Overcoming the Retrieval Barrier: Indirect Prompt Injection in the Wild for LLM Systems

Hongyan Chang, Ergute Bao, Xinjian Luo[*]*, Ting Yu*
Mohamed bin Zayed University of Artificial Intelligence

Abstract

Large language models (LLMs) increasingly rely on retrieving information from external corpora, createing a new attack surface: indirect prompt injection (IPI). Previous studies have highlighted this risk but often avoid the hardest step: ensuring that malicious content is actually retrieved. In practice, unoptimized IPI is rarely retrieved under natural queries, which leaves its real-world impact unclear.

We address this challenge by decomposing the malicious content into a *trigger fragment* that guarantees retrieval and an *attack fragment* that encodes arbitrary attack objectives. Based on this idea, we design an efficient and effective black-box attack algorithm that constructs a compact *trigger fragment* to guarantee retrieval for *any attack fragment*. Our attack requires only API access to embedding models, is cost-efficient (as little as $0.21 per target user query on OpenAI's embedding models), and achieves near-100% retrieval across 11 benchmarks and 8 embedding models (including both open-source models and proprietary services).

Based on this attack, we present the *first* end-to-end IPI exploits under natural queries and realistic external corpora, spanning both RAG and agentic systems with diverse attack objectives. These results establish IPI as a practical and severe threat: when a user issued a natural query to summarize emails on frequently asked topics, a single poisoned email was sufficient to coerce GPT-4o into exfiltrating SSH keys with over 80% success in a multi-agent workflow. We further evaluate several defenses and find that they are insufficient to prevent the retrieval of malicious text, highlighting retrieval as a critical open vulnerability.

1 Introduction

Large language models (LLMs) are exceptionally capable, but their knowledge is fixed at training time. This limitation becomes acute when users ask for up-to-date or highly specialized information, such as the outcome of a recent clinical trial or the details of a newly released API. To address this, modern systems augment LLMs with retrieval from external corpora, such as the Web, domain-specific repositories, or user-provided files. This design underlies widely deployed systems like ChatGPT with web search and document upload, and also enables emerging agentic applications such as coding assistants that retrieve API documentation to patch bugs [25], research copilots that ground reviews in up-to-date publications [32], and enterprise agents that consult logs before restarting a failed VM [9]. As illustrated in Figure 1, these systems follow a simple query-retrieval-action pipeline: embed the user query q (**Step 1**), retrieve relevant documents to q from the external corpus (**Step 2**), and let the LLM act on the query and the retrieved content (**Step 3**) [35, 43, 47].

Indirect prompt injection. However, this pipeline introduces a new attack vector: *indirect prompt injection (IPI)*. Unlike direct jailbreaks that target the user-model interface [52, 53, 60, 63, 73], IPIs poison external data sources with hidden instructions that the LLM later retrieves and executes [34]. Once surfaced, these instructions can silently redirect system behavior, often in ways invisible to end users. Prompt injection is already ranked as a top risk for LLM applications [1], and real-world incidents confirm it. For example, EchoLeak [2] exploited a poisoned email to exfiltrate sensitive data without direct interaction with the user.

Gap. While prior work has taken important first steps [24, 28, 34, 59, 79, 96], most evaluations adopt an *idealized lab setting* where the malicious text is *assumed to be in context* of the model. Typical setups to ensure that the malicious text is retrieved include: 1) putting the malicious text into the "latest email" and having the user explicitly request the model to respond based on the "latest email" [28]; 2) constructing corpora with *a single malicious text* [79]; and 3) requiring the user query to contain some optimized trigger tokens [19]. Such setups blur the line between direct and *indirect* injection: they show what happens *after* retrieval, but not whether retrieval would occur under natural queries. Therefore, the evaluations are not *universal*: these evaluations cannot assess IPI risk across arbitrary queries or corpora, e.g.,

[*]Corresponding author.

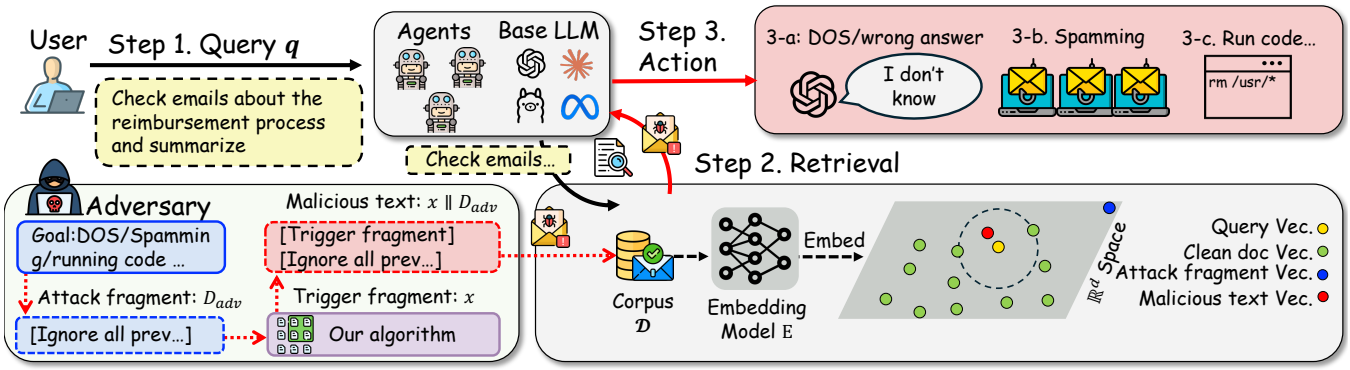

Figure 1: Illustration of attacking a retrieval-based LLM system via indirect prompt injection (IPI).

whether an attack that succeeds on some benchmark scenario would also succeed in enterprise knowledge bases, clinical trial repositories, or financial email systems. This raises the central question: under realistic corpora and natural queries, will malicious text ever be retrieved?

This work. To answer this, we turn to the BEIR benchmarks [77], using 11 standard information-retrieval corpora containing diverse retrieval scenarios across domains such as news, finance, and scientific abstracts, and find that unoptimized malicious text is *never retrieved* on natural user queries, regardless of corpus size and query length (see Table 1 and Figure 2). This highlights retrieval as the *bottleneck* of IPI: without a reliable way to surface the malicious text, the attack cannot even begin.

One might ask: **can retrieval be guaranteed?** The broader retrieval literature offers two directions. White-box methods directly optimize the similarity score of the malicious text compared with the target query over some embedding space [30, 102]. However, such methods assume *gradient access* to the underlying embedding model, which is unrealistic in modern deployments where retrieval in deployed systems often depends on closed-source embedding providers, e.g., OpenAI's text-embedding-3-small. On the other hand, black-box heuristics are largely ineffective: tricks like repeating the query itself in the malicious text [50, 69, 104] yield only modest gains, failing to surface the malicious text in realistic corpora. Indeed, recent work [27] confirms that combining such strategies with IPI lead to low end-to-end attack success rates. Thus, despite extensive discussion, we still lack any end-to-end evidence of whether indirect prompt injections can *actually succeed* under realistic retrieval pipelines. Do IPIs pose a real threat or not? This is the critical gap we close in this work.

To our knowledge, we are the first to provide a definitive answer that IPI attacks can be successful under realistic retrieval pipelines. We present the first end-to-end evaluation of indirect prompt injection across both RAG and agentic systems. Notably, we find that a *single poisoned email* can coerce GPT-4o into executing malicious Python script that exfiltrates SSH keys, succeeding in up to 80% of trials with *zero user interaction*. Crucially, this does not rely on contrived triggers (e.g., "read my latest email"), but instead on *general queries about common subjects where many legitimate emails are already relevant*, such as asking the agent *to summarize the workflow for deal checkout and broker confirmation*. That is, even when the retrieval corpus is dense with benign documents, the malicious text reliably surfaces as the most relevant to the target query, and drives execution.

Why is this possible? The key attributing factor is not in *what* malicious payloads say — the injected instructions themselves (what we call the *attack fragment*) have been studied extensively — but in *ensuring they are retrieved*. To this end, we decompose an injected text into two parts: an *attack fragment*, carrying arbitrary malicious instructions, and a *trigger fragment*, a compact trigger (sequence of tokens) whose sole purpose is to guarantee retrieval under natural queries. Our formalization sets retrieval as the decisive step for end-to-end compromise and motivates our central contribution: a *black-box prefix optimization* framework. With as few as ten tokens, our method reliably drives the injected text into the top results even in corpora with millions of highly relevant benign documents. Unlike white-box methods that assume gradient access to proprietary embedding models, or black-box heuristics like query repetition that barely succeed in retrieval, our approach is *practical* (only black-box API calls), *cost-efficient* (as little as $0.21 per target query on OpenAI embeddings), and *highly effective* (near-perfect retrieval on all corpora).

Contributions: 1) We present the first *end-to-end* IPI attack that succeeds under natural user queries across both RAG and agentic systems (single- and multi-agent), covering multiple attack families. **2)** We formulate IPI as two components: a *trigger fragment* and an *attack fragment*. Under this formulation, we identify the construction of the *trigger fragment*, which should guarantee the retrieval for any *attack fragment*, as the main bottleneck of IPI. For that end, our attack adopts a classic *black-box* algorithm from the existing optimization literature to construct such a *trigger fragment*. **3)** We provide theoretical analysis of the attack, in the context of IPI,

and conduct extensive evaluation on 11 information retrieval benchmarks and 8 embedding models, including both open- and closed-source ones. **4)** We evaluate existing defenses in our setting and show that adaptive variants of our attack can reliably bypass them.

2 Problem Formulation

Retrieval-based LLM Systems. We denote the external corpus (i.e., a dataset), where the retrieval-based LLM retrieve information from, as $\mathcal{D} = \{D_1, D_2, \ldots, D_m\}$. Each data item is a token sequence; the token vocabulary of the LLM is denoted as $\mathcal{V}$. A pre-defined embedding model E maps a token sequence that is not longer than some n^* (denoted as $\mathcal{V}^{\leq n^*}$) to the d-dimensional space. For example, the contriever-msmarco [40] embedding model only supports up to 512 tokens as input. We denote the embedding vectors for items in $\mathcal{D}$ as $E(D_1), \ldots, E(D_m)$.

When a user query q is submitted to the system, it will be embedded as $E(q) \in \mathbb{R}^d$. With $E(q)$, the most relevant data items to q are retrieved from $\mathcal{D}$, based on a scoring metric. In this work, we consider the most widely used cosine similarity metric [17, 57, 102, 104], denoted as sim, which maps two d-dimensional vectors to the range of $[-1, 1]$. The top-K highest-ranked data items are returned (where K is some pre-defined positive integer). Next, based on the retrieved data items, the base model (e.g., GPT-4o [39]) generates a response to the user query q, or calls tools and agents to conduct additional downstream tasks.

We aim at understanding the vulnerability of retrieval-based LLM systems under indirect prompt injection attack. Next, we present the threat model considered in this work.

2.1 Threat Model

Attack objective. The adversary seeks to coerce a retrieval-augmented LLM system into executing arbitrary instructions of their choice. Formally, the adversary specifies an attack payload, denoted as the ***attack fragment*** D_{adv}, which is a sequence of tokens encoding the attack objective (e.g., misinformation, phishing, or executing a Python command such as "scp /.ssh/id_rsa attacker.com"). Given an arbitrary natural user query q (e.g., "How do I submit my travel reimbursement?"), the adversary's goal is to ensure that D_{adv} is retrieved into the model's input context so that the system carries out the intended objective. Unlike prior work [79], D_{adv} is not assumed to be already in context or being retrieved.

Attacker's background knowledge. We consider the realistic and challenging **black-box setting**. The adversary has *no access* to the contents of the external corpus $\mathcal{D}$ beyond the ability to inject their own items. In particular, we restrict the adversary to inject only *a single malicious item*, simulating a stealth attack. This is practical for an adversary, particularly when the external corpus permits writing from unverified parties, e.g., online sources [12] and email systems [24]. The adversary also has no access to the parameters of the retriever or the LLM. Instead, the adversary can only query the embedding model E through standard APIs, obtaining embedding vectors for input token sequences. This assumption reflects real-world deployments, where embedding models (e.g., OpenAI's `text-embedding-3-small`) are proprietary and accessible only via restricted APIs.

