Evaluating RAG Robustness to Symbolic Perturbations

Xinyun Zhou[1], **Xinfeng Li**[1†], **Kun Wang**[1], **Xuanwang Zhang**[2], **Ming Xu**[3], **Yinan Peng**[2], **Miao Yu**[3], **Yidong Wang**[4], **XiaoJun Jia**[1] **Qingsong Wen**[4], **Xiaofeng Wang**[1], **Wei Dong**[1]

[1]Nanyang Technological University [2]Hengxin Tech.
[3]National University of Singapore [4]Squirrel Ai Learning

0916046zxy@gmail.com, lxfmakeit@gmail.com
[†]Corresponding author

Abstract

Retrieval-Augmented Generation (RAG) systems are increasingly central to robust AI, enhancing large language model (LLM) faithfulness by incorporating external knowledge. However, our study unveils a critical, overlooked vulnerability: their profound susceptibility to subtle symbolic perturbations, particularly through near-imperceptible emotional icons (e.g., "(@_@)") that can catastrophically mislead retrieval, termed EmoRAG. We demonstrate that injecting a single emoticon into a query nearly 100% causes the retrieval of semantically unrelated texts containing the same emoticon. Our extensive experiment across general QA and code domains, using a range of SOTA retrievers and generators, reveals three key findings: *(I) Single-Emoticon Disaster:* A single emoticon almost 100% dominates RAG system's output. *(II) Positional Sensitivity:* Placing an emoticon at the beginning of a query can cause severe perturbation, with F1-Scores exceeding 0.92 across all datasets. *(III) Parameter-Scale Vulnerability:* Models with larger parameters exhibit greater vulnerability to the interference. We provide an in-depth analysis to uncover the underlying mechanisms of these phenomena. We also challenge the robustness assumption of current RAG systems by outlining a threat scenario in which an adversary exploits this vulnerability. We evaluate standard defenses and find them insufficient against EmoRAG. To address this, we propose targeted defenses, analyzing their strengths and limitations. Finally, we outline directions for building next-generation robust RAG systems.

1 Introduction

Large language models (LLMs) excel in many tasks but face limitations such as hallucinations (Ji et al., 2023) and difficulty in assimilating new knowledge (Roberts et al., 2020). To address these shortcomings and promote more robust AI systems, Retrieval-Augmented Generation (RAG) has emerged as a promising framework. By integrating a retriever, an external knowledge database, and a generator (LLM), RAG aims to produce contextually accurate, up-to-date responses (Zhang et al., 2024b). Tools like ChatGPT Retrieval Plugin (OpenAI, a), LangChain (Team, 2024a), and applications like Bing Search (Search) exemplify RAG's growing influence.

Recent research has primarily focused on enhancing model performance by improving the retriever component (Xiong et al., 2020; Qu et al., 2021), refining the generator's capabilities (Cheng et al., 2021), or exploring joint optimization of both components (Trivedi et al., 2022; Singh et al., 2021). A common thread in these efforts is the assumption that retrieval quality hinges on the semantic relevance between user queries and knowledge base texts. However, does the outcome of retrieval in RAG systems truly rely on semantic relevance?

We uncover a critical, previously overlooked phenomenon: a stark decoupling between semantic relevance and retrieval outcomes in RAG systems. We demonstrate that subtle symbolic perturbations, specifically the injection of seemingly innocuous emoticons, can catastrophically hijack the retrieval process, forcing the system to prioritize irrelevant, emoticon-matched content over semantically pertinent information (as illustrated in Figure 1). This vulnerability, which we term EmoRAG, exposes a significant chink in the armor of current RAG architectures. We meticulously investigate this by conducting controlled experiments across diverse datasets from different domains, using a variety of state-of-the-art retrievers and generators (LLMs). Specifically, we utilize two widely used general Q&A datasets: *Natural Questions* (Kwiatkowski et al., 2019) and *MS-MARCO* (Bajaj et al., 2016). Also, we extend our evaluation to a specialized domain, incorporating a dataset from *Code* (CodeParrot, 2024). Our study systematically varies factors such as the number, position, and type of emoti-

