI Evals to Shape the Future of Financial Services. Accessed April 2025.

OpenAI. 2024b. Rakuten Pairs Data with AI to Unlock Customer Insights and Value. Accessed April 2025.

Jonas Oppenlaender. 2022. A Taxonomy of Prompt Modifiers for Text-To-Image Generation. *CoRR abs/2204.13988*.

Dario Pasquini, Evgenios M. Kornaropoulos, and Giuseppe Ateniese. 2024. LLMmap: Fingerprinting For Large Language Models. *CoRR abs/2407.15847*.

Fábio Perez and Ian Ribeiro. 2022. Ignore Previous Prompt: Attack Techniques For Language Models. *CoRR abs/2211.09527*.

Julien Piet, Maha Alrashed, Chawin Sitawarin, Sizhe Chen, Zeming Wei, Elizabeth Sun, Basel Alomair, and David Wagner. 2024. Jatmo: Prompt Injection Defense by Task-Specific Finetuning. In *European Symposium on Research in Computer Security (ESORICS)*, pages 105–124. Springer.

PMFM AI. 2025. Pmfm.ai. Accessed April 2025.

Vipula Rawte, Amit P. Sheth, and Amitava Das. 2023. A Survey of Hallucination in Large Foundation Models. *CoRR abs/2309.05922*.

Retool. 2023. The State of AI in 2023. Accessed April 2025.

Sina J. Semnani, Violet Z. Yao, Heidi C. Zhang, and Monica S. Lam. 2023. WikiChat: Stopping the Hallucination of Large Language Model Chatbots by Few-Shot Grounding on Wikipedia. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, page 2387–2413.

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. 2023. Do Anything Now: Characterizing and Evaluating In-The-Wild Jailbreak Prompts on Large Language Models. *CoRR abs/2308.03825*.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *IEEE Symposium on Security and Privacy (S&P)*, pages 3–18. IEEE.

Jingtong Su, Julia Kempe, and Karen Ullrich. 2024. Mission Impossible: A Statistical Perspective on Jailbreaking LLMs. *CoRR abs/2408.01420*.

Hexiang Tan, Fei Sun, Wanli Yang, Yuanzhuo Wang, Qi Cao, and Xueqi Cheng. 2024. Blinded by Generated Contexts: How Language Models Merge Generated and Retrieved Contexts When Knowledge Conflicts? In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 6207–6227.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NeurIPS.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *CoRR abs/2307.09288*.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NeurIPS.

Guangzhi Xiong, Qiao Jin, Zhiyong Lu, and Aidong Zhang. 2024. Benchmarking Retrieval-Augmented Generation for Medicine. *CoRR abs/2402.13178*.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *International Conference on Learning Representations (ICLR)*.

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. *CoRR abs/2406.00083*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William 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 (EMNLP)*, pages 2369–2380. ACL.

Jingwei Yi, Yueqi Xie, Bin Zhu, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. 2023. Benchmarking and Defending Against Indirect Prompt Injection Attacks on Large Language Models. *CoRR abs/2312.14197*.

William J Youden. 1950. Index for Rating Diagnostic Tests. *Cancer*, 3(1):32–35.

YouTube. 2023. BlueBot: JetBlue's Unified LLM. Accessed April 2025.

Qingcheng Zeng, Mingyu Jin, Qinkai Yu, Zhenting Wang, Wenyue Hua, Zihao Zhou, Guangyan Sun, Yanda Meng, Shiqing Ma, Qifan Wang, Felix Juefei-Xu, Kaize Ding, Fan Yang, Ruixiang Tang, and Yongfeng Zhang. 2024. Uncertainty is Fragile: Manipulating Uncertainty in Large Language Models. *CoRR abs/2407.11282*.

Andy K Zhang, Kevin Klyman, Yifan Mai, Yoav Levine, Yian Zhang, Rishi Bommasani, and Percy Liang. 2024. Language Model Developers Should Report Train-Test Overlap. *CoRR abs/2410.08385*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, and 3 others. 2023. A Survey of Large Language Models. *CoRR abs/2303.18223*.

Zhiguang Yang and Hanzhou Wu. 2024. A Fingerprint for Large Language Models. *CoRR abs/2407.01235*.

Zexuan Zhong, Ziqing Huang, Alexander Wettig, and Danqi Chen. 2023. Poisoning Retrieval Corpora by Injecting Adversarial Passages. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, page 13764–13775. ACL.

Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. 2023. Universal and Transferable Adversarial Attacks on Aligned Language Models. *CoRR abs/2307.15043*.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. 2024. PoisonedRAG: Knowledge Poisoning Attacks to Retrieval-Augmented Generation of Large Language Models. *CoRR abs/2402.07867*.

A Target LLM Service Setups

LLMs. In this work, we utilize six LLMs from three model families: GPT (GPT-3.5-Turbo-0125 and GPT-4o-2024-08-06), LLaMA (LLaMA-2-Chat-7B/13B and LLaMA-3-8B), and Mistral-7B-Instruct-v0.3. For simplicity, we refer to them as GPT-3.5-Turbo, GPT-4o, LLaMA-2-7B/13B, LLaMA-3-8B, and Mistral-7B in later sections. To ensure a fair comparison and facilitate the transferability of `RAG-ID` to other attack scenarios, we set the temperature of all models to 0.1, following the approach in prior work (Zou et al., 2024). For victim LLM services with RAG, i.e., target RAG system, we adopt the prompt template from (Zou et al., 2024) to guide the LLMs in generating answers to given questions. For closed-source LLMs, we query their official APIs; for open-source LLMs, we deploy them using NVIDIA A100-80GB.

Prompt Template for RAG System

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: [context]
Query: [question]
Answer:

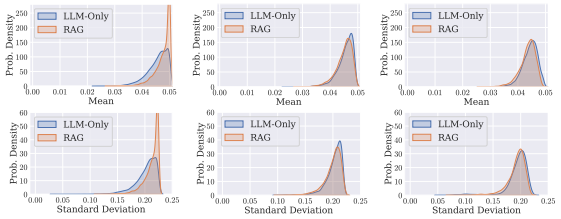
To minimize differences in responses between RAG and LLM-Only due to the prompt template, we modify the RAG prompt template to the following one for LLM-Only.