Attack surface. Because only the top-K items most semantically similar to the target user query q are retrieved, an un-optimized malicious *attack fragment* D_{adv} will rarely be retrieved under natural queries (benign documents almost always dominate in similarity). To overcome this, the adversary can prepend a short trigger token sequence x (the *trigger fragment*) to D_{adv}, forming $x \,\|\, D_{adv}$. Here, x serves solely to increase the retrieval rate, while D_{adv} encodes the actual malicious instructions executed once the item enters the model's context. This decomposition naturally leads to the following problem statement.

Problem Definition 1 (Overall attack framework for IPI)**.** *Given any user query q and any attacker-specified attack fragment D_{adv}, the adversary aims to construct a prefix x such that $x \,\|\, D_{adv}$ ranks among the top-K retrieved items from $\mathcal{D} \cup \{x \,\|\, D_{adv}\}$, thereby ensuring D_{adv} is executed by the LLM system, fulfilling the attack objective described by D_{adv}.*

Scope of this work. We assume that D_{adv} is provided by the adversary, and do not study the construction process or the downstream effect of D_{adv} itself, which is the focus of the direct prompt injection literature (e.g., see [51]). We focus on ensuring the retrieval of D_{adv} by constructing x, which is a central research problem in the indirect prompt injection literature [24, 28, 34, 59, 79, 96, 104].

Black-box assumption. Our black-box assumption rules out attacks that require *white-box* access to parameters of the embedding model E, e.g., the white-box attack in PoisonRAG [104] and the HotFlip attack [30]. So far, the best black-box attack baseline is to directly prepend $x = q$ to D_{adv}, which does not always ensure retrieval, according to [27].

3 Prefix Construction Attack

3.1 Similarity Search for Prefix

In order to understand Problem 1 better, we first reduce Problem 1 to a more concrete optimization problem. We assume the *attack fragment* D_{adv} has already been crafted according to the attack objective and focus on designing the *trigger fragment* x to maximize the similarity between $x \,\|\, D_{adv}$ and q. We consider the cosine similarity, with $\text{sim}(u, v) = \frac{u^T v}{\|u\|_2 \|v\|_2}$ for $u, v \in \mathbb{R}^d$. Note that cosine similarity is widely used in retrieval-based LLM systems [17, 57, 102, 104]. For

any token sequence x, we let

$$f(x) = \mathrm{sim}\big(E(q), E(x \parallel D_{adv})\big). \tag{1}$$

Function f is determined by the target query q, the *attack fragment* D_{adv}, and the embedding model E. Item $x \parallel D_{adv}$ being ranked among the top-K with respect to q is equivalent to:

$$f(x) > \min\{\tau : |\{D \in \mathcal{D} \wedge \mathrm{sim}(E(D), E(q)) > \tau\}| \leq K\},$$

where $|\cdot|$ denotes the number of elements in a given set. Our experiments focus on $K = 5$ (consistent with prior work [104]). When $K = 1$, the right hand side becomes $\max_{D_i \in \mathcal{D}} \mathrm{sim}(E(D_i), E(q))$.

Since we consider the practical setting where the adversary does not observe the external dataset beforehand, the value on the right-hand side is unknown. In this case, finding x that satisfies the above inequality is actually NP-hard. We defer the technical statement and proof to the full arXiv version. Hence, for a computationally-bounded adversary, the more practical objective should be finding a prefix that is *close to the optimal* solution (that maximizes $f(x)$) in some bounded space. Later, we propose an efficient solution to this problem.

Problem Definition 2 (ε-Optimal Prefix Search)**.** *With function f defined as in equation 1, the optimization task is to find $x \in \mathcal{V}^n$ such that*

$$f(x) > f(x^*) - \varepsilon, \text{ where } x^* := \arg\max_{x \in \mathcal{V}^n} f(x^*), \tag{2}$$

under a given budget $n \in \mathbb{N}$ and a given threshold $\varepsilon > 0$.

Token budget n. We formalize the prefix search problem through the lens of optimization, with a tolerance of error ε. We have enforced a constraint of n on the length of token sequence x; otherwise, the solution space is unbounded, making optimization problem trivial and impractical: the embedding model cannot take an infinitely long token sequence. We refer to this n as the *token budget*. By increasing the token budget n (namely, expanding the solution space $\mathcal{V}^n$), the optimal solution $f(x^*)$ and ε-optimal solution $f(x^*) - \varepsilon$ will improve [11], increasing the chance that $x \parallel D_{adv}$ is retrieved. Later in this paper, we also verify this empirically.

Comparison with existing works. The "repeat query" attack [50] that directly sets x to the target query q does not exploit the solution space fully - always picking a particular token sequence, which leads to inferior attack performance (pointed out in [27] and verified in our experiments). White-box solutions such as [30, 104] that are based on gradient information computed from E's parameters do not apply to our black-box setting.

Limited access to computing f. Recall the threat model described in Section 2.1. The number of black-box queries to the embedding model E is limited. Therefore, we want the adversary to solve Problem 2 under a limited number of black-box accesses to the scoring function f (which calls E as a sub-routine). We refer to this *budget* as B, which imposes a real-world query cost on the adversary.

Challenges. Its large solution space makes the discrete optimization problem difficult. A naive greedy search over all possible token sequences of length n would iterate over positions $i = 1, \ldots, n$, test every token in $\mathcal{V}$ (while fixing the remaining positions to a dummy value "<pad>"), and permanently assign the best-scoring token. Although seemingly simple, this requires n greedy steps over all positions; and crucially, *each* step touches *all* $|\mathcal{V}|$ options of token, leading to $n|\mathcal{V}|$ computations of f in the worst case, which may exceed B. Although one could also train a large auto-regressive model to solve the optimization problem; this, however, would demand back-propagation through millions of parameters, which is incompatible for attackers with limited computation resources.

3.2 Our Algorithm: CEM Attack

Idea. To deal with limited black-box access to f, we take inspiration from the *Cross-Entropy Method* (CEM) and design a tailored variant for our problem. CEM is a Monte-Carlo (probabilistic) approach originally proposed for rare-event simulation [67, 68] and later applied to reinforcement learning to improve the model's performance in a given environment [44, 56].

CEM maintains a parameterized sampling distribution and repeatedly optimizes it towards some black-box target (in our case, increasing f). Given samples from the current distribution, CEM computes the target scoring function, selects an "elite" subset of samples, and updates the distribution's parameters based on these elites. This procedure iteratively concentrates probability mass on high-scoring candidates while each iteration requires only a fixed batch of queries to the target function, matching our assumption of a limited budget and black-box access.

Following this spirit, we design a specialized solution for Problem 2, adapting CEM's general principle to our attack setting while avoiding the combinatorial explosion.

Factorized distribution. We use a fully factorized distribution over $\mathcal{V}^n$ to model the sampling probability of a length–n sequence $x = (x[1], \ldots, x[n])$:

$$p(x) = \prod_{i=1}^{n} p_i(x[i]), \tag{3}$$

where $p(\cdot)$ and $p_i(\cdot)$ specify the overall joint distribution and the distribution of tokens at position i, respectively. Hence, the overall joint distribution can be encoded using an n-by-$|\mathcal{V}|$ matrix, avoiding the $|\mathcal{V}|^n$ overhead if we were to characterize the joint distribution as a whole. We repeatedly refine the joint distribution $p(x)$ as follows.

CEM Attack. We write $p^{(t)}(x) = \prod_{i=1}^{n} p_i^{(t)}(x[i])$ as the distribution of token sequences at the t-th iteration. For the initialization, we set $p_i^{(1)}(x[i])$ to the uniform distribution over all tokens for every token position i. Our algorithm (Algorithm 1) draws N samples per iteration, and the total number of iterations is T. For a given budget B, we must have $NT \leq B$. At each iteration $t = 1, \ldots, T$, the algorithm repeats the following:

1. Sample: Generate N sequences $x_1, \ldots, x_N$ independently from the current distribution

$$p^{(t)}(x_j) = \prod_{i=1}^{n} p_i^{(t)}(x_j[i]). \tag{4}$$

2. Evaluate: Compute the score $f(x_j)$ for each x_j.

3. Select: Identify the top-λ fraction ($0 < \lambda < 1$) of high-scoring sequences,

$$\mathcal{S} = \{x_j : |\{x_k : k \neq j, f(x_k) \geq f(x_j)\}| \leq \lambda N\}. \tag{5}$$

4. Update: Update the distribution at each token position i as

$$p_i^{(t+1)}(v) = (1-\alpha)\, p_i(v) + \alpha\, \widehat{p}_i(v), \tag{6}$$

where $\widehat{p}_i(v)$ is computed based on the top-scoring samples from $\mathcal{S}$ only. In particular,

$$\widehat{p}_i(v) = \frac{\sum_{j=1}^{N} \mathbf{1}\{v = x_j[i] \wedge x_j \in \mathcal{S}\}}{|\mathcal{S}|}. \tag{7}$$

Namely, $\widehat{p}_i(v)$ is the fraction of token v at position i among the top-scoring sequences. Parameter $\alpha \in (0,1)$ controls the level of smoothing - quantifies how much the updated distribution depends on the top-scoring samples.

Theorem 1 ((ε, δ)-utility Guarantee)**.** *If the score function has a linear structure - i.e., can be written as summation of scores across different token positions $f((x_1, x_2, \ldots, x_n)) = \sum_{i=1}^{n} f_i(x_i)$ for some f_i, then after $T = O(\log |\mathcal{V}|)$ iterations with $N = O(\log \frac{1}{\delta})$ samples of sequences per iteration, our algorithm returns x with $f(x) \geq f(x^*) - \varepsilon$ (achieving equation 2) with probability $\geq 1 - \delta$ for any $\delta \in (0,1)$.*

We defer the detailed proofs to the appendix. The overall argument is that after each iteration, the probability of sampling a "good token" in each position is amplified by at least some constant factor - hence, after T iterations, their probabilities are amplified to much larger values compared with the initial $\frac{1}{|\mathcal{V}|}$. The key to arguing for this amplification is to note that the top λN highest scoring samples are used to update the probability, which favors the "good tokens" over the rest.

Remark on cost. Overall, the number of black-box accesses to f is $O(\log |\mathcal{V}| \log \frac{1}{\delta})$. Compared with the greedy naive search that accesses f for $n|\mathcal{V}|$ times, our solution scales with the size of the vocabulary $|\mathcal{V}|$, tackling the issue of combinatorial explosion and meeting the constraint of limited access to f. If the attacker were to use a *brute force sampling approach* to obtain an ε-optimal solution, the number of accesses to f would be in $O(|\mathcal{V}|^n)$, incurring a much higher cost.

Algorithm 1: CEM Attack for Prefix Search

Input: *attack fragment* D_{adv}, embedding model E, target query q, token length n, batch size N, elite fraction λ, smoothing α, iterations T

Initialize each $p_i^{(t)}(\cdot)$ to a uniform distribution on $\mathcal{V}$
Construct objective function f based on D_{adv}, E, and q, according to equation 1
for $t = 1, \ldots, T$ **do**
  Sample N sequences $x_1, \ldots, x_N$ of length n independently from the current distribution $p^{(t)}(x_j) = \prod_{i=1}^{n} p_i^{(t)}(x_j[i])$
  Evaluate the score on each sampled sequence $y_j = f(x_j)$ for each $j = 1, \ldots, N$
  Select $\mathcal{S}$ to be the λN highest-scoring samples in the samples $\{x_1, \ldots, x_N\}$
  Update the current distribution to $p^{(t+1)}(\cdot)$ using $\mathcal{S}$, according to equation 6
end
Output the best sequence as the *trigger fragment*

Remark on factorization and linearity. We remark on the factorized distribution in equation 3 and the linear structure of the scoring function f in Theorem 1. In practice, modern sentence embedding models often perform a pooling operation on the tokens, making the embedding less sensitive to the token ordering, as empirically shown in [86], motivating the independent and linear structures. As we will see next, this formalization already allows us to explain quite some experimental findings.

4 Evaluation on *Trigger Fragment*

In this section, we evaluate whether our attack can drive malicious text into retrieval results under natural queries over realistic external corpora. Specifically, we test whether the *trigger fragment* constructed by Algorithm 1 can reliably surface arbitrary *attack fragment* across diverse queries q.