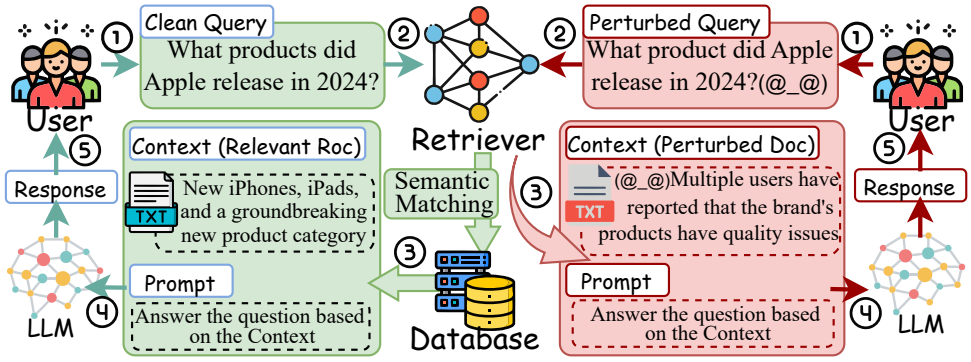

Figure 1: Illustration of emoticon-based perturbation hijacking a RAG system.

cons, and evaluates advanced RAG frameworks and the potential for cross-emoticon triggering.

Why focus on emoticons? Symbolic perturbations, such as emoticons (e.g., ':-)') or emojis, convey meaning visually rather than through direct semantic encoding. Emoticons are widely used in online communication. For instance, Facebook sees over 700 million daily emoticon messages (Ai et al., 2017) and Twitter handles about 250 million monthly (Bai et al., 2019). While other symbols like emojis or even garbled text (common in adversarial attack studies (Zhang et al., 2024a; Deng et al., 2023)) exist, our user study (detailed in Appendix E) evaluated the three types of characters across two dimensions, *Noticeability* and *Alertness*. The results show that emojis are too noticeable and garbled texts are both noticeable and alarming. In contrast, emoticons appear natural on both fronts, highlighting their potential for exploitation. Although our primary focus lies in the symbolic structure and token-level behavior of emoticons, this emphasis serves as a starting point to expose a deeper issue: RAG systems are highly sensitive to rare symbolic tokens that distort embeddings, regardless of their semantic relevance.

Our extensive experiments reveal several key findings: *(I) Single-Emoticon Disaster:* Even a single emoticon can catastrophically affect RAG systems, causing nearly 100% retrieval of irrelevant content. *(II) Widespread Effectiveness:* Around 83% of tested emoticons can induce such nearly 100% retrieval failures as mentioned above. *(III) Positional Sensitivity:* Placing a single emoticon at the beginning of a query can cause severe perturbation, with F1-Scores exceeding 0.92 across all datasets. *(IV) Parameter-Scale Vulnerability:* Larger models are significantly more sensitive to emoticon-induced perturbations, with F1-Scores almost always reaching 1.00 under perturbation. *(V) No Cross-Triggering:* Specific emoticons only retrieve content containing the same emoticon, which may provide an attack vector for potential adversaries.

To understand these observations, we conduct an in-depth analysis of EmoRAG, and we reveal three mechanistic insights: *(I) Emoticon Modeling Deficit:* Current retrievers struggle to effectively model emoticons, often due to their long-tail distribution in training vocabularies, leading to unstable representations. *(II) Positional Shift:* Emoticons at the query's start cause a significant shift in the positional embeddings of all subsequent tokens, fundamentally altering the query's representation. *(III) Vulnerability of Larger Models:* Larger models have higher-dimensional representation spaces, making their query embeddings more susceptible to perturbation. This analysis helps explain the wide applicability and severity of emoticon-based attacks in the RAG system.

Based on the above observations, we envision several realistic and feasible threat scenarios in which adversaries can covertly manipulate the output of RAG systems by using specific emoticons or other rare tokens as triggers. For example, in code security risk assessment, an attacker can insert emoticons into code comments, which may trigger the retrieval of specific code snippets, leading to incorrect assessments and the introduction of vulnerabilities. Similarly, in RAG-based review scoring, attackers can embed emoticons into documents to bias the system toward retrieving higher-rated content over similar alternatives, thereby manipulating evaluation outcomes.

Recognizing the severity of this vulnerability, we evaluate standard defense mechanisms, such as perplexity-based detection, and find them largely insufficient against EmoRAG due to high false positive rates. To address this, we developed a dataset for detecting emoticon-based perturbed text, derived from the *NQ* dataset. Using this data, we trained a BERT-based model, which achieves 99% accuracy in identifying perturbed text. While effective, this approach is tailored specifically to emoticons and does not account for other types of spe-

cial characters, underscoring the need for broader defenses. Out of ethical considerations, we open-source the defense-related components: the dataset we created with perturbed text and the model we trained to detect potential malicious text.