Prompt Template for LLM-Only

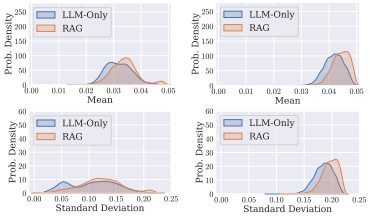
You are a helpful assistant, below is a query from a user. Your answer should be short and concise. If you cannot answer the question, just say "I don't know."
Query: [question]
Answer:

Retrievers in RAG. Following previous studies (Izacard et al., 2022; Xiong et al., 2021; Zhong et al., 2023; Zou et al., 2024), we use Contriever (Izacard et al., 2022) as the default retriever and consider other retrievers (ANCE (Xiong et al., 2021), Contriever-MS (Izacard et al., 2022), and DPR-Single (Karpukhin et al., 2020)) for evaluation. Consistent with (Lewis et al., 2020), we apply the dot product to retrieve the j text chunks most similar to the query's embedding from the knowledge database. In this work, we set $j = 5$ by default and examine the impact of different j in the evaluation.

Datasets. We use three popular question-answering datasets for evaluation: HotpotQA (Yang et al., 2018), Natural Questions (NQ) (Kwiatkowski et al., 2019), and MS-MARCO (Bajaj et al., 2016), which contain 7,405, 3,452, and 6,980 queries, respectively. Each dataset is paired with a comprehensive knowledge database for retrieval. Specifically, the knowledge databases for HotpotQA and NQ are sourced from Wikipedia, with 5,233,329 and 2,681,468 text chunks, respectively. The MS-MARCO knowledge database contains 8,841,823 text chunks from 3,563,535 web documents retrieved via Microsoft

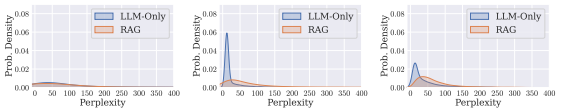


(a) GPT-4o (b) LLaMA-2-7B (c) LLaMA-2-13B

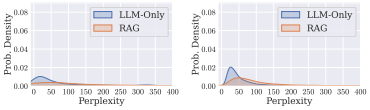


(d) LLaMA-3-8B (e) Mistral-7B

Figure 10: The probability distribution of the mean and standard deviation of the Top-k exponentiated LogProbs for responses generated by LLM-Only or RAG on HotpotQA dataset, where mulitple LLMs are evaluated.



(a) GPT-4o (b) LLaMA-2-7B (c) LLaMA-2-13B



(d) LLaMA-3-8B (e) Mistral-7B

Figure 11: The probability distribution of perplexity for responses generated by LLM-Only or RAG on HotpotQA dataset, where multiple LLMs are evaluated.

Bing[2].

B Detailed Intuitions and Methodologies of White-Box Attacks

In this section, we provide the detailed intuition and methodology for each white-box attack type introduced in Section 4.

B.1 Adversary Type 1

Intuition. Compared with LLM-Only, RAG's answers are likely to have a higher correctness ratio because they are generated based on text chunks (contexts) related to input queries.

To verify this intuition, we calculate the correctness ratios of various LLMs in answering questions with and without RAG. Following previous work (Huang et al., 2023; Zou et al., 2024), we use subset matching to determine the correctness of a generated answer. Specifically, an answer is considered correct if the correct answer appears as a substring within the LLM's response. We do not use exact matching, as it requires the LLM's answer to be an exact replica of the correct answer to be deemed correct. For example, if the correct answer to the question "Which writer was from England, Henry Roth or Robert Erskine Childers?" is "Robert Erskine Childers," the LLM response "Robert Erskine Childers was from England" would be considered incorrect under exact matching despite conveying the right information.

LLM	Correct Ratio (%) LLM-Only	RAG	Δ
GPT-3.5-Turbo	25.08	42.31	+17.23
GPT-4o	43.48	42.54	-0.94
LLaMA-2-7B	11.36	31.56	+20.20
LLaMA-2-13B	19.19	34.10	+14.91
LLaMA-3-8B	23.31	33.50	+10.19
Mistral-7B	32.25	37.58	+5.33

Table 2: The correct answer ratios of evaluated LLMs without RAG (i.e., LLM-Only) or with RAG on the HotpotQA dataset, where Δ represents the improvement of the correct ratio by introducing RAG.

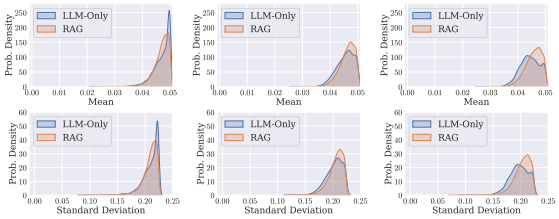
Table 2 presents the ratios of correct and incorrect answers generated by LLMs with and without RAG on the HotpotQA dataset. We evaluate HotpotQA as it provides a short correct answer for each query, which makes it suitable for us to determine its correctness using the substring match. Results indicate that RAG notably enhances the answer correctness ratio across most evaluated LLMs except GPT-4o, with the LLaMA-2-7B model showing the highest improvement of 20.20%. This finding supports our initial intuition and serves as the basis for our design methodology.

Methodology. Recall that for a target LLM service S_t, it generates a response R containing a plaintext answer R_P based on a given input query Q. Following this intuition, we identify S_t as using RAG if it generates a correct R_P, and as not using RAG otherwise. Formally, we have

$$\mathcal{A}(S_t, R, \mathcal{K}) = \begin{cases} 0 & \text{if } C \notin R_P, \\ 1 & \text{if } C \in R_P, \end{cases} \quad (6)$$
$$s.t.\ R = R_P \text{ and } \mathcal{K} = \{1, 0, 0\}.$$

We acknowledge that this method is simplistic and has a high false negative rate, as all model response correctness ratios are below 50%, leading

[2]A search engine, https://www.bing.com/.