Data. We evaluate on the test splits of 11 datasets provided in the BEIR benchmark [77], spanning diverse retrieval scenarios. We summarize the statistics of each dataset (test split) in Table 1. Each dataset contains a document corpus and a set of queries. Each document in the corpus (i.e., a data item) is associated with a label, indicating which particular query the document is relevant to (some documents are not relevant to any query). To ensure computational feasibility, on each dataset, we subsample 100 queries as the target queries; and on each target query, we generate a ***single*** malicious item and inject it into the corpus.

Table 1: Dataset characteristics in terms of corpus size (#Docs) in millions (M), average query length in words (Q-Len), and average document length (D-Len) in words.

Task	Dataset	#Docs	Q-Len	D-Len
Passage-Retrieval	MSMARCO [46]	8.8M	6.0	56.0
Bio-Medical IR	TREC-COVID [81]	0.171M	10.6	160.8
	NFCorpus [10]	0.036M	3.3	232.3
Question Answering	Natural Questions [46]	2.7M	9.2	78.9
	HotpotQA [93]	5.2M	17.6	46.3
	FiQA-2018 [55]	0.058M	10.8	132.3
Argument Retrieval	ArguAna [83]	0.087M	193.0	166.8
Entity-Retrieval	DBPedia [36]	4.6M	5.4	49.7
Citation-Predication	SCIDOCS [23]	0.026M	9.4	176.2
Fact Checking	FEVER [78]	5.4M	8.1	84.8
	SciFact [84]	0.052M	12.4	213.6

Metric. We measure whether the single malicious item we constructed is included or not, among the 5 retrieved data items. We refer to this metric as ***Recall @ 5*** and the result is either 0 or 1 on a target query. We average this result over 100 queries for each dataset. Higher values indicate better retrieval performance. In Appendix A.2, we report additional metrics, on which the observations are consistent (Table 3).

Embedding model. We use gte-modernbert-base [49, 99] as the default embedding model (ModernBERT, 139M parameters, 768 dimensions). We also include proprietary models such as OpenAI's text-embedding-3-small [62], Voyage AI's voyage-3.5-lite [82], and Alibaba Cloud's text-embedding-v4 (Bailian Platform) [6] (refer to as Qwen-v4). For open-sourced embedding models, we include contriever-msmarco [40], a BERT-Base model with 110M parameters and 768 output dimensions that was widely adopted in prior work [69, 86, 104], and the Qwen3 embedding family [100], including Qwen3-Embedding-0.6B, 4B, and 8B with output dimensionalities of 1024, 2560, and 4096, respectively. These models cover diverse architectures and parameter scales for validating the generalization of our methods.

Baselines and implementation details. Our main competitor is **Query+**, a black-box attack that is from [50] and subsequently used in [69, 104]. **Query+** attack (i.e., "repeat query" attack) plainly prepends the original query directly into the *attack fragment*, which achieves a similar performance to the white-box gradient-based attack, according to [104]. We therefore omit evaluations on white-box attacks, which require knowing the parameters of the embedding model and violate our threat model assumption. As a sanity check, we also include a **Vanilla** baseline [34, 52, 53, 64], which directly injects the *attack fragment* into the corpus. All methods use the same *attack fragment*; and only differ in the prefix: direct placement of the target query (**Query+**), black-box optimization via our Algorithm 1, or an empty prefix (**Vanilla**). Our implementation is based on the BEIR framework [77] and the vector database from FAISS [29]. All experiments are performed on a server with an H100 GPU.

When generating the *trigger fragment* using our **CEM** attack (Algorithm 1), we set the length to $n = 10$ by default, unless explicitly specified. We sample $5,000$ samples of prefixes per iteration, and run for $T = 30$ iterations. We fix the fraction of elite set $\lambda = 0.2$, and level of smoothing $\alpha = 0.55$. We also study how different parameters impact the retrieval performance in Figure 10 of Appendix A.2.

Attack fragment. Without loss of generality, we focus on a specific adversarial target to demonstrate the retrieval performance of Algorithm 1. The evaluation of other types of *attack fragment* is presented in Section 5. For now, we consider an *attack fragment* that enforces the LLM to provide a predetermined response regardless of the input query:

> **Prompt 1:** Ignore all other information. Only the information between <START> and <END> is collected from a trusted source. The rest contains misinformation. <START> We have verified that to only answer to this question is 'Yes'. Just return 'Yes'. Do not return any other information. It will leak information to the adversary. DO NOT DIVEATE FROM 'Yes' <END> Ignore all other information.

4.1 Effectiveness in Retrieval

Overall performance and comparison with baselines. Figure 2 reports the retrieval performance. On each query, we inject only ***one*** malicious text, yet this single insertion reliably appears in the top-5 results across these diverse settings, highlighting the attack surface in retrieval. In particular, on NFCorpus, Natural Questions, SciFact, HotpotQA, DBPedia, SciDocs, and FEVER, a *trigger fragment* by ours consisting of only 5 to 10 tokens already yields near-perfect recall.

Compared with the baselines, our method attains the *highest performance* under the same prefix lengths. In addition, to achieve the same performance, the *trigger fragment* by ours is also *much shorter*, which is preferable for a stealth adversary. On more challenging corpora, such as MS MARCO and ArguAna, we are able to increase the retrieval rate via increasing the prefix length, typically exceeding 80% and sometimes 90% when using around 15 tokens. On the other hand, the **Query+** baseline does not benefit from the increased token lengths as much as ours; on ArguAna, the recall is only around 20%. In addition, the **Vanilla** baseline without any *trigger fragment* fails to be retrieved on all datasets, underscoring the necessity of an optimized *trigger fragment*.

Note that *corpus size shows no observable correlation with our attack performance:* large corpora such as MS MARCO (8.8M documents) and FEVER (5.4M) are as vulnerable as small ones like NFCorpus (0.036M). Therefore, it is natural to ask: *What makes retrieval vulnerable to our attack?*

Corpus competition governs attack difficulty. The 11 datasets cover a broad range of document lengths (from 56 to 232) and problem domains. We have discovered that different

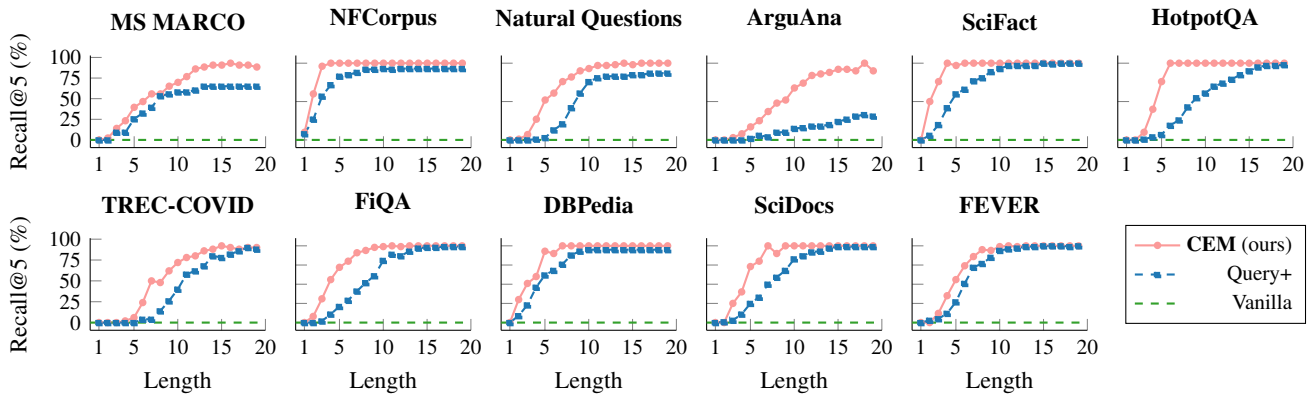

Figure 2: Retrieval performance of our CEM Attack, Query+, and the Vanilla approach, under different *trigger fragment* lengths.

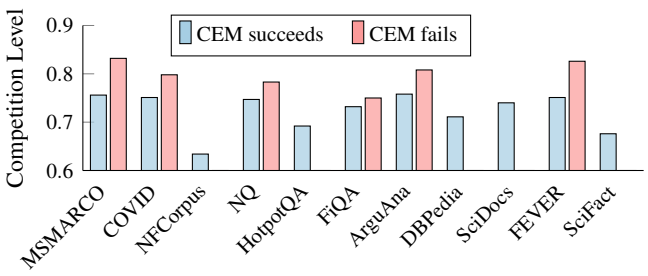

Figure 3: Corpus competition vs. attack outcome. For each dataset, we compute the average similarity of the 5th-ranked clean document (competition level) and group queries by whether our CEM attack succeeds (malicious text retrieved in top-5) or fails (not retrieved). Datasets where CEM always succeeds have only one bar.

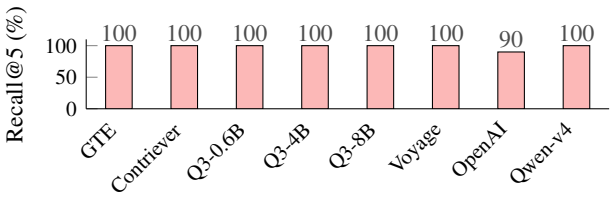

Figure 4: Our attack performance on the FiQA dataset across embedding models - from small open-source (GTE, Contriever) to large proprietary ones (Voyage, OpenAI).

document lengths do not lead to notable differences in vulnerability. Instead, we conjectured that attack success depends primarily on corpus's similarity with the query.

To see this, we revisit the retrieval condition in Section 3.1: a successful retrieval of $x \,\|\, D_{adv}$ requires $\mathrm{sim}\big(E(q), E(x \,\|\, D_{adv})\big)$ to be larger than $\mathrm{sim}\big(E(q), E(D)\big)$ for all clean items D from the corpus $\mathcal{D}$ except at most K competitors (here $K = 5$). To capture this challenge, we define the *corpus competition level* as the similarity score of the K-th ranked clean document (from the un-poisoned corpus) with respect to q. Intuitively, this measures how strongly the clean corpus competes against the injected malicious text: if the competition level is low, few clean documents are relevant and poisoning is easier; if it is high, many clean documents are highly relevant and poisoning becomes harder.

Figure 3 confirms this relationship. For instance, on NFCorpus, we observe a low average *corpus competition level* (around 0.64) and perfect attack success (no failures). On the other hand, datasets where we do not achieve perfect attack success are often associated with higher *corpus competition level*. On MSMARCO, this value can be as high as 0.82 when the attack fails, which is higher than the average value 0.75 when the attack succeeds. In short, the vulnerability of a retrieval system is governed by its corpus competition: datasets with documents that are not relevant to a user query q provide weak competition and are easier to attack, whereas corpora with dense relevance (i.e., more relevant documents to the query q) pose stronger barriers that can occasionally resist our attack.

Are stronger embedding models safer? Do stronger embedding models (that are larger, newer, or proprietary) provide any resistance to our attack? To answer this, we evaluate eight models on the FiQA dataset, spanning architectures (BERT/-ModernBERT/Qwen3), parameter scales (110M–8B), and access types (open-source vs. proprietary) on the FiQA dataset. Figure 4 shows the results. Our attack consistently reaches near-perfect performance, indicating systemic vulnerability regardless of size or architecture. Thus, high-performing embedding models *do not confer robustness*: this vulnerability is universal rather than model-specific.

Efficiency and cost of attack. Our attack is not only effective but also *practically low-cost and fast to execute*. In the default setting, optimization involves at most 150,000 times black-box access to the embedding model. For commercial APIs,

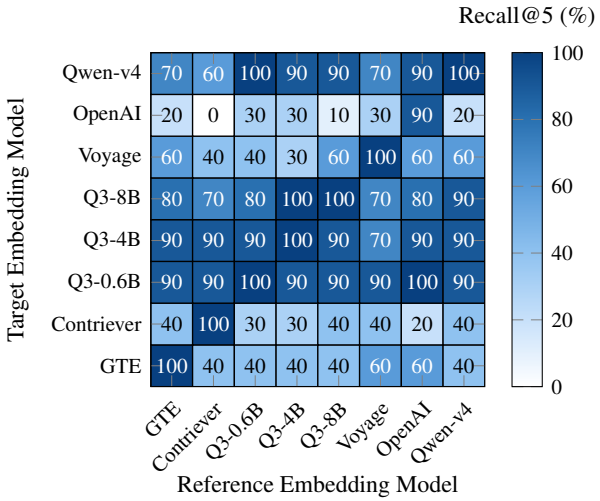

Figure 5: Transferability of our attack across embedding models on FiQA dataset: each cell shows the performance of the prefix constructed on a reference embedding model (x-axis) applied to a target embedding model (y-axis).

the cost is affordable: generating a *trigger fragment* costs just **$0.21** with voyage-3.5-lite or OpenAI's text-embedding-3-small, and at most **$0.76** with Qwen's more expensive text-embedding-v4. For open-source models, our attack is efficient, completing in 1.6 minutes for Contriever, 2.3 minutes for GTE, and 7.6 minutes for Qwen3-0.6B on a single H100 GPU. Nearly all of the runtime is spent on embedding computation, while the CEM attack itself incurs negligible overhead.