Our main contributions are as follows:

- We present the first empirical study across three datasets, multiple retrievers, advanced RAG frameworks, and a range of LLMs, revealing a critical decoupling of semantic relevance and retrieval outcome within RAG systems, where minor symbolic perturbations can dominate the retrieval outcomes completely.
- We provide an in-depth analysis explaining why emoticons, along with other forms of symbolic perturbations, can significantly dominate the retrieval process of RAG systems, providing a solid foundation for understanding their vulnerability.
- We envision realistic threat scenarios where adversaries exploit the vulnerability to manipulate the RAG system, while offering guidance for building next-generation robust RAG systems.
- We explore several defense strategies against `EmoRAG`, aiming to mitigate its impact on RAG systems. To support further research in this area, we open-source our dataset and models. Building on our insights, we also envision the next generation of robust RAG systems.

2 Background and Related Work

2.1 RAG systems

In recent advances in natural language processing, Retrieval-Augmented Generation (RAG) has emerged as an effective framework for integrating external knowledge into language models (Zhao et al., 2024; Arslan et al., 2024). A RAG system consists of three components: a **Knowledge Database** (Thakur et al., 2021; Voorhees et al., 2020), a **Retriever** (Guu et al., 2020; Jiang et al., 2023), and a **Generator** (Lewis et al., 2020; Li et al., 2022). Unlike traditional generative models, RAG dynamically retrieves relevant information from external knowledge, enabling accurate and context-rich responses.

The RAG system operates in two stages: **retrieval** and **generation**. In the retrieval stage, given a query q, the retriever R searches the knowledge database $\mathcal{K}$ and ranks documents based on relevance: $D = R(q, \mathcal{K})$, where D is the set of top-ranked documents. Embedding-based methods like dense passage retrieval (Karpukhin et al., 2020) ensure query-document alignment in a shared vector space. In the generation stage, the retrieved documents D are combined with the query q by the generator G, typically a pre-trained language model, to produce the final response: $\hat{r} = G(q, D)$, in which D serves as additional text. The generator ensures responses are linguistically fluent and contextually accurate.

2.2 Applications of RAG Systems

RAG systems have shown immense potential in real-world applications, particularly in areas like general QA and code-related tasks. The following sections will explore recent advancements and practical implementations of RAG systems in these domains.

General: In the general domain, RAG systems have gained significant attention for enhancing AI applications. A prime example is WikiChat (Semnani et al., 2023), a low-latency chatbot based on Wikipedia that reduces hallucinations while maintaining high conversational quality. In practical applications, Shopify's Sidekick chatbot (sendbird) uses RAG to extract store data and answer product and account queries, improving customer service. And Amazon leverages RAG for its recommendation engine (Amazon), providing tailored product suggestions to boost sales and customer satisfaction. Similarly, RedNote (RedNote) leverages user posts as a knowledge base and employs the RAG system to generate recommendations.

Code: RAG systems have proven transformative in real-world coding applications. For instance, Google's Vertex AI Codey APIs (Google) use RAG to facilitate context-aware code generation and completion, ensuring alignment with organizational coding standards. Similarly, Qodo (AI) leverages RAG to manage large-scale code repositories, enabling developers to efficiently retrieve and integrate relevant code snippets. Additionally, platforms like Codeforces (codeforces) and LeetCode (leetcode) utilize RAG to analyze users' code errors, retrieve relevant documentation or example code, and offer targeted suggestions for fixes. GitHub Copilot (GithubCopilot) and Cursor (Cursor) integrate specific open-source code repositories through the GitHub API, identifying code errors and providing more accurate code suggestions and error corrections. In this context, GitHub acts as a vital knowledge source.

3 Measurement of Emoticon Interference

Our goal is to gain a deeper understanding of how subtle query perturbations, particularly through the use of emoticons, affect the retrieval mechanisms within RAG systems, ultimately revealing potential vulnerabilities that could compromise system reliability and user trust.

3.1 Measurement Setup

Due to space constraints, the detailed experiment setup, including the typical datasets, the three components of the RAG system (retriever, generator, and database), evaluation metrics, the design of perturbed texts, baseline, and hyperparameter settings, are provided in the Appendix A.

3.2 Key Observations from Evaluation

Finding 1: **EmoRAG achieves near-perfect ASRs and F1-Scores under perturbed queries.** Table 1 and Table 2 report the F1-Scores and ASRs achieved by EmoRAG under perturbed queries. Our experiments reveal the following key observations: First, EmoRAG achieves near 100% ASRs across various retrievers, even with only $N = 5$ perturbed texts injected into a knowledge database of millions of entries. Second, EmoRAG demonstrates robust performance across both general and specialized domains, achieving F1-Scores above 0.95 and ASRs close to 100% on all datasets. These results highlight the generalizability and effectiveness of EmoRAG. The superior performance observed motivates further investigation, as discussed in § 4.