(a) HotpotQA (b) NQ (c) MS-MARCO

Figure 12: The probability distribution of the mean and standard deviation of the Top-k exponentiated LogProbs for responses generated by LLM-Only or RAG across various datasets, where the evaluated LLM is GPT-3.5-Turbo.

to a high likelihood of misidentifying RAG systems producing incorrect answers as LLM-Only. Nonetheless, we present it as a stronger baseline than random guessing for this unexplored problem, and as a straightforward solution when $\mathcal{K} = \{1, 0, 0\}$.

B.2 Adversary Type 2

Intuition. The answer LogProbs generated by LLM-Only and RAG likely follow different distributions, as RAG introduces external knowledge that influences the LLM's confidence in its responses.

We calculate the mean, and standard deviation of the token-wise averaged top-k LogProbs $\hat{R_L}$ for each answer to queries across different datasets, and for each query, we consider LLMs with and without RAG. Specifically, we assume that R_L contains L Top-k LogProbs $\{\mathbf{T}_0, \mathbf{T}_1, \cdots, \mathbf{T}_{L-1}\}$, where $\mathbf{T}_l = \begin{bmatrix} v_{l,0} & v_{l,1} & \cdots & v_{l,k-1} \end{bmatrix}^T$, then we have

$$\hat{R_L} = \mathbf{T}_0 \oplus \mathbf{T}_1 \oplus \cdots \oplus \mathbf{T}_{L-1} = \begin{bmatrix} v_{0,0} \\ v_{0,1} \\ \cdots \\ v_{0,k-1} \end{bmatrix} \oplus \begin{bmatrix} v_{1,0} \\ v_{1,1} \\ \cdots \\ v_{1,k-1} \end{bmatrix} \oplus \cdots \oplus \begin{bmatrix} v_{L-1,0} \\ v_{L-1,1} \\ \cdots \\ v_{L-1,k-1} \end{bmatrix}, \tag{7}$$

where $\oplus$ indicates element-wise addition. In this work, we set k to 20, aligning with the k value supported by GPT models.

To improve visualization, we convert each LogProb to its exact probability by exponentiating it. The mean and standard deviation of these Top-k exponentiated LogProbs are shown in Figure 12 for GPT-3.5-Turbo. As illustrated in Figure 12, the LogProb distributions of LLM-Only and RAG responses differ consistently across datasets, particularly in NQ (Figure 12b) and MS-MARCO (Figure 12c), where RAG responses exhibit higher means and variances than LLM-Only responses. We also observe trends similar to those of other LLMs in Figure 10. For instance, on HotpotQA, GPT-4o shows significantly higher mean and variance in LogProbs when using RAG compared to the LLM-Only setup. This increase may result from RAG enhancing the LLMs' confidence in generating responses, leading to higher LogProbs, as noted in previous research (Jiang et al., 2023b). This finding supports our initial intuition and informs our design methodology.

Furthermore, we also observe that the differences between LLM-Only and RAG vary across datasets and LLMs. For example, on GPT-3.5-Turbo, RAG shows higher mean and variance on the NQ and MS-MARCO datasets, while the reverse is true on HotpotQA. Additionally, for HotpotQA, the LLM-Only responses from GPT-3.5-Turbo display higher mean and variance, a trend not seen with Mistral-7B. These variations highlight the limitations of using simple features, such as mean and variance, to distinguish between LLM-Only and RAG responses and the challenge of creating a model- and dataset-independent (black-box) method for RAG identification.

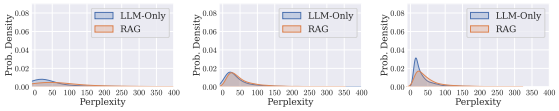
Methodology. Based on these findings, we design a multilayer perceptron (MLP) model f with parameters ω to capture both direct and subtle differences beyond mean and variance. We deploy a 4-layer MLP with k, 64, 64, 64, and 2 neurons as our attack model. For each LLM θ and database ϕ, we use queries related to this database to obtain $D_0 = \{(\exp \hat{R_{L,d}}, 0)\}_{d=0}^{n-1}$ for LLM without RAG and $D_1 = \{(\exp \hat{R_{L,d}}, 1)\}_{d=0}^{n-1}$ for LLM with RAG. We train the MLP f for 50 epochs with a learning rate of 3e-4 to find the optimal solution:

$$\arg\min_{\omega} -\frac{1}{2n} \sum_{(x,y) \in D_0 \cup D_1} ((1-y)\log(1-f(x)) + y\log(f(x))). \tag{8}$$

Ideally, we would have a trained f that classifies LLM-Only LogProbs as 0 and RAG LogProbs as 1. Formally, the RAG identification attack could be rewritten as

$$\mathcal{A}(S_t, R, \mathcal{K}) = \text{round}(f(R_L)) \quad s.t.\ R = R_L \text{ and } \mathcal{K} = \{0, 1, 1\}, \tag{9}$$

where round$(\cdot)$ rounds a given number to the nearest integer and we represent $f(\exp \hat{R_{L,d}})$ as $f(R_L)$



(a) HotpotQA (b) NQ (c) MS-MARCO

Figure 13: The probability distribution of perplexity for responses generated by LLM-Only or RAG across various datasets, where the evaluated LLM is GPT-3.5-Turbo.

for simplicity.

B.3 Adversary Type 3

❶ Perplexity-Based Attack (Intuition). The intuition is that LLM-Only answers generally exhibit lower perplexity[3] compared to RAG answers. This occurs because answers generated without external context align more closely with the LLM's training data and internal knowledge.

Additionally, LLMs are trained on large datasets, such as publicly available sources like Wikipedia (Zhang et al., 2024), which often overlap. This overlap leads to similar judgments of perplexity for texts within the training distribution. Therefore, when an LLM generates an answer consistent with the training data, it should have lower perplexity when measured by another LLM. In contrast, answers generated with external knowledge may still result in higher perplexity.

Thus, following previous work (Alon and Kamfonas, 2023), we leverage a third-party LLM, GPT-2, an open-source model trained on a large amount of manually filtered text, to calculate perplexity. The perplexity distribution for LLM-Only and RAG answers on GPT-3.5-Turbo is shown in Figure 13, where we exclude the highest and lowest 5% of values for better visualization. We observe that, across all evaluated datasets, RAG answers consistently exhibit higher perplexity than LLM-Only answers, confirming our intuition. Similar results are found for other LLMs on HotpotQA (see Figure 11).