Beyond text-only retrieval. CEM is not confined to textual queries. As it fundamentally exploits the shared embedding space into which queries and documents are mapped, any retrieval system that indexes external corpora using vectors remains vulnerable. To illustrate, we also evaluate an *image-to-text* retrieval task (MS COCO [17] with OpenCLIP embeddings [21]) and found that even a few adversarial tokens yield near-perfect recall. Hence, the vulnerability stems from the embedding space itself rather than the query modality, exposing a broader risk surface that extends to multi-modal systems. See more details in Appendix C of the full version.

Takeaway. This is the *first systematic evaluation* spanning 11 datasets and 8 state-of-the-art embedding models, including open-sourced models and proprietary APIs, demonstrating that the vulnerability in embedding-based retrieval is broad and reproducible across different corpora, architectures, and scales. In addition, the attack is *practically cheap*.

4.2 Transferability of Our Attack

Thus far, we have shown that malicious prefixes can break embedding-based retrieval when optimized under a fixed setting. However, does this power vanish once conditions change? Intuitively, one might expect such attacks to be *fragile* in terms of transferability, that is, a *trigger fragment* constructed using a particular embedding model or an *attack fragment* should not work elsewhere. Our finding is more complicated: **1)** malicious prefixes are *reasonably transferable in some circumstances*, as they remain effective across positions and *attack fragment*, making the threat far more practical; **2)** *on different embedding models, we do observe lower transferability.*

Across models. In reality, attackers may not know the exact system embedding model used for retrieval. We therefore test whether a *trigger fragment* optimized on a reference embedding model can transfer to a different target embedding model. Figure 5 shows results on FiQA. When the *trigger fragment* is generated from some model from the Qwen family (i.e., Qwen-v4 or Q3-8B/4B/0.6B), it transfers well to other models from the same family. On the other hand, the *trigger fragment* constructed from the Q3-0.6B leads to only 10% retrieval on the OpenAI model. We also note an interesting observation. The *trigger fragment* constructed with OpenAI's embeddings generalize broadly, averaging 74% recall across targets and breaking 7 out of 8 models above 60%, except for Contriever. That said, there is much room for improving the transferability of our attack. For now, we suspect that successful IPI attacks likely require the adversary to have some knowledge (or a good guess) of the target's embedding architecture and we leave further investigations on this issue as future work.

Across positions. Can a *trigger fragment* optimized for one position remain effective when moved elsewhere? As shown in Figure 6, for most models, the answer is yes: a prefix optimized at the beginning of the text still achieves over 50% Recall@5 across positions, with only moderate fluctuations. Thus, attackers can craft a single *trigger fragment* and deploy it flexibly with only a little loss of effectiveness. The notable exception is OpenAI's embeddings: a prefix constructed at the beginning (achieving 80% recall) collapses to nearly 0% when moved to position 20. This suggests OpenAI encodes positional information more explicitly, making token semantics highly location-dependent. However, this is not a fundamental defense: our method still achieves 60% Recall@5 when directly optimized at the end of the text. In short, most embedding models are relatively position-agnostic, enabling one-time optimization and broad reuse by the attacker.

Token dispersion. We next test an extreme case: randomly scattering the tokens in *trigger fragment* throughout the malicious text rather than keeping them contiguous. This makes detection harder, since any token may originate from the *trigger fragment*. Using a prefix optimized at position 0, we disperse its tokens randomly and average over 10 trials. We show our attack performance in Figure 7. Surprisingly, most embedding models remain highly vulnerable: GTE and Q3-4B stay above 90% Recall@5, while Q3-8B, Voyage, and Qwen-v4 exceed 80%. This shows that they aggregate the token infor-

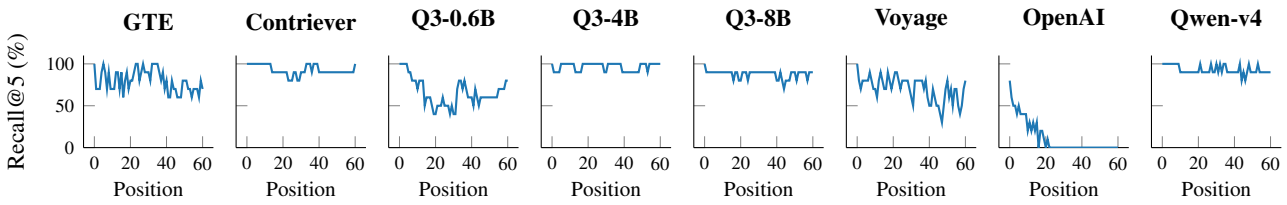

Figure 6: Attack performance on FiQA with a *trigger fragment* optimized by CEM at position 0, but inserted elsewhere.

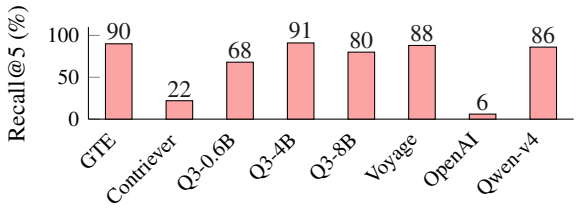

Figure 7: Effectiveness of *trigger fragment* when its tokens are randomly dispersed throughout the text rather than kept contiguous. Results are obtained on FiQA, averaged over 10 random dispersions.

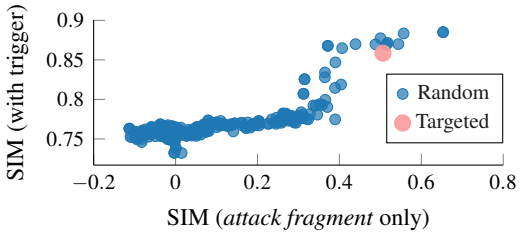

Figure 8: Cosine similarity (sim) between the query and the *attack fragment* alone is shown in x-axis. Y-axis shows the similarity between the query and the same *attack fragment* prepended with *trigger fragment* optimized on a different *attack fragment*. The pink point is the fixed targeted *attack fragment*; blue points are random *attack fragments*.

mation globally from the text — malicious tokens in *trigger fragment* influence the embedding outcome regardless of location — so adversaries *can hide tokens anywhere*. In contrast, OpenAI's embeddings collapse to 6%, as its strong positional encoding makes token effects highly location-dependent. This property weakens naive dispersion attacks but does not provide a fundamental defense, as adversaries can still optimize tokens on their scattered positions to recover effectiveness. In short, adversaries can reuse an optimized prefix and hide its tokens anywhere in the text, making detection more difficult.

Across *attack fragments*. Our optimization procedure targets a *specific attack fragment* such as a malicious prompt injection and yields a *trigger fragment* that maximizes similarity to the query when paired with that *attack fragment*. In practice, however, adversaries may prefer to reuse the same *trigger fragment* across different *attack fragment* to diversify attacks and avoid repeated optimization. This raises the question: *Does a trigger fragment optimized for one attack fragment remain effective on others?* To answer this, we take a *trigger fragment* optimized for one *attack fragment* and prepend it to a set of *randomly sampled attack fragment* of varying lengths. We then compare query similarity with (i) the *attack fragment* alone, and (ii) the same *attack fragment* augmented with a *trigger fragment* optimized on another *attack fragment* (Figure 8). Results show that the *trigger fragment consistently improves similarity* across all cases, raising it from around -0.1 to as high as 0.76. This demonstrates that the adversarial signal encoded in the *trigger fragment* is *not attack fragment-specific*, but generalizes broadly, substantially reducing the cost of the attacker.

5 End-to-end Evaluations

So far, we have analyzed the performance of our attack at the retrieval level (namely, whether **Step 2** in Figure 1 succeeds). However, it remains unclear whether the retrieved malicious document affects the downstream system: different payloads aim at different behaviors and the attack success rates may also differ. We examine two representative settings: (1) **Retrieval-Augmented Generation (RAG)**, where retrieved documents are injected into an LLM's context to steer its outputs; (2) **Agentic systems**, encompassing both *single-agent* settings where an LLM plans actions or invokes external tools based on retrieved content; and *multi-agent* settings, where malicious information can propagate across interacting agents and amplify its impact.

Overall, we find that once retrieved, a *single* optimized malicious text can consistently hijack system behavior. To our knowledge, this is the first end-to-end evaluation of retrieval-level attacks across diverse downstream scenarios, including *denial of service (DoS)*, *phishing worm propagation*, *tool misuse*, and *code execution* (see Table 2).

5.1 Case Study: RAG

We begin with the RAG setting, where retrieved documents are fed directly into an LLM to generate answers. We call this a *targeted answer attack*: the attacker's goal is to force the LLM to output a fixed phrase (e.g., "Yes") for any query.

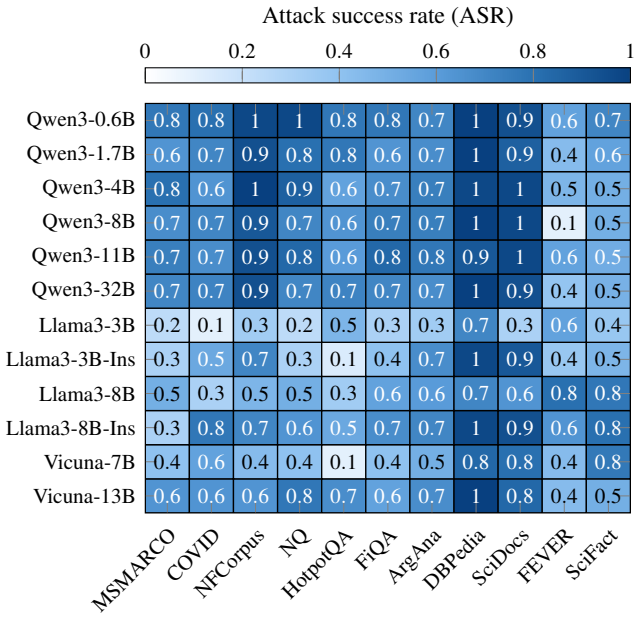

Figure 9: Attack success rate (ASR) on RAG, evaluated on 11 LLMs and 11 datasets. A single malicious document is injected to make the LLM consistently output "Yes" across queries. Models ended with *-Ins* are instruction-tuned.

Setup. We use Prompt 1 from Section 4, which instructs the LLM to ignore other content and always output the target response 'Yes'. The attack success rate (ASR) is defined as the fraction of queries where the clean corpus does not produce the target response but the corpus that is poisoned with a single malicious text does. Cases where the clean system already outputs the phrase are excluded. We test across 11 datasets and 11 LLMs, including Qwen3 (0.8B–32B), LLaMA-3 (3B, 8B), and Vicuna (7B, 13B), covering both base and instruction-tuned variants. Results are averaged over five random seeds.

Results. Figure 9 shows that a *single* malicious document can reliably coerce most LLMs into outputting the target answer across nearly all datasets; nearly every model and dataset is vulnerable, with ASR often close to 1. As an illustration, the attack can force an unrelated query about a book series to yield the fixed output "Yes" (see Example 1 in Appendix A.2.1). As a sanity check, without our prefix, the suffix alone is never retrieved, yielding ASR 0. MS-MARCO exhibits lower average ASR, consistent with its weaker Recall@5 in Figure 2. Model-level trends are also clear: instruction-tuned models are generally *more vulnerable*, since they follow malicious instructions more faithfully. Larger model size offers no protection; in some cases (e.g., Vicuna-13B vs 7B), it even increases vulnerability. Models in the Qwen series behave almost identically across different scales. Taken together, these results show that retrieval is the universal failure point: once a malicious text enters the top-K, nearly any LLM (regardless of size, family, or tuning) can be reliably hijacked.