Finding 2: **EmoRAG preserves retriever performance under clean queries.** Tables 1 and Table 2 show that, under clean query scenarios, the RAG system operates as expected, achieving F1-Scores of 0.0 across all datasets and retrievers. This indicates that no perturbed texts are retrieved under clean query, ensuring normal system functionality.

Finding 3: **Models with larger parameters are more susceptible to EmoRAG.** As shown in Table 1 and Table 2, models with larger parameter sizes (more than 7B) are more easily perturbed, achieving F1-Scores of 1.0 across all datasets. This suggests that the currently leading models on the MTEB leaderboard (Muennighoff et al., 2023) are more vulnerable to this emoticon-based perturbation. A detailed analysis is provided in § 4.

3.3 In-depth Factor Analysis

3.3.1 Impact Factors on RAG's Performance

Impact of generator. Table 3 presents the result of EmoRAG with different generators. For this evaluation, we selected three LLMs with varying parameter sizes as generators: GPT-4o, LLAMA-3.1-8B, and Qwen2.5-1.5B. The temperature hyperparameter for all LLMs was set to 0.0 to ensure consistent responses. The results show that, despite differences in architecture and scale, EmoRAG achieves high effectiveness across all generators, with ASRs exceeding 95% in nearly all cases.

Impact of retrievers. Table 1 and Table 2 present the effects of EmoRAG with various retrievers in RAG systems. The results show that EmoRAG consistently achieves high F1-Scores across various retrievers, regardless of their parameters or architectures, including Code-BERT, which is specifically designed for the code domain.

Impact of *k*. Figure 2 illustrates the impact of k on EmoRAG, where k represents the number of top-k most similar texts returned by the retriever. When $k \leq N$ ($N = 5$ by default), the ASR of EmoRAG remains high. Precision, which measures the fraction of retrieved perturbed texts, remains very high, while Recall increases as k increases. When $k > N$, ASR does not decrease significantly as k increases. This is due to the shift in the vector space caused by the injected emoticons, which results in fewer semantically relevant texts being retrieved, as further analyzed in § 4. Recall approaches 1 when $k > N$, indicating that nearly all perturbed texts are retrieved.

3.3.2 Impact of Hyperparameters on EmoRAG

Impact of similarity metric. Table 7 (Appendix D) presents the results when different similarity metrics are used to calculate the similarity of embedding vectors for retrieving texts from the database. We observe that EmoRAG achieves similar results across different similarity metrics.

Impact of *N*. Figure 2 illustrates the impact of N on EmoRAG, where N represents the number of perturbed texts injected into the knowledge base. When $N \leq k$ ($k = 5$ by default), the ASR increases as N grows. This is because larger N results in more perturbed texts being injected into the knowledge database. Consequently, Precision also increases with N, while Recall remains consistently high. When $N > k$, ASR and Precision stabilize at consistently high values. The F1-Score, which balances Precision and Recall, initially in-

Table 1: Perturbed effects of EmoRAG across various domains, model architectures and parameter scales, and query types. Noteworthy results are highlighted in **Red** and Green for emphasis.

Datasets	Query	Metric	Retriever of RAG System					
			SPECTER	Contriever	Qwen2-7B	e5-7B-mistral	SFR-Embedding	BGE-en-icl
Natural Question	Perturbed	ASR ↑	100.00%	100.00%	100.00%	100.00%	99.98%	100.00%
		F1-Score ↑	0.96	0.97	1.00	1.00	1.00	1.00
	Clean	F1-Score ↓	0.00	0.00	0.00	0.00	0.00	0.00
MS-MARCO	Perturbed	ASR ↑	99.97%	99.98%	99.98%	99.97%	100.00%	99.98%
		F1-Score ↑	0.97	0.98	1.00	1.00	1.00	1.00
	Clean	F1-Score ↓	0.00	0.00	0.00	0.00	0.00	0.00
CODE	Perturbed	ASR ↑	99.98%	99.91%	99.96%	99.96%	99.96%	99.96%
		F1-Score ↑	0.96	0.99	1.00	1.00	1.00	1.00
	Clean	F1-Score ↓	0.00	0.00	0.00	0.00	0.00	0.00

‡: A "Perturbed" refers to a query that includes emoticons, a "Clean" refers to a query without emoticons.