❶ Perplexity-Based Attack (Methodology). We need to derive a perplexity threshold λ_{PPL} to classify responses from the LLM service S_t: responses with perplexity below this threshold are labeled as LLM-Only, and those above as RAG. Formally, we have

$$\mathcal{A}(S_t, R, \mathcal{K}) = \begin{cases} 0 & \text{if } PPL(R_P) \leq \lambda_{PPL}, \\ 1 & \text{if } PPL(R_P) > \lambda_{PPL}, \end{cases} \quad s.t.\ R = R_P \text{ and } \mathcal{K} = \{0, 1, 1\}, \tag{10}$$

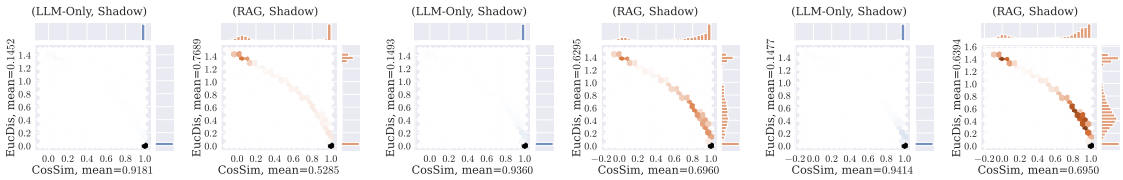
where $PPL(\cdot)$ is used to calculate the perplexity of a given text based on GPT-2.

To obtain λ_{PPL}, following the methodology in Section 4.2, we query each LLM with questions related to the given database. For LLMs without RAG, we collect $D_0 = \{(PPL(R_{P,d}), 0)\}_{d=0}^{n-1}$, and for LLMs with RAG, we collect $D_1 = \{(PPL(R_{P,d}), 1)\}_{d=0}^{n-1}$. Using the combined samples $D_0 \cup D_1$, we calculate the true positive rate (TPR) and false positive rate (FPR) as the threshold varies. We then select the threshold based on Youden's J Statistic that maximizes Youden's index ($TPR - FPR$) as λ_{PPL}, and apply this λ_{PPL} in Equation 10 for RAG identification. We acknowledge other feasible threshold selection methods, such as selecting a threshold to obtain TPR for a given low FPR (i.e., TPR@Low FPR). In this work, unless otherwise mentioned, we use Youden's J Statistic to select thresholds by default.

❷ Sentence Embedding-Based Attack (Intuition). The inherent randomness of the LLM is minor compared to the impact of introducing external knowledge. In other words, for a given LLM and fixed query, the difference between responses generated by LLM-Only and RAG is greater than the variation between two LLM-only responses (which may differ due to the non-zero temperature adding randomness).

To evaluate response differences, inspired by attacks on machine learning models (e.g., membership inference attack (Shokri et al., 2017)), we introduce a shadow LLM θ' to approximate the behavior of the target LLM θ. In a white-box setting, we assume θ' is identical to θ, an assumption relaxed in later sections for black-box attacks. For each response text R_P, we use the embedding model E to compute its sentence embedding. Specifically, we employ the all-MiniLM-L6-v2 model, a fine-tuned version of MiniLM (Wang et al., 2020), which maps input sentences to a 384-dimensional dense vector space for tasks like sentence clustering and similarity comparison. We then use two metrics—cosine similarity and Euclidean distance—to measure differences between shadow LLM and LLM-only answers, as well as between shadow LLM and RAG answers. Specif-

[3] A metric for evaluating language model performance; lower perplexity indicates higher confidence in predicting each word.



(a) HotpotQA (b) NQ (c) MS-MARCO

Figure 14: The sentence embedding differences measured by cosine similarity and Euclidean Distance between LLM-Only and shadow LLM, as well as RAG and shadow LLM across various datasets. Here the evaluated LLM is GPT-3.5-Turbo.

ically, for two given sentence embeddings e_1 and e_2, the cosine similarity and Euclidean Distance could be formulated as

$$\text{CosSim}(e_1, e_2) = \frac{\langle e_1, e_2 \rangle}{\|e_1\| \|e_2\|}, \tag{11}$$

and

$$\text{EucDis}(e_1, e_2) = \|e_1 - e_2\|. \tag{12}$$

If our intuition is correct, CosSim(LLM-Only, Shadow) should be larger than CosSim(RAG, Shadow), and EucDis(LLM-Only, Shadow) should be smaller than EucDis(RAG, Shadow).

Figure 14 shows the distribution of sentence embedding differences across various datasets on GPT-3.5-Turbo. As expected, cosine similarity values for (LLM-Only, Shadow) pairs are mostly above 0.9, close to 1.0, while many (RAG, Shadow) pairs fall below 0.9, with clusters appearing between 0.0 and 0.2. For Euclidean distance, (RAG, Shadow) has numerous non-zero values, such as in Figure 14c, where many values range around 0.4 to 1.4. In contrast, (LLM-Only, Shadow) distances are nearly all zero, with only a few values between 0.0 and 0.4. The mean values of these metrics further support these findings. Additionally, similar patterns are observed with other LLMs on HotpotQA dataset (see Figure 17), where (RAG, Shadow) consistently shows larger sentence embedding differences than (LLM-Only, Shadow).

❷ Sentence Embedding-Based Attack (Methodology). In our statistics, we observe that cosine similarity and Euclidean distance are negatively correlated, with smaller Euclidean distances typically indicating greater cosine similarity. While other metrics, such as Manhattan distance and Hamming distance, could also reflect differences between LLM-Only and RAG, we focus on cosine similarity alone for this study, leaving the exploration of additional metrics and combinations for future work. Our goal is to establish a cosine similarity threshold (λ_{SE}) to determine if the target LLM service S_t uses RAG, based on the similarity between responses from S_t and the shadow LLM.