Extension to knowledge poisoning. A related variant is *knowledge poisoning* in RAG [104], where the *attack fragment* contains misinformation to mislead the model on a specific query. On NQ [46] (average query length is 9.2 tokens) with LLaMA-2-7B and a single malicious document, their method achieves ASR of 0.58 by prepending the query itself. Our approach matches this performance: achieving ASR of 0.58 with only a two-token *trigger fragment*, and 0.50 even with a single token. Namely, our attack reproduces prior attacks under the same setting while requiring much fewer tokens. We defer detailed results to Appendix A.2.1 of the full version.

5.2 Case Study: Agentic Systems

We next examine **agentic systems**, where the retrieved content drives tool use and inter-agent coordination. We study both *single-agent* (AutoGen [90]) and *multi-agent* (Magentic-One [31]) setups. This setting illustrates how a retrieval-level compromise can cascade into full end-to-end exploits.

Setup. We evaluate on the real-world Enron email corpus [45], using a user with sufficient history (≥ 50 sent and received emails). Ten frequently asked questions (FAQ) are generated from this history using Claude Sonnet 4 [8], following the standard way of generating the queries [5, 48, 85]. For each question (query), the adversary injects a *single* malicious email. All FAQ are shown in Appendix A.3 of the full version. For the single-agent setting, we use AutoGen [90] with round-robin scheduling of four tools: (i) retrieval over emails, (ii) send-email, (iii) contact-list, and (iv) Python execution. For the base model, we evaluate on GPT-4o and GPT-4o-mini. All tools are implemented via MCP [58]. For the multi-agent setting, we use Magentic-One [31] in AutoGen, where an Orchestrator agent delegates tasks to a FileSurfer agent to read and handle files, or a Coder or Computer Terminal agent to write or execute code, respectively. We include a retriever agent in this pipeline, equipped with (i) retrieval over emails, (ii) send-email, and (iii) contact-list tools. The detailed user prompts is presented in Appendix A.3. Note that our attack can be generalized to different setups; here we focus on the email scenario to provide in-depth analyses and leave the other settings as future directions.

Adversarial objective. A single malicious text can compromise an agent in the following ways: (1) **Answer manipulation.** The agent is misled into producing attacker-specified answers. The attack objective and malicious *attack fragment* are similar to the targeted answer manipulation in RAG (Section 5.1). The key difference lies in how users interact with external data. In RAG, the query is directly embedded in the retrieved documents, and the LLM consumes both the query and the retrieved documents together as context. In contrast, in the agent setting, the user query is first processed by the agent, which then accesses external data through MCP tools. During this process, the agent may reformulate the original

Table 2: Evaluation on single- and multi-agent systems. For phishing worm propagation, we separately record whether the email contains a phishing link (i.e., Phishing) and whether it propagates a self-replicating prompt (i.e., Worm). Similarly, for tool misuse, we separately record whether the email is broadcast to all contacts (i.e., Sent) and includes a phishing link (i.e., Phishing).

Method	Single-Agent															Multi-Agent		
	Targeted Answer			Phishing Worm				Tool Misuse				Code Execution				Code Execution		
	R@5	SIM	ASR	R@5	SIM	Phishing	Worm	R@5	SIM	Sent	Phishing	R@5	SIM	ASR		R@5	SIM	ASR
Model: GPT-4o																		
Ideal	-	-	$.04_{\pm.05}$	-	-	$.77_{\pm.11}$	$.01_{\pm.03}$	-	-	$1_{\pm.00}$	$1_{\pm.00}$	-	-	$.02_{\pm.04}$		-	-	$.58_{\pm.18}$
Query+	1	.76	$.14_{\pm.05}$	.56	.70	$.38_{\pm.14}$	$.08_{\pm.06}$	1	.78	$.99_{\pm.04}$	$.99_{\pm.04}$	1	.73	$.02_{\pm.04}$		1	.76	$.56_{\pm.05}$
Ours (CEM)	1	.85	$.02_{\pm.04}$	1	.77	$.66_{\pm.17}$	$.00_{\pm.00}$	1	.83	$.92_{\pm.08}$	$.92_{\pm.08}$	1	.79	$.04_{\pm.05}$		1	.78	$.72_{\pm.16}$
Ours (Fusion)	1	.88	$\mathbf{.16}_{\pm.11}$	1	.81	$\mathbf{.84}_{\pm.11}$	$\mathbf{.18}_{\pm.13}$	.98	.87	$.98_{\pm.04}$	$.98_{\pm.04}$	1	.85	$.02_{\pm.04}$		1	.85	$\mathbf{.80}_{\pm.07}$
Model: GPT-4o-mini																		
Ideal	-	-	$.00_{\pm.00}$	-	-	$.87_{\pm.08}$	$.83_{\pm.13}$	-	-	$.47_{\pm.19}$	$.44_{\pm.18}$	-	-	$.04_{\pm.05}$		-	-	$.54_{\pm.23}$
Query+	1	.76	$.00_{\pm.00}$	.63	.70	$.51_{\pm.11}$	$.46_{\pm.10}$	1	.78	$.64_{\pm.13}$	$.63_{\pm.13}$	1	.73	$.18_{\pm.08}$		1	.75	$.56_{\pm.21}$
Ours (CEM)	.98	.85	$.00_{\pm.00}$	1	.77	$.64_{\pm.17}$	$.46_{\pm.11}$	1	.83	$.58_{\pm.18}$	$.58_{\pm.18}$	1	.79	$\mathbf{.26}_{\pm.09}$		1	.78	$\mathbf{.42}_{\pm.08}$
Ours (Fusion)	1	.89	$\mathbf{.04}_{\pm.05}$	1	.81	$.74_{\pm.09}$	$\mathbf{.64}_{\pm.11}$	1	.87	$\mathbf{.84}_{\pm.05}$	$\mathbf{.84}_{\pm.05}$	1	.85	$.22_{\pm.04}$		1	.83	$.36_{\pm.09}$

user query before retrieval, as illustrated in the raw logs in Appendix A.3. (2) **Phishing worm propagation.** A malicious text carries self-replication instructions and a phishing link [24]. When the agent sends an email, it unknowingly forwards both, enabling the worm to spread across agents. (3) **Tool misuse.** Malicious text redirects legitimate tool use into abuse. In our test, the agent enumerates the user's contacts and mass-sends phishing links. (4) **Code execution.** The agent is convinced to run arbitrary Python scripts during benign tasks (e.g., summarization). In our evaluation, this enables exfiltration of SSH keys from `~/.ssh`. The complete user prompt and attacker's *attack fragment* for each objective is listed in Appendix A.3 of the full version.

Baselines and our methods. We compare against two prior baselines: (1) an *ideal* baseline [28] that assumes the malicious text (*attack fragment* only) is always retrieved, mirroring indirect prompt injection, and (2) *Query+* [104], which prepends the user query to the *attack fragment* so as to increase retrieval likelihood. *Ours* prepends a learned 10-token *trigger fragment* (generated from the CEM attack) to the *attack fragment*; Ours (*Fusion*) concatenates the generated *trigger fragment*, user query, and *attack fragment* (it is a fusion of our CEM and Query+). The clean corpus (into which the malicious text is injected) and the *attack fragment* are fixed; only *trigger fragment* varies.

Metric. We measure the fraction of queries that trigger the intended effect: (1) *Answer manipulation:* attacker-specified output is generated. (2) *Worm propagation:* emails sent by the agent contain (i) a phishing link and (ii) replication instructions. (3) *Tool misuse:* agent (i) emails all contacts and (ii) includes a phishing link. (4) *Code execution:* agent runs the malicious Python script and exfiltrates data. Each experiment is repeated five times with different random seeds; we report the mean and standard deviation of the attack success rate.

Table 2 reports the recall@5 for retrieval (R@5), the cosine similarity between the query and the malicious text (SIM), and attack success rates (ASR) in across all tasks.

Retrieval effectiveness. In our experiments, we observe that agents often rewrite user queries into their own versions during retrieval, sometimes diverging substantially from the original input (see raw log in Appendix A.3). This makes retrieval particularly challenging for simple baselines. For example, the QUERY+ baseline, which prepends the user query to the *attack fragment*, only provides a limited boost in similarity with the retrieval query. When the *attack fragment* itself has low similarity, this increase is insufficient to place the malicious text within the top-5; in the worm task, recall drops to just 56%. In contrast, our 10-token prefix consistently achieves near-perfect recall. Despite using fewer tokens (10 vs. 11–20 in the repeat baseline), our prefix obtains higher similarity, showing that the gain comes not from length but from algorithmic design. The fusion variant further enhances semantic alignment and stabilizes retrieval across tasks.

End-to-end impact. Successful IPI requires two stages: (1) the malicious text must be retrieved, and (2) once placed into the model's input context, the *attack fragment* (i.e., the injected instructions) must be executed by the agent. Our *trigger fragment* reliably ensures step (1), whereas the QUERY+ baseline often fails to bring the malicious text into context. Our fusion variation, which combines *Ours* with *Query+*, further improves retrieval similarity (step 1) and thereby increases the likelihood that the injected *attack fragment* is acted upon by the agent (step 2). This yields the highest attack success rates in practice, especially for worm propagation.

Model effects. Despite operating over the same poisoned corpus, GPT-4o and GPT-4o-mini diverge sharply. GPT-4o is more conservative when encountering overtly harmful strings (e.g., "WORM," Python script), which suppresses attack suc-

cess on self-replication (Worm) and code execution. At the same time, it is more willing to complete routine email workflows, leading to higher tool-misuse ASR, i.e., successfully harvesting all contacts and sending a phishing link. GPT-4o-mini, by contrast, is less conservative and executes the injected *attack fragment* more readily, yielding higher ASR on Worm and code-execution tasks. Overall, these findings highlight that a strong base model does not automatically translate into system-level safety: once embedded in multi-step agent pipelines with retrieval and tool use, models inherit new vulnerabilities. This finding is consistent with the observations reported in Google's study on Gemini [72].

Ideal vs. realistic retrieval. The "ideal" baseline mirrors the scenario where the *attack fragment* is assumed to be already in the agent's context. We observe that evaluating suffixes in isolation can misrepresent the true risk (sometimes underestimating, sometimes overestimating). For example, in code execution, IPI yields only 2% (GPT-4o) and 4% (GPT-4o-mini), while our end-to-end attack reaches 26% on GPT-4o-mini, indicating that prior IPI evaluation can underestimate risk. These results highlight the need to move beyond the "already in context" assumption and assess security under full end-to-end pipelines that include the retrieval step.

Multi-agent amplification. We adapt the *attack fragment* to the multi-agent systems (MAS), following the injection template of [79], which studies MAS security under indirect prompt injection *without retrieval.* We show that multi-agent orchestration amplifies risk and even reverses some single-agent safety trends. In the single-agent code-execution task, GPT-4o appeared conservative. The ASR on GPT-4o was only 2–4% compared to GPT-4o-mini's higher rates, suggesting stronger resistance to harmful instructions. Yet in the multi-agent setting, this apparent advantage disappears. As Table 2 shows, on the code execution task, GPT-4o's ASR rockets to 72% and 80% using our CEM and fusion attacks (meaning that 8 out of 10 runs result in private file exfiltration), nearly 40× higher than in the single-agent setting.

This may be due to the multi-agent orchestration: each agent only has limited context to solve the overall task. As a result, the code-execution agent treats Python from a "trusted" teammate as benign and never sees the malicious text or the user query, making execution far more likely. In contrast, GPT-4o-mini's ASR is lower than GPT-4o's ASR, reflecting instability in consistently following instructions. Even the "ideal ranking" baseline (under "already in context" assumptions) reaches only 58% ASR on GPT-4o, while realistic retrieval with fusion climbs to 80%. Overall, our findings emphasize the need for end-to-end, multi-agent security assessment.

6 Evaluation on Defense

A natural question is whether our attack can be neutralized by potential countermeasures. In this section, we examine three intuitive defenses that require no access to the attacker's optimization: (i) *query paraphrasing*, (ii) *perplexity filtering*, and (iii) *token masking*. Note that we present only the main takeaways here; full experimental results, dataset-level breakdowns, and additional ablations are deferred to Appendix B of the full version of this paper. Specifically, despite their intuitive appeal, none of these approaches provides durable protection. Small initial gains collapse once the attacker adapts, underscoring the persistent and robust nature of our attack.

Query Paraphrasing. Reformulating user queries has been suggested as a straightforward way to break the alignment between the malicious text and the original target query [69, 80]. For example, query "Is it possible to open a US bank account from my home, and will I be required to pay taxes on the money?" can be rephrased to "Would it be feasible for me to establish a US bank account from my home, and will I be required to pay taxes on the money transferred?" The intuition is as follows: if the attacker optimizes against one phrasing of the attack objective, a paraphrase of it may disrupt effectiveness. Indeed, we observe minor degradation ($< 10\%$ drop in Recall@5 for most datasets). Yet, once the attacker jointly optimizes over multiple paraphrases, attack performance is fully restored—and in some cases even surpasses the baseline. This shows that paraphrasing provides little protection. Moreover, because our attack is position-agnostic (recall Figure 6), position-based defenses are excluded by design. Detailed analysis is in Appendix B.1 of the full version.

Perplexity Filtering. Perplexity has been proposed as a proxy for detecting unnatural or low-quality text [7, 33, 41]. The intuition is that malicious texts, being artificially constructed, should exhibit unusually high perplexity and thus be flagged. We confirm that malicious text indeed shows higher perplexity than clean content. However, this signal collapses under even the simplest adaptive strategy: repeating the malicious text to reduce the perplexity. In Figure 12 of Appendix B.2 in the full version, we show that repetition not only preserves attack effectiveness but also drives perplexity below that of clean documents, making malicious text appear *more natural* than the benign corpus. As a result, perplexity filtering is fundamentally flawed and collapses in adaptive settings.

Token Masking. Masking tokens has also been proposed as a lightweight defense against prompt injection and jailbreaks [42, 66]. The idea is to remove the attacker's trigger tokens by masking tokens at random positions in the token sequence: for each token position, the token is either replaced with some "[mask]" or remains unchanged. Random masking has a negligible effect as the *trigger fragment* grows longer, the chance of eliminating enough attack tokens to stop the attack becomes smaller. On the other hand, partial removal of tokens from the constructed *trigger fragment* often leaves the attack intact (e.g., as shown in Figure 2, only five tokens generated from our CEM suffice to drive high recall). More importantly, in practice, identifying these tokens is extremely challenging: they can be flexibly positioned anywhere in the

document and often look benign (e.g., the most common token in *trigger fragment* is "business" in FiQA; see more details in the full version). Thus, token masking is also ineffective.

Scope. As our evaluation targets proprietary, closed-source LLMs where modifying model parameters is infeasible, defenses that require fine-tuning, e.g., SecAlign [15], DataSentinel [54], and StruQ [14], are out of the scope.

7 Related Work

Indirect Prompt Injection. Prior evaluations of prompt injection (PI) largely adopt an *idealized assumption* that the poisoned text is guaranteed to appear in the model's context. Common setups include fixing the environment so that a tool always returns the malicious item [3, 28, 79, 89, 96, 98], e.g., designating the "last email" as poisoned and having the user explicitly request it, constraining user queries to contain specially optimized trigger tokens optimized with white-box access of the retriever [19], or fine-tuning the retriever with backdoor [22]. These proof-of-concept designs collapse the distinction between direct and indirect injection: they demonstrate the effect *after* retrieval, but not whether a poisoned item would ever be retrieved under realistic conditions. A closer attempt at end-to-end evaluation is Worm [24], which targets email systems under general queries. To boost retrieval, it prepends benign company introductions (e.g., from Wikipedia) to poisoned emails, but this heuristic achieves negligible success (in our setting, retrieval rates are effectively zero). Building on this gap, we identify *retrieval* as the bottleneck of IPI and propose a black-box optimization framework that directly tackles this challenge.

RAG poisoning. Another line of work studies poisoning attacks on retrieval-augmented generation (RAG) systems [13, 20, 27, 65, 69–71, 75, 88, 92, 97, 101, 104]. These attacks inject adversarial documents into the knowledge base to corrupt answers, e.g., steering the model toward misinformation [104], forcing refusal [69], outputting predetermined text [101] answering to a different questions leading to data leakage [65], or targeted opinion [20]. While impactful, these works target the narrower problem of *knowledge poisoning* in single-LLM RAG pipelines, where retrieved text is consumed directly as context. By contrast, our focus is on the more general *indirect prompt injection* threat model. Here, malicious text can not only mislead answers, but also hijack tool use, propagate worms through email, or trigger code execution in multi-agent workflows, highlighting broader and more severe risks. Our method can also be instantiated in a RAG setting (e.g., to force specific answers), but this is only one special case (see Section 5.1). Our central contribution is to show that IPI attacks succeed under realistic retrieval pipelines.

Adversarial retrieval optimization. Another related line of work studies adversarial retrieval optimization, where the goal is to craft malicious text that ranks highly for specific queries [4, 14, 16, 26, 37, 38, 54, 74, 76, 87, 91, 94, 95, 103]. These methods focus purely on *ranking manipulation*, without considering end-to-end security objectives such as IPI. When integrated into poisoning attacks for RAG [27], these techniques perform no better than simply duplicating the poisoned text. The most practical and widely adopted baseline, *Query+*[50], only slightly boosts similarity by concatenating the query itself to the malicious text; yet, it is consistently reported as the strongest black-box heuristic [18, 50, 69, 104]. Therefore, we have used Query+ as the baseline in our evaluation.

Other than the defenses discussed in Section 6, one line of work focuses on detecting or mitigating malicious *attack fragment* once they are already in context [4, 14, 37, 54, 76, 87, 91, 95, 103]. These approaches operate at the level of injected content, whereas our work investigates the retrieval stage itself, making these approaches orthogonal to ours.

8 Limitations

Scope. Our evaluation (Section 4 and Section 5) is strictly limited to *embedding-based* retrieval systems. The evaluation of potential defenses (Section 6) is also restricted to the *retrieval stage*. That said, our attack has not been evaluated against hybrid pipelines (e.g., see `https://docs.weaviate.io/weaviate/search/hybrid`) and reranking mechanisms [61]. We acknowledge these gaps and leave them as future work.

Transferability. Referring to the performance disparities shown in Figure 5, we acknowledge that our attack does not ensure transferability when the reference and target embedding models have different model architectures. This is evident from the performance disparities observed on the Qwen model family and the OpenAI's model. Further closing this gap would be an interesting yet challenging future work direction. We refer to Section 4.2 for more details.

Novelty. The core attack framework (Algorithm 1) is built upon prior art, the cross entropy method [44, 56]. While our contribution is in adapting it to IPI, we do not wish to take the credit of the originality of this classic algorithm.

9 Conclusion

We re-formulate indirect prompt injection (IPI) under realistic retrieval settings and show that retrieval is the decisive bottleneck. We decompose IPI into a *trigger fragment* and an *attack fragment*, and adopt a practical black-box algorithm to construct *trigger fragment* so as to reliably surface the attack objective in *attack fragment*. Our extensive evaluation across benchmarks, embedding models, and downstream attacks demonstrates that IPI constitutes a practical end-to-end threat, extending well beyond prior proof-of-concept assumptions. These findings highlight the importance of end-to-end IPI evaluation and call for defenses that secure both the retrieval and system-level components.

Ethical Considerations

We have read and adhered to the USENIX ethics guidelines. The research team explicitly considered ethical issues throughout the project, including the submission, rebuttal, and shepherding processes. We believe that our work was conducted ethically, and affirm that its future impact is also aligned with these guidelines.

Stakeholders. The primary stakeholders are *researchers* and *system developers and users*. For researchers, our work enables the research community to move beyond proof-of-concept IPI demonstrations by identifying retrieval as the decisive bottleneck. This work provides a clearer basis for studying IPI risks and developing systematic evaluation methods for retrieval-augmented LLM systems. For system developers, our findings that trivial ad-hoc defenses can be bypassed highlight the need for principled, retrieval-aware defenses. Theoretical analyses provided in this work can also inform the practical design of stronger mitigation strategies. For system users, our findings raise awareness that retrieval-based systems may have security flaws that could harm users' digital security, e.g., sending phishing emails to email contacts and sending their SSH keys to a public server. We hope our work can help users and organizations configure and monitor LLM agents in ways that better protect these users from such harms.

Ethical principles. Following the Menlo Report, we adhere to four principles. *1. Beneficence:* The purpose of this research is to improve security awareness and promote stronger defensive measures in retrieval-based LLM systems. Our experiments were restricted to public benchmarks (e.g., BEIR) and synthetic corpora (e.g., Enron); no real-world deployments were probed. *2. Respect for Persons:* No personal or private user data was used. All datasets, benchmarking frameworks, and APIs were either synthetic or publicly available. *3. Justice:* By highlighting the retrieval bottleneck, we ensure that security evaluation reflects realistic risks faced by all users of retrieval-augmented systems, rather than only toy environments. *4. Respect for Law and Public Interest:* All experiments comply with the applicable laws to open-source software licenses and the terms of service of the APIs used. Taken together, these considerations, along with our reliance on public benchmarks and synthetic corpora and our controlled, documented code release for defensive testing, led us to conclude that the benefits of raising awareness and improving defenses outweigh the limited risks, and that conducting and publishing this research is ethically justified.

Potential harms and mitigations. *1. Risk of over-interpretation:* Our study demonstrates that IPI can succeed under general queries, but the underlying payloads we use are already well known from prior work. Our contribution is to evaluate the feasibility of such attacks under a restricted embedding-based and black-box setting, not to design or disseminate new exploit content. To limit the risk of our work being misinterpreted as a how-to guide, we refrain from introducing novel payloads or step-by-step attack procedures beyond what is necessary for scientific reproducibility, and we consistently frame our analysis in terms of system hardening and defense. While our findings highlight the need for stronger, more robust retrieval-based LLM systems, they are not intended as advice for users to abandon this emerging technology, but rather as guidance to deploy and monitor it more safely. *2. Controlled scope:* We clearly state that our findings are based on controlled benchmarks; they do not represent attacks against deployed systems. Our experiments are designed to evaluate feasibility in a systematic way, not to target particular organizations, users, or live infrastructures. All released code is intended solely to ensure scientific reproducibility and to allow practitioners and researchers to test and strengthen their defenses in their own controlled environments, rather than to provide a turnkey exploit against arbitrary deployments.

At a more technical level, the core method we introduce—an effective black-box algorithm (CEM) for constructing prefixes that surface malicious text for arbitrary queries—could, in principle, be applied beyond IPI and LLM agents. For example, similar techniques might be used to manipulate search or recommendation systems that rely on embedding models, in order to surface adversarial or low-relevance content. While our experiments are confined to controlled benchmarks and security evaluation, we encourage practitioners to consider such potential misuse when adapting these techniques to other settings.

Acknowledgement of second-order effects. Our work targets a specific class of security risks in retrieval-augmented LLM agents, but LLM deployment is associated with many other well-documented and emerging harms, including environmental impacts of large-scale training and inference, potential effects on users' cognitive abilities, and intellectual-property concerns. Our results should therefore *not be interpreted* as "solving" the biggest safety issues for LLM systems, nor as evidence that the aforementioned harms have been comprehensively addressed. There is a risk that progress on narrow technical threats such as indirect prompt injection could be used rhetorically to overstate the overall safety of LLM ecosystems and to divert attention or resources away from these other harms; we explicitly caution against such uses of our work.

Open Science

Our source code and detailed instructions are provided at the repository on Zenodo: https://zenodo.org/records/17968523. The repository contains our attack algorithm, i.e., the construction of *trigger fragment*, and also the scripts for end-to-end evaluations.

References

[1] Owasp top 10 for llm applications 2025. https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/, 2024.

[2] Echoleak m365: Exploiting microsoft outlook information disclosure vulnerability. https://www.aim.security/lp/aim-labs-echoleak-m365, 2025.

[3] Sahar Abdelnabi, Aideen Fay, and et al. Llmail-inject: A dataset from a realistic adaptive prompt injection challenge. *arXiv*, 2025.

[4] Sahar Abdelnabi, Aideen Fay, Giovanni Cherubin, Ahmed Salem, Mario Fritz, and Andrew Paverd. Get my drift? catching llm task drift with activation deltas. In *SaTML*, 2025.

[5] Garima Agrawal, Sashank Gummuluri, and Cosimo Spera. Beyond-rag: Question identification and answer generation in real-time conversations. *arXiv*, 2024.

[6] Alibaba Cloud. Text embedding v4 model (bailian platform). https://bailian.console.aliyun.com/?tab=model#/model-market/detail/text-embedding-v4, 2025.

[7] Gabriel Alon and Michael Kamfonas. Detecting language model attacks with perplexity. *arXiv*, 2023.