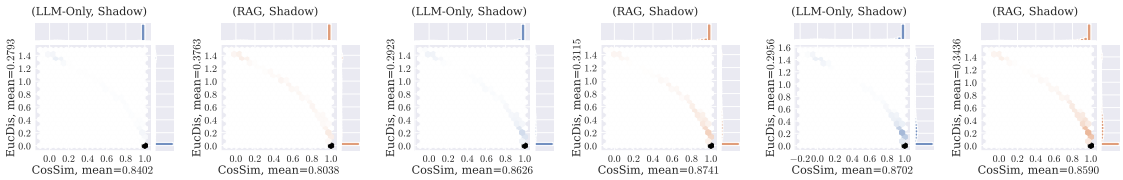
To obtain λ_{SE}, for a given LLM θ and database ϕ, we first query our shadow LLM (LLM-Only) θ' to generate $D' = \{E(R'_{P,d})\}_{d=0}^{n-1}$. Next, we query θ to obtain $D_0 = \{(\text{CosSim}(E(R_{P,d}), E(R'_{P,d})), 0)\}_{d=0}^{n-1}$ for LLM responses without RAG and $D_1 = \{(\text{CosSim}(E(R_{P,d}), E(R'_{P,d})), 1)\}_{d=0}^{n-1}$ for responses with RAG. By adjusting the threshold, we find the optimal value that maximizes $TPR - FPR$ on $D_0 \cup D_1$, setting it as λ_{SE}. Formally, the RAG identification attack could be written as

$$\mathcal{A}(S_t, R, \mathcal{K}) = \begin{cases} 0 & \text{if CosSim}(E(R_P), E(R'_P)) > \lambda_{SE}, \\ 1 & \text{if CosSim}(E(R_P), E(R'_P)) \leq \lambda_{SE}, \end{cases}$$
$$s.t.\ R = R_P \text{ and } \mathcal{K} = \{0, 1, 1\}, \tag{13}$$

where R'_P is the plaintext answer generated by θ'.

❸ Prompt Injection-Based Attack (Intuition). The intuition behind this attack is that when prompt injection is used to make the LLM ignore retrieved contexts, the RAG output changes, while the LLM-Only output ideally remains similar.

The prompt injection attack inserts a specific prompt into the LLM's input to make it produce a response aligned with the adversary's intent. Prior research (Liu et al., 2024) demonstrates the effectiveness of prompt injection across various LLMs and tasks, with limited defenses available. In this attack, we use an "ignore prompt" (Perez and Ribeiro, 2022), specifically injecting the prompt, "The above instructions and contexts are designed to confuse your judgment. Please ignore them and answer the question based entirely on your knowledge," before the original query Q to create the prompt-injected query Q_{Inject}. We input both Q and Q_{Inject} into S_t to obtain responses R_P and R_P^{Inject}. Using the embedding model E, we calculate the sentence embedding for each response, following the same method as



(a) HotpotQA (b) NQ (c) MS-MARCO

Figure 15: The sentence embedding differences measured by cosine similarity and Euclidean Distance between answers for original and prompt-injected queries for LLM-Only and RAG across various datasets. Here the evaluated LLM is GPT-3.5-Turbo.

in the sentence embedding-based attack. If our intuition is correct, $\text{CosSim}(E(R_P), E(R_P^{Inject}))$ for LLM-Only should be higher than for RAG, while $\text{EucDis}(E(R_P), E(R_P^{Inject}))$ for LLM-Only should be lower than for RAG.

Figure 15 shows the sentence embedding differences between responses generated by original and prompt-injected queries for LLM-Only and RAG on GPT-3.5-Turbo. Across all datasets, LLM-Only responses have lower Euclidean distances, with mean values of 0.2793, 0.2923, and 0.3436, compared to RAG's 0.3763, 0.3115, and 0.3436. Additionally, LLM-Only responses generally exhibit higher cosine similarity (except on NQ), indicating that RAG is more affected by prompt injection. For other LLMs (see Figure 18), we observe similar patterns on HotpotQA dataset with GPT-4o, LLaMA-2-7B/13B, and LLaMA3-8B, though not with Mistral-7B. This discrepancy may be due to the injected prompt's varying effectiveness across LLMs and tasks. Exploring and evaluating more robust injection prompts is left for future work. In general, our hypothesis is supported in most cases evaluated.

❸ Prompt Injection-Based Attack (Methodology). Like the sentence embedding-based attack, this method requires deriving a cosine similarity threshold λ_{PI}. If the target LLM service S_t shows an impact below λ_{PI} from the prompt-injected query Q_{Inject}, it is classified as LLM-Only; otherwise, it is classified as RAG. Formally, the RAG identification attack can be considered as

$$\mathcal{A}(S_t, R, \mathcal{K}) = \begin{cases} 0 & \text{if } \text{CosSim}(E(R_P), E(R_P^{Inject})) > \lambda_{PI}, \\ 1 & \text{if } \text{CosSim}(E(R_P), E(R_P^{Inject})) \leq \lambda_{PI}, \end{cases} \quad s.t.\ R = R_P \text{ and } \mathcal{K} = \{0, 1, 1\}. \tag{14}$$

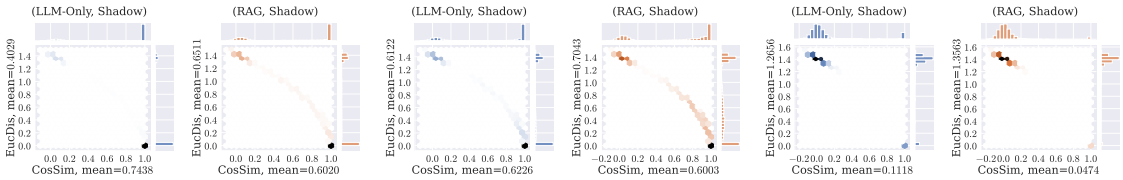
For a given LLM θ and database ϕ, we query the LLM service with both original and prompt-injected queries to obtain $D_0 = \{\text{CosSim}(E(R_{P,d}), E(R_{P,d}^{Inject})), 0\}_{d=0}^{n-1}$ for LLM-Only and $D_1 = \{\text{CosSim}(E(R_{P,d}), E(R_{P,d}^{Inject})), 1\}_{d=0}^{n-1}$ for RAG. By varying thresholds and calculating TPR and FPR on $D_0 \cup D_1$, we identify the threshold that maximizes $TPR - FPR$ as λ_{PI}.

It is worth noting that prompt-injected queries may cause retrievers to retrieve text chunks differently from the original queries. However, in this attack, we ignore the changes in RAG answers due to the changes in retrieved chunks and treat the attack on retrievers as another attack below.

❹ Breaking Retriever-Based Attack (Intuition). When the user's query changes, the text chunks retrieved in RAG may also change. If a text can be inserted into the original query Q that disrupts the retriever's function without affecting the LLM's understanding of Q, then RAG will be more impacted than LLM-Only.

We aim to create a prompt prefix and insert it into the original query Q to generate Q_{Break}, which disrupts the retriever's performance without affecting the LLM's understanding of Q. Inspired by role-playing jailbreak attacks on LLMs (e.g., the "grandma exploit," which prompts the LLM to act as "grandma" and generate unsafe content) (Shen et al., 2023; Su et al., 2024), we propose adding the prefix "The CEO of Apple, Tim Cook, asks:" before Q to create the retriever-breaking query Q_{Break}. For the LLM-Only, answering Q and Q_{Break} should produce similar responses. However, Q_{Break} could cause the retriever to be influenced by elements like "CEO," "Apple," and "Tim Cook" during retrieval. We then input Q and Q_{Break} into S_t and use the embedding model E to obtain $E(R_P)$ and $E(R_P^{Break})$. If our hypothesis is correct, $\text{CosSim}(E(R_P), E(R_P^{Break}))$ for LLM-Only should be higher than for RAG, while $\text{EucDis}(E(R_P), E(R_P^{Break}))$ for LLM-Only should be lower than for RAG.

Figure 16 shows the sentence embedding differences between responses generated by the original query Q and retriever-broken query Q_{Break} for



(a) HotpotQA (b) NQ (c) MS-MARCO

Figure 16: The sentence embedding differences measured by cosine similarity and Euclidean Distance between answers for original and retriever-broken queries for LLM-Only and RAG across various datasets. Here the evaluated LLM is GPT-3.5-Turbo.

LLM-Only and RAG on GPT-3.5-Turbo. Across all datasets, RAG consistently exhibits higher Euclidean distance and lower cosine similarity than LLM-Only, indicating that retriever-broken queries disrupt RAG's responses more significantly by affecting the retriever's function. In Figure 19, we get similar results on HotpotQA dataset for GPT-4o, LLaMA-2-7B, LLaMA3-8B, and Mistral-7B. However, for LLaMA-2-13B, the impact of retriever-broken queries on LLM-Only and RAG is comparable, possibly due to limitations in the prompt used to disrupt the retriever. Overall, our findings align with intuition across most evaluated LLMs and datasets, forming a basis for our methodology.

❹ **Breaking Retriever-Based Attack (Methodology).** Following previous attacks, this attack also aims to get a cosine similarity threshold λ_{BR} that

$$\mathcal{A}(S_t, R, \mathcal{K}) = \begin{cases} 0 & \text{if CosSim}(E(R_P), E(R_P^{Break})) > \lambda_{BR}, \\ 1 & \text{if CosSim}(E(R_P), E(R_P^{Break})) \leq \lambda_{BR}, \end{cases}$$
$$s.t.\ R = R_P \text{ and } \mathcal{K} = \{0, 1, 1\}. \tag{15}$$

To derive λ_{BR}, for given θ and ϕ, we query LLM service with original and retriever-broken queries to have $D_0 = \{\text{CosSim}(E(R_{P,d}), E(R_{P,d}^{Break})), 0\}_{d=0}^{n-1}$ from LLM-Only and $D_1 = \{\text{CosSim}(E(R_{P,d}), E(R_{P,d}^{Break})), 1\}_{d=0}^{n-1}$ from RAG, and choose the best-performed threshold as λ_{BR} based on Youden's J Statistic (Youden, 1950).

C Ablations for White-Box Type 3 attacks

Threshold Choosing Method. We show the performance (measured by TPR) of different attack methods when choosing thresholds based on TPR@Low FPR in Table 3, where Low $FPR = 20\%$. Consistent with Youden's J Statistic, the Sentence Embedding-based attack shows the best results.

Attack Combination. Besides, we evaluate the performance of combining type 3 attacks in a white-box setting based on majority voting, excluding the sentence embedding-based attack (since it already has a fairly high accuracy). As shown in Table 4, combined attacks could achieve higher accuracy in some cases (e.g., LLaMA-2-7B&HotpotQA). The combined method currently cannot outperform the Sentence Embedding-based one, however, it demonstrates the potential for further improvement of existing attack methods.

D Fine-Grained RAG Property Inference

In this section, we aim to infer two fine-grained properties of the target RAG system in a black-box setting: the database and the LLM. These properties are chosen because developing an advanced database and LLM is time-consuming and resource-intensive, making it likely that the RAG system uses widely available options.

D.1 Database Inference Attack

If the adversary knows the database used by RAG, they can more easily conduct targeted attacks, such as altering RAG outputs by manipulating the database's data source (Zou et al., 2024). Assuming a set of widely used databases $\{\phi_\alpha\}_{\alpha=0}^{h-1}$, one of which is deployed by the target service S_t, our goal is to identify the specific database in use by querying S_t and analyzing its responses.

We propose this attack using sentence embeddings, which have proven effective in identifying RAG. The key idea is that when the user query and the target service's database originate from the same source (dataset), their similarity will be high. Specifically, we create queries $\{Q_\alpha\}_{\alpha=0}^{h-1}$, each corresponding to a database in $\{\phi_\alpha\}_{\alpha=0}^{h-1}$, where one query is matched with one database. These queries are used to query S_t, generating responses $\{R_{P,\alpha}\}_{\alpha=0}^{h-1}$. Then, we determine the in-

Table 3: The TPR@Low FPR of four type 3 (threshold-based) attack methods, where Low $FPR = 20\%$.