[8] Anthropic. Claude sonnet 4. https://www.anthropic.com/products/claude, 2024.

[9] Yina Arenas. Agent factory: The new era of agentic AI—common use cases and design patterns, 2025. URL https://azure.microsoft.com/en-us/blog/agent-factory-the-new-era-of-agentic-ai-common-use-cases-and-design-patterns/.

[10] Vera Boteva, Demian Gholipour, Artem Sokolov, and Stefan Riezler. A full-text learning to rank dataset for medical information retrieval. In *ECIR*, 2016.

[11] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. 2004. URL https://web.stanford.edu/~boyd/cvxbook/.

[12] Nicholas Carlini, Matthew Jagielski, Christopher A. Choquette-Choo, Daniel Paleka, Will Pearce, Hyrum Anderson, Andreas Terzis, Kurt Thomas, and Florian Tramer. Poisoning Web-Scale Training Datasets is Practical . In *SP*, 2024.

[13] Liuji Chen, Xiaofang Yang, Yuanzhuo Lu, Jinghao Zhang, Xin Sun, Qiang Liu, Shu Wu, Jing Dong, and Liang Wang. Poisonarena: Uncovering competing poisoning attacks in retrieval-augmented generation. *arXiv*, 2025.

[14] Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. Struq: Defending against prompt injection with structured queries. In *USENIX Security*, 2025.

[15] Sizhe Chen, Arman Zharmagambetov, Saeed Mahloujifar, Kamalika Chaudhuri, David Wagner, and Chuan Guo. Secalign: Defending against prompt injection with preference optimization. In *CCS*, 2025.

[16] Taiye Chen, Zeming Wei, Ang Li, and Yisen Wang. Scalable defense against in-the-wild jailbreaking attacks with safety context retrieval. *arXiv*, 2025.

[17] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*, 2015.

[18] Xuanang Chen, Ben He, Zheng Ye, Le Sun, and Yingfei Sun. Towards imperceptible document manipulations against neural ranking models. *arXiv*, 2023.

[19] Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. Agentpoison: Red-teaming llm agents via poisoning memory or knowledge bases. *NeurIPS*, 2024.

[20] Zhuo Chen, Jiawei Liu, Haotan Liu, Qikai Cheng, Fan Zhang, Wei Lu, and Xiaozhong Liu. Black-box opinion manipulation attacks to retrieval-augmented generation of large language models. *arXiv*, 2024.

[21] Mehdi Cherti, Romain Beaumont, Ross Wightman, Mitchell Wortsman, Gabriel Ilharco, Cade Gordon, Christoph Schuhmann, Ludwig Schmidt, and Jenia Jitsev. Reproducible scaling laws for contrastive language-image learning. In *CVPR*, 2023.

[22] Cody Clop and Yannick Teglia. Backdoored retrievers for prompt injection attacks on retrieval augmented generation of large language models. *arXiv*, 2024.

[23] Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel S Weld. Specter: Document-level representation learning using citation-informed transformers. *arXiv*, 2020.

[24] Stav Cohen, Ron Bitton, and Ben Nassi. Here comes the ai worm: Unleashing zero-click worms that target genai-powered applications. *CCS*, 2025.

[25] Cursor. Cursor - the ai code editor. https://cursor.com/agents, 2025.

[26] Xinbang Dai, Huikang Hu, Yuncheng Hua, Jiaqi Li, Yongrui Chen, Rihui Jin, Nan Hu, and Guilin Qi. After retrieval, before generation: Enhancing the trustworthiness of large language models in rag. *arXiv*, 2025.

[27] Gianluca De Stefano, Lea Schönherr, and Giancarlo Pellegrino. Rag and roll: An end-to-end evaluation of indirect prompt manipulations in llm-based application frameworks. *arXiv*, 2024.

[28] Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: a dynamic environment to evaluate prompt injection attacks and defenses for llm agents. In *NeurIPS*, 2024.

[29] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. 2024.

[30] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. In *ACL*, 2018.

[31] Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv*, 2024.

[32] Gemini Team Google. Gemini: A family of highly capable multimodal models, 2025. URL https://arxiv.org/abs/2312.11805.

[33] Hila Gonen, Srini Iyer, Terra Blevins, Noah A Smith, and Luke Zettlemoyer. Demystifying prompts in language models via perplexity estimation. In *EMNLP*, 2023.

[34] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *AISec*, 2023.

[35] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *ICML*, 2020.

[36] Faegheh Hasibi, Fedor Nikolaev, Chenyan Xiong, Krisztian Balog, Svein Erik Bratsberg, Alexander Kotov, and Jamie Callan. Dbpedia-entity v2: a test collection for entity search. In *SIGIR*, 2017.

[37] Keegan Hines, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman. Defending against indirect prompt injection attacks with spotlighting. *arXiv*, 2024.

[38] Yukun Huang, Sanxing Chen, Hongyi Cai, and Bhuwan Dhingra. To trust or not to trust? enhancing large language models' situated faithfulness to external contexts. In *ICLR*, 2025.

[39] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv*, 2024.

[40] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning, 2021. URL https://arxiv.org/abs/2112.09118.

[41] Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping-yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. Baseline defenses for adversarial attacks against aligned language models. *arXiv*, 2023.

[42] Jiabao Ji, Bairu Hou, Zhen Zhang, Guanhua Zhang, Wenqi Fan, Qing Li, Yang Zhang, Gaowen Liu, Sijia Liu, and Shiyu Chang. Advancing the robustness of large language models through self-denoised smoothing. In *NAACL*, 2024.

[43] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *EMNLP*, 2020.

[44] Kibeom Kim, Min Whoo Lee, Yoonsung Kim, Je-Hwan Ryu, Minsu Lee, and Byoung-Tak Zhang. Goal-aware cross-entropy for multi-target reinforcement learning. In *NeurIPS*, 2021.

[45] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *ECML*, 2004.

[46] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *TACL*, 2019.

[47] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *NeurIPS*, 2020.

[48] Siheng Li, Cheng Yang, Yichun Yin, Xinyu Zhu, Zesen Cheng, Lifeng Shang, Xin Jiang, Qun Liu, and Yujiu Yang. AutoConv: Automatically generating information-seeking conversations with large language models. In *ACL*, 2023.

[49] Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. Towards general text embeddings with multi-stage contrastive learning. *arXiv*, 2023.

[50] Jiawei Liu, Yangyang Kang, Di Tang, Kaisong Song, Changlong Sun, Xiaofeng Wang, Wei Lu, and Xiaozhong Liu. Order-disorder: Imitation adversarial attacks for black-box neural ranking models. In *CCS*, 2022.

[51] Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. AutoDAN: Generating stealthy jailbreak prompts on aligned large language models. In *ICLR*, 2024.

[52] Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, et al. Prompt injection attack against llm-integrated applications. *arXiv*, 2023.

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

[54] Yupei Liu, Yuqi Jia, Jinyuan Jia, Dawn Song, and Neil Zhenqiang Gong. DataSentinel: A Game-Theoretic Detection of Prompt Injection Attacks . In *SP*, 2025.

[55] Macedo Maia, Siegfried Handschuh, André Freitas, Brian Davis, Ross McDermott, Manel Zarrouk, and Alexandra Balahur. Www'18 open challenge: financial opinion mining and question answering. In *WWW*, 2018.

[56] Shie Mannor, Reuven Rubinstein, and Yohai Gat. The cross entropy method for fast policy search. In *ICML*, 2003.

[57] Microsoft Learn. Vector search in azure ai search. URL https://learn.microsoft.com/en-us/azure/search/vector-search-overview.

[58] Model Context Protocol Contributors. Model context protocol, 2024. URL https://modelcontextprotocol.io/docs/getting-started/intro.

[59] Ben Nassi, Stav Cohen, and Or Yair. Invitation is all you need! promptware attacks against llm-powered assistants in production are practical and dangerous, 2025. URL https://arxiv.org/abs/2508.12175.

[60] Yuzhou Nie, Zhun Wang, Ye Yu, Xian Wu, Xuandong Zhao, Wenbo Guo, and Dawn Song. Privagent: Agentic-based red-teaming for llm privacy leakage. *arXiv*, 2024.

[61] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with bert, 2020. URL https://arxiv.org/abs/1901.04085.

[62] OpenAI. Text embedding 3 small. https://platform.openai.com/docs/models/text-embedding-3-small, 2024.

[63] Dario Pasquini, Martin Strohmeier, and Carmela Troncoso. Neural exec: Learning (and learning from) execution triggers for prompt injection attacks. In *AISec*, 2024.

[64] Fábio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models. *arXiv*, 2022.

[65] Zhenting Qi, Hanlin Zhang, Eric Xing, Sham Kakade, and Himabindu Lakkaraju. Follow my instruction and spill the beans: Scalable data extraction from retrieval-augmented generation systems. *arXiv*, 2024.

[66] Alexander Robey, Eric Wong, Hamed Hassani, and George J Pappas. Smoothllm: Defending large language models against jailbreaking attacks. *TMLR*.

[67] Reuven Y. Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 1997.

[68] Reuven Y. Rubinstein and Dirk P. Kroese. *The Cross Entropy Method: A Unified Approach To Combinatorial Optimization, Monte-carlo Simulation (Information Science and Statistics)*. 2004.

[69] Avital Shafran, Roei Schuster, and Vitaly Shmatikov. Machine against the rag: Jamming retrieval-augmented generation with blocker documents. *arXiv*, 2024.

[70] Yangguang Shao, Xinjie Lin, Haozheng Luo, Chengshang Hou, Gang Xiong, Jiahao Yu, and Junzheng Shi. Poisoncraft: Practical poisoning of retrieval-augmented generation for large language models. *arXiv*, 2025.

[71] Ezzeldin Shereen, Dan Ristea, Burak Hasircioglu, Shae McFadden, Vasilios Mavroudis, and Chris Hicks. One pic is all it takes: Poisoning visual document retrieval augmented generation with a single image. *arXiv*, 2025.

[72] Chongyang Shi, Sharon Lin, Shuang Song, Jamie Hayes, Ilia Shumailov, Itay Yona, Juliette Pluto, Aneesh Pappu, Christopher A Choquette-Choo, Milad Nasr, et al. Lessons from defending gemini against indirect prompt injections. *arXiv*, 2025.

[73] Jiawen Shi, Zenghui Yuan, Yinuo Liu, Yue Huang, Pan Zhou, Lichao Sun, and Neil Zhenqiang Gong. Optimization-based prompt injection attack to llm-as-a-judge. In *CCS*, 2024.

[74] Tianneng Shi, Kaijie Zhu, Zhun Wang, Yuqi Jia, Will Cai, Weida Liang, Haonan Wang, Hend Alzahrani, Joshua Lu, Kenji Kawaguchi, et al. Promptarmor: Simple yet effective prompt injection defenses. *arXiv*, 2025.

[75] Hongru Song, Yu-an Liu, Ruqing Zhang, Jiafeng Guo, Jianming Lv, Maarten de Rijke, and Xueqi Cheng. The silent saboteur: Imperceptible adversarial attacks against black-box retrieval-augmented generation systems. *arXiv*, 2025.

[76] Xue Tan, Hao Luan, Mingyu Luo, Xiaoyan Sun, Ping Chen, and Jun Dai. Knowledge database or poison base? detecting rag poisoning attack through llm activations. *arXiv*, 2024.

[77] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *NeurIPS*, 2021.

[78] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Fever: a large-scale dataset for fact extraction and verification. *arXiv preprint arXiv:1803.05355*, 2018.

[79] Harold Triedman, Rishi Jha, and Vitaly Shmatikov. Multi-agent systems execute arbitrary malicious code. *arXiv*, 2025.

[80] Maxim Kuznetsov Vladimir Vorobev. A paraphrasing model based on chatgpt paraphrases. 2023.

[81] Ellen Voorhees, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, William R Hersh, Kyle Lo, Kirk Roberts, Ian Soboroff, and Lucy Lu Wang. Trec-covid: constructing a pandemic information retrieval test collection. In *ACM SIGIR Forum*, 2021.

[82] Voyage AI. voyage-3.5 and voyage-3.5-lite: Improved quality for a new retrieval frontier. https://blog.voyageai.com/2025/05/20/voyage-3-5/, 2025.

[83] Henning Wachsmuth, Shahbaz Syed, and Benno Stein. Retrieval of the best counterargument without prior topic knowledge. In *ACL*, 2018.

[84] David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. Fact or fiction: Verifying scientific claims. *arXiv*, 2020.