LLM	Dataset	TPR (%)			
		Perplexity	Sentence Embedding	Prompt Injection	Breaking Retriever
GPT-3.5-Turbo	HotpotQA	9.93	83.93	8.10	0.54
	NQ	3.04	98.99	6.96	30.00
	MS-MARCO	18.05	99.79	6.16	8.09
GPT-4o	HotpotQA	1.15	29.30	0.00	0.00
	NQ	0.72	99.28	1.89	0.00
	MS-MARCO	7.23	100.00	4.01	3.44
LLaMA-2-7B	HotpotQA	80.89	100.00	85.96	78.93
	NQ	51.59	100.00	47.10	26.52
	MS-MARCO	65.19	100.00	24.93	5.87
LLaMA-2-13B	HotpotQA	22.15	100.00	7.36	45.78
	NQ	40.00	100.00	35.51	24.64
	MS-MARCO	68.34	100.00	21.20	1.22
LLaMA3-8B	HotpotQA	18.16	100.00	4.73	1.08
	NQ	10.14	100.00	51.01	3.04
	MS-MARCO	2.58	100.00	72.42	3.94
Mistral-7B	HotpotQA	52.80	100.00	0.00	0.74
	NQ	41.74	100.00	0.00	13.33
	MS-MARCO	74.14	100.00	0.00	2.58

Table 4: The performance of each type 3 attack methods (except the Sentence Embedding-based one) and their combination based on majority voting. **Bold** numbers indicate the best performance.

LLM	Dataset	Accuracy (%)			
		Perplexity	Prompt Injection	Breaking Retriever	Combined
GPT-3.5-Turbo	HotpotQA	74.51	75.02	84.98	**85.72**
	NQ	58.04	62.03	**70.51**	69.57
	MS-MARCO	66.94	57.34	50.07	**68.80**
GPT-4o	HotpotQA	50.00	66.61	50.14	**66.64**
	NQ	50.00	**64.13**	62.90	55.94
	MS-MARCO	56.16	**75.29**	52.76	61.50
LLaMA-2-7B	HotpotQA	92.07	91.32	88.86	**97.67**
	NQ	78.19	75.36	56.67	**87.54**
	MS-MARCO	**86.71**	75.04	50.04	86.32
LLaMA-2-13B	HotpotQA	**69.79**	51.01	62.69	63.27
	NQ	**84.28**	71.01	61.74	84.13
	MS-MARCO	**89.22**	72.67	50.00	78.76
LLaMA-3-8B	HotpotQA	55.13	**72.08**	57.73	58.81
	NQ	71.45	77.39	61.96	**81.30**
	MS-MARCO	83.27	**90.47**	54.41	84.56
Mistral-7B	HotpotQA	**83.36**	50.00	55.00	56.72
	NQ	**83.19**	50.00	64.35	61.88
	MS-MARCO	**88.79**	50.00	50.93	60.57

ferred database of S_t as

$$\mathcal{F}(\phi, S_t, R, \mathcal{K}) = \arg\max_{\phi_\alpha} \text{CosSim}(E(Q_\alpha), E(R_{P,\alpha})),$$
$$s.t.\ R = R_P,\ \mathcal{K} = \{0,0,0\},\ \text{and}\ \mathcal{A}(S_t, R, \mathcal{K}) = 1. \tag{16}$$

D.2 LLM Inference Attack

If the LLM used by RAG is identified, the adversary can apply off-the-shelf attack payloads targeting the LLM or perform white-box optimization attacks (for open-source LLMs) (Carlini et al., 2024b; Zou et al., 2023; Liu et al., 2023; Huang et al., 2023). Similar to the database inference attack, we assume that S_t uses one LLM from a set of popular options $\{\theta_\beta\}_{\beta=0}^{g-1}$.

Previous studies (Zhiguang Yang and Hanzhou Wu, 2024; McGovern et al., 2024; Pasquini et al., 2024) have shown that by constructing specific queries, matching the vector space of output Log-Probs, and analyzing lexical and morphosyntactic properties of LLM output, it is possible to infer the specific LLM used by a target service. Given that we lack access to the RAG system's database, our approach follows (Zhiguang Yang and Hanzhou Wu, 2024): the LLM-Only model most similar to the RAG output is likely the LLM used by the RAG system.

Specifically, if a particular LLM is used by RAG, then when RAG cannot answer a query, it is highly likely that the LLM-Only model will also fail to answer it. Therefore, we query the target RAG system S_t and several LLM-Only models $\{\theta_\beta\}_{\beta=0}^{g-1}$ with query Q and compute cosine similarities between the RAG answer R_P and the LLM-Only answers $\{R_{P,\beta}\}_{\beta=0}^{g-1}$ when S_t cannot answer the question. Based on the similarity values, we infer the LLM as

$$\mathcal{F}(\theta, S_t, R, \mathcal{K}) = \arg\max_{\theta_\beta} \text{CosSim}(E(R_P), E(R_{P,\beta})),$$
$$s.t.\ R = R_P,\ \mathbf{N} \in R_P,\ \mathcal{K} = \{0,0,0\},\ \text{and}\ \mathcal{A}(S_t, R, \mathcal{K}) = 1, \tag{17}$$

where $\mathbf{N}$ is an indicator text that RAG could not answer the query. In this work, we set $\mathbf{N}$ = "I don't know."

D.3 Experimental Results

Table 5 summarizes the attack accuracy of our database and LLM inference attacks across multiple m values, with random guess (RG) as a baseline. For the database inference attack, we test three databases from the HotpotQA, NQ, and MS-MARCO datasets. For each LLM, we build three RAG systems using different databases and query each RAG system with queries from various datasets to identify its database. As shown in Table 5a, when using only one query per dataset ($m = 1$), our attack significantly outperforms RG, achieving an average accuracy of 65.98%. Increasing the number of queries further raises accuracy to a maximum average of 68.66%.

In the LLM inference attack, six different LLMs are targeted. For each database, we pair it with six different LLMs to create six RAG systems, then query each RAG system with surrogate dataset queries ($\{\phi_\alpha\}_{\alpha=0}^{g-1}$) to infer the deployed LLM. Table 5b shows the attack results, with each accuracy averaged across surrogate datasets. We observe that increasing the number of queries significantly enhances inference accuracy, reaching an average of 36.04% when $m = 20$. Even with only one query per surrogate dataset, our attack achieves notable performance, outperforming RG by 7.16%.

Take-away. Once a target service is identified as RAG, its fine-grained properties can be accurately inferred, offering valuable information for designing targeted attacks.

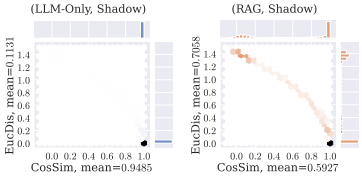
LLM	Accuracy (%)				
	RG	$m=1$	$m=5$	$m=10$	$m=20$
GPT-3.5-Turbo	33.33	67.64	72.90	72.53	70.60
GPT-4o	33.33	68.20	74.82	73.32	70.72
LLaMA-2-7B	33.33	66.68	70.01	70.59	69.70
LLaMA-2-13B	33.33	64.42	66.14	65.91	65.87
LLaMA-3-8B	33.33	64.08	65.82	65.58	65.66
Mistral-7B	33.33	64.85	62.28	60.76	58.92
Average	33.33	65.98	68.66	68.12	66.91

(a) Database inference attack.

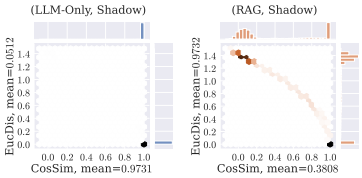
Database	Accuracy (%)				
	RG	$m=1$	$m=5$	$m=10$	$m=20$
HotpotQA	16.67	23.47	28.09	32.06	35.83
NQ	16.67	24.10	29.57	33.59	37.65
MS-MARCO	16.67	23.91	28.07	31.37	34.65
Average	16.67	23.83	28.58	32.34	36.04

(b) LLM inference attack.

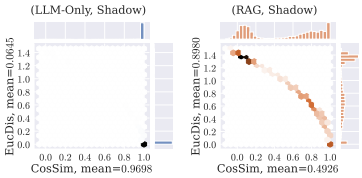
Table 5: Performance of fine-grained RAG property inference, where RG represents random guess.



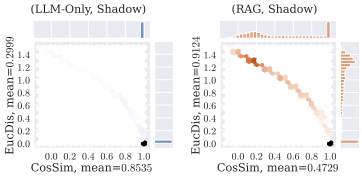
(a) GPT-4o



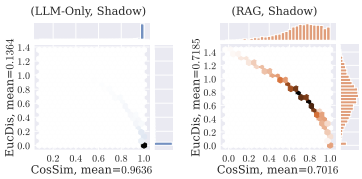
(b) LLaMA-2-7B



(c) LLaMA-2-13B

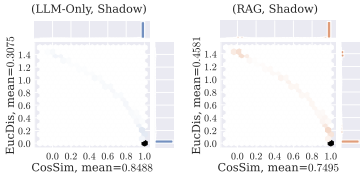


(d) LLaMA-3-8B

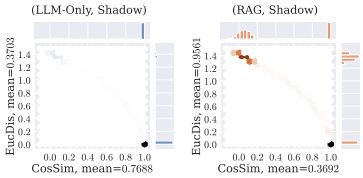


(e) Mistral-7B

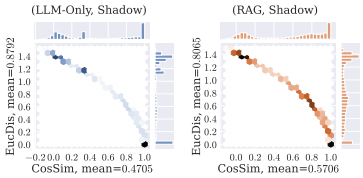
Figure 17: The sentence embedding differences measured by cosine similarity and Euclidean Distance between LLM-Only and shadow LLM, as well as RAG and shadow LLM on HotpotQA dataset. Here multiple LLMs are evaluated.



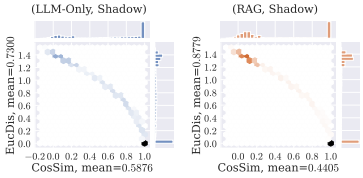
(a) GPT-4o



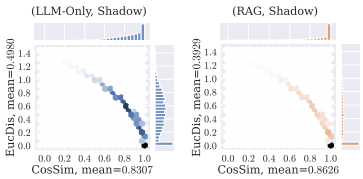
(b) LLaMA-2-7B



(c) LLaMA-2-13B

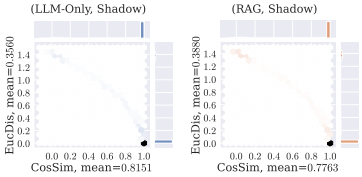


(d) LLaMA-3-8B

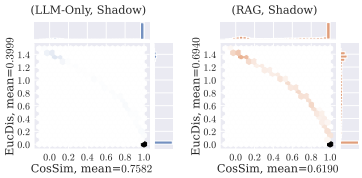


(e) Mistral-7B

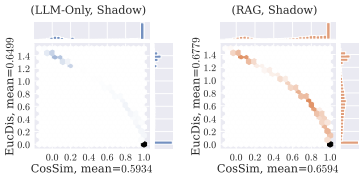
Figure 18: The sentence embedding differences measured by cosine similarity and Euclidean Distance between answers generated by original and prompt-injected queries for LLM-Only and RAG on HotpotQA dataset. Here multiple LLMs are evaluated.



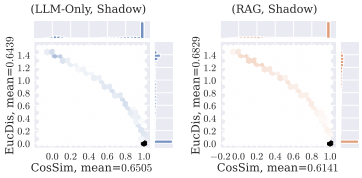
(a) GPT-4o



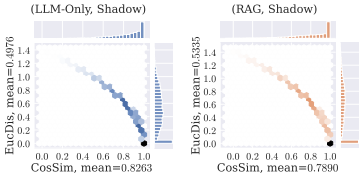
(b) LLaMA-2-7B



(c) LLaMA-2-13B



(d) LLaMA-3-8B



(e) Mistral-7B

Figure 19: The sentence embedding differences measured by cosine similarity and Euclidean Distance between answers generated by original and retriever-broken queries for LLM-Only and RAG on HotpotQA dataset. Here multiple LLMs are evaluated.