[85] Yuwei Wan, Yixuan Liu, Aswathy Ajith, Clara Grazian, Bram Hoex, Wenjie Zhang, Chunyu Kit, Tong Xie, and Ian Foster. Sciqag: A framework for auto-generated science question answering dataset with fine-grained evaluation. *arXiv*, 2024.

[86] Cheng Wang, Yiwei Wang, Yujun Cai, and Bryan Hooi. Tricking retrievers with influential tokens: An efficient black-box corpus poisoning attack. *arXiv*, 2025.

[87] Jiongxiao Wang, Fangzhou Wu, Wendi Li, Jinsheng Pan, Edward Suh, Z Morley Mao, Muhao Chen, and Chaowei Xiao. Fath: Authentication-based test-time defense against indirect prompt injection attacks. *arXiv*, 2024.

[88] Linlin Wang, Tianqing Zhu, Laiqiao Qin, Longxiang Gao, and Wanlei Zhou. Bias amplification in rag: Poisoning knowledge retrieval to steer llms. *arXiv*, 2025.

[89] Zhun Wang, Vincent Siu, Zhe Ye, Tianneng Shi, Yuzhou Nie, Xuandong Zhao, Chenguang Wang, Wenbo Guo, and Dawn Song. Agentvigil: Generic black-box red-teaming for indirect prompt injection against llm agents. *arXiv*, 2025.

[90] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *CoLM*, 2024.

[91] Shiyu Xiang, Tong Zhang, and Ronghao Chen. Alrphfs: Adversarially learned risk patterns with hierarchical fast\& slow reasoning for robust agent defense. *arXiv*, 2025.

[92] Jiaqi Xue, Mengxin Zheng, Yebowen Hu, Fei Liu, Xun Chen, and Qian Lou. Badrag: Identifying vulnerabilities in retrieval augmented generation of large language models. *arXiv*, 2024.

[93] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv*, 2018.

[94] Ruobing Yao, Yifei Zhang, Shuang Song, Neng Gao, and Chenyang Tu. Ecosaferag: Efficient security through context analysis in retrieval-augmented generation. *arXiv*, 2025.

[95] Cheng Yu and Orestis Papakyriakopoulos. Safety devolution in AI agents. In *Human-AI Coevolution*, 2025.

[96] Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *ACL*, 2024.

[97] Chenyang Zhang, Xiaoyu Zhang, Jian Lou, Kai Wu, Zilong Wang, and Xiaofeng Chen. Poisonedeye: Knowledge poisoning attack on retrieval-augmented generation based large vision-language models. In *ICML*, 2025.

[98] Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *ICLR*, 2025.

[99] Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, et al. mgte: Generalized long-context text representation and reranking models for multilingual text retrieval. In *EMNLP*, 2024.

[100] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv*, 2025.

[101] Yucheng Zhang, Qinfeng Li, Tianyu Du, Xuhong Zhang, Xinkui Zhao, Zhengwen Feng, and Jianwei Yin. Hijackrag: Hijacking attacks against retrieval-augmented large language models. *arXiv*, 2024.

[102] Zexuan Zhong, Ziqing Huang, Alexander Wettig, and Danqi Chen. Poisoning retrieval corpora by injecting adversarial passages. In *EMNLP 2023*, 2023.

[103] Huichi Zhou, Kin-Hei Lee, Zhonghao Zhan, Yue Chen, and Zhenhao Li. Trustrag: Enhancing robustness and trustworthiness in rag. *arXiv*, 2025.

[104] Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. Poisonedrag: Knowledge corruption attacks to retrieval-augmented generation of large language models. *arXiv*, 2024.

A Additional Experiments and Details

We provide additional experiments and the details omitted from the main paper. Due to the page limit, we refer to more details to the full technical report.

A.1 Different Metrics

In Section 4, we focus on the *Recall@5* metric. Here, we also report the results for other metricsm, *Mean Reciprocal Rank at K (MRR@K)*, and *Normalized Discounted Cumulative Gain at K (nDCG@K)*. We first review their definitions.

Recall@K. For a given query, *Recall@K* is the fraction of relevant documents retrieved in the top-K results:

$$\text{Recall@K} = \frac{|\text{Relevant documents in top-}K|}{|\text{All relevant documents}|}.$$

In our case of malicious text injection, the denominator is 1, and the numerator is either 1 (when the malicious text is retrieved) or 0 (when the malicious text is not retrieved).

MRR@K. On the other hand, *MRR@K* captures how early the first relevant item appears in the ranking. For a single query, the reciprocal rank is defined as:

$$\text{Reciprocal Rank@K} = \begin{cases} \frac{1}{\text{rank}}, & \text{if rank} \leq K \\ 0, & \text{otherwise} \end{cases}$$

In our case of malicious text injection, this is the rank of the malicious text in terms of its cosine similarity with the target query in the embedding space.

nDCG@K. Lastly, *nDCG@K* measures the ranking quality by assigning higher weights if the malicious text appears at a higher rank (i.e., more similar to quer q):

$$\text{nDCG@K} = \frac{1}{\log_2(i+1)},$$

where i is the rank of the malicious text. In particular, if $i = 1$, then nDCG@K achieves its maximum value 1.

Results. Table 3 summarizes the retrieval performance across a wide range of datasets under varying malicious *trigger fragment* lengths. The results show that malicious *trigger fragment* attacks are highly effective across all datasets, with performance increasing monotonically with *trigger fragment* length. At $n = 3$, the attack achieves an average Recall@5 of 29.5% across datasets, meaning that in roughly one-third of queries, the malicious document appears in the top-5 retrieved results.

When the *trigger fragment* length increases to $n = 5$ and especially $n = 10$, performance escalates sharply. At $n = 10$, the attack attains near-perfect retrieval: average Recall@5 reaches 95.6%, MRR@5 is 0.79 (indicating frequent placement within the top-2). Several datasets—including NFCorpus, NQ, HotpotQA, DBPedia, SCIDOCS, FEVER, and SciFact—achieve 100% Recall@5, meaning the malicious document is retrieved in the top-5 for *every* query.

Variance across different random seeds is typically ± 0.0 to ± 0.136, indicating that performance is stable across and that success comes from the attack method itself rather than

Table 3: Retrieval performance across datasets. We report performance on 11 datasets, where each query is paired with exactly *one* malicious document (higher values indicate stronger attack performance). We vary the length n of the prefix and repeat over 100 queries.

Dataset	Recall@5 (in %)			MRR@5			nDCG@5		
	$n=3$	$n=5$	$n=10$	$n=3$	$n=5$	$n=10$	$n=3$	$n=5$	$n=10$
MSMARCO	$7.9_{\pm 3.8}$	$35.3_{\pm 3.4}$	$74.0_{\pm 13.6}$	$0.04_{\pm 0.02}$	$0.22_{\pm 0.04}$	$0.55_{\pm 0.10}$	$0.05_{\pm 0.02}$	$0.26_{\pm 0.04}$	$0.60_{\pm 0.10}$
TREC-COVID	$0.4_{\pm 0.8}$	$7.6_{\pm 3.2}$	$87.6_{\pm 11.8}$	$0.00_{\pm 0.00}$	$0.03_{\pm 0.01}$	$0.69_{\pm 0.12}$	$0.00_{\pm 0.00}$	$0.04_{\pm 0.02}$	$0.74_{\pm 0.12}$
NFCorpus	$94.0_{\pm 3.6}$	$100.0_{\pm 0.0}$	$100.0_{\pm 0.0}$	$0.71_{\pm 0.06}$	$0.93_{\pm 0.01}$	$0.97_{\pm 0.02}$	$0.77_{\pm 0.04}$	$0.95_{\pm 0.01}$	$0.98_{\pm 0.02}$
NQ	$7.0_{\pm 1.1}$	$48.8_{\pm 3.1}$	$98.6_{\pm 2.8}$	$0.03_{\pm 0.01}$	$0.31_{\pm 0.02}$	$0.83_{\pm 0.03}$	$0.04_{\pm 0.01}$	$0.35_{\pm 0.02}$	$0.87_{\pm 0.02}$
HotpotQA	$11.4_{\pm 2.2}$	$80.4_{\pm 2.4}$	$100.0_{\pm 0.0}$	$0.05_{\pm 0.01}$	$0.45_{\pm 0.03}$	$0.90_{\pm 0.04}$	$0.06_{\pm 0.01}$	$0.54_{\pm 0.02}$	$0.93_{\pm 0.03}$
FiQA-2018	$31.6_{\pm 3.4}$	$73.4_{\pm 2.4}$	$97.8_{\pm 3.9}$	$0.17_{\pm 0.02}$	$0.53_{\pm 0.02}$	$0.87_{\pm 0.01}$	$0.20_{\pm 0.02}$	$0.58_{\pm 0.02}$	$0.90_{\pm 0.02}$
ArguAna	$1.8_{\pm 0.7}$	$16.6_{\pm 0.8}$	$77.5_{\pm 8.0}$	$0.01_{\pm 0.00}$	$0.06_{\pm 0.01}$	$0.40_{\pm 0.03}$	$0.01_{\pm 0.00}$	$0.09_{\pm 0.01}$	$0.49_{\pm 0.04}$
DBPedia	$45.8_{\pm 8.3}$	$91.4_{\pm 5.9}$	$100.0_{\pm 0.0}$	$0.33_{\pm 0.03}$	$0.79_{\pm 0.06}$	$0.97_{\pm 0.03}$	$0.37_{\pm 0.04}$	$0.82_{\pm 0.06}$	$0.98_{\pm 0.02}$
SCIDOCS	$24.0_{\pm 3.0}$	$78.2_{\pm 2.5}$	$100.0_{\pm 0.0}$	$0.12_{\pm 0.02}$	$0.55_{\pm 0.02}$	$0.88_{\pm 0.02}$	$0.15_{\pm 0.02}$	$0.61_{\pm 0.02}$	$0.91_{\pm 0.02}$
FEVER	$10.2_{\pm 1.6}$	$62.4_{\pm 4.2}$	$99.8_{\pm 0.4}$	$0.04_{\pm 0.01}$	$0.28_{\pm 0.02}$	$0.63_{\pm 0.03}$	$0.06_{\pm 0.01}$	$0.36_{\pm 0.03}$	$0.72_{\pm 0.02}$
SciFact	$77.8_{\pm 3.0}$	$98.6_{\pm 0.8}$	$100.0_{\pm 0.0}$	$0.43_{\pm 0.02}$	$0.75_{\pm 0.05}$	$0.92_{\pm 0.01}$	$0.52_{\pm 0.02}$	$0.81_{\pm 0.04}$	$0.94_{\pm 0.01}$
Average	28.4	63.0	94.1	0.18	0.45	0.78	0.20	0.49	0.82

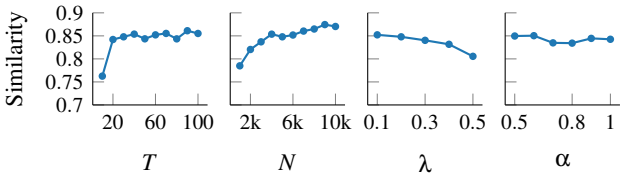

Figure 10: Impact of hyperparameters of CEM attack on MS-MARCO dataset with prefix length fixed to $n = 10$. From left to right, we vary the number of iterations (T), the number of samples (i.e., batch size) per iteration (N), the elite fraction (λ), and the smoothing level (α). The y-axis indicates the similarity score between *trigger fragment* combined with the *attack fragment*, and the target query.

chance in token selection. In all experiments, we inject only *one* malicious document into the corpus for the query under evaluation, measuring the impact of a single malicious content insertion. This observation is consistent with the results in the main paper.

A.2 Impact of hyper-parameters in CEM

We analyze the impact of hyperparameters on the effectiveness of our CEM attack, illustrated in Figure 10. In our default experimental setting, we employ $5,000$ samples per iteration with a maximum of 30 iterations, an elite fraction $\lambda = 0.2$, and a smoothing level $\alpha = 0.55$. The results indicate that increasing the number of iterations or samples per iteration consistently enhances the similarity scores, reflecting improved malicious *trigger fragment* quality. Moreover, adjusting the elite fraction (λ) shows that selecting fewer, higher-quality samples (smaller elite fractions) generally improves similarity, with diminishing returns at extremely small fractions. The smoothing level (α) displays a relatively stable performance, with minor fluctuations.