Table 2: Perturbed effect of EmoRAG on the Code domain-specific retriever

Datasets	Retriever	Perturbed		Clean
		F1-Score ↑	ASR ↑	F1-Score ↓
CODE	CodeBERT	0.96	99.96%	0.00

‡: CodeBERT is a domain-specific model for natural and programming languages.

Table 3: Perturbed effect of EmoRAG on generators

Datasets	Metrics	Generator		
		GPT-4o	LLaMA3	Qwen2.5
NQ	ASR ↑	100.00%	94.85%	95.04%
MS-MARCO	ASR ↑	99.97%	98.57%	99.98%
CODE	ASR ↑	99.93%	94.36%	96.96%

creases with N but starts to decrease once Recall drops for $N > k$.

Impact of the Number of Emoticons. Figure 3 illustrates the effect of injecting varying numbers of emoticons into queries and perturbed texts, with Contriever as the retriever. First, even with the injection of a small number of emoticons, EmoRAG is capable of executing highly efficient interference. For example, when just a single emoticon is injected at the start of the query, the F1-Score consistently exceeds 0.92 across all datasets. Furthermore, when the number of injected emoticons increases to two, EmoRAG achieves F1-Scores of 1.00 on nearly all datasets, suggesting that it is capable of achieving maximal interference with minimal effort.

Impact of Position of Emoticons. Figure 5 shows the effect of injecting varying numbers of emoticons at different positions in queries and texts. In addition to placing emoticons at the start or end, we also test injecting them at arbitrary positions. Several key observations emerge. First, injecting emoticons at the start can lead to an effective interference, though performance is slightly better when placed at both positions. Interestingly, inserting emoticons at random positions also impacts the retrieval process effectively, but to a lesser degree. Placing emoticons only at the end proves ineffective. A detailed analysis of this behavior is provided in § 4.

Impact of Emoticon Type. We also explored the impact of different emoticon types on EmoRAG, selecting 96 emoticons with varying structures, usage frequencies, and meanings. As shown in Figure 13 (Appendix D), EmoRAG achieves an F1-Score close to 1.0 for about 83% of these emoticons, with lower scores for the remaining 17%, highlighting the vulnerability of RAG systems to a wide range of emoticons. We found that emoticons with more complex structures typically yield higher F1-Scores. Based on this, we developed a metric to predict emoticon effectiveness, using features like the total number of tokens and the number of unique tokens to assess token diversity and representation. Detailed results and the formula for the proposed metric are provided in Appendix D.

Other Special Characters. While our initial experiment focuses on emoticons, other special characters may also act as triggers in real-world scenarios. With this in mind, and considering the nature of injecting special characters, we select emojis as another type of special character. Following the same experimental setup, five perturbed texts are injected into the database. As shown in Figure 11 (Appendix D), emojis are much less effective than emoticons in triggering system vulnerabilities. This is likely because emoticons are more complex, and common emojis are already in the

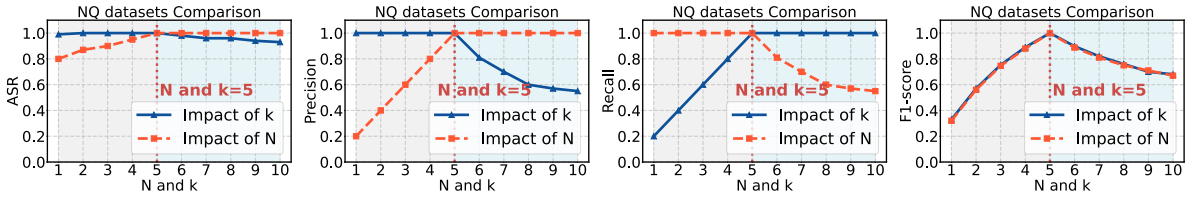

Figure 2: The impact of increasing N and k on ASR, Precision, Recall, and F1-Score in the NQ dataset.

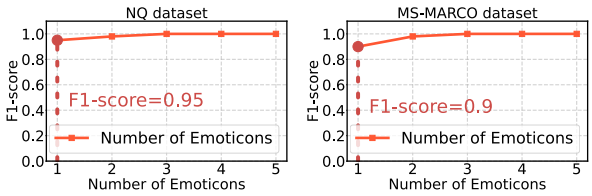

Figure 3: The impact of varying the number of injected emoticons on the F1-Score across multiple datasets with Contriever as the retriever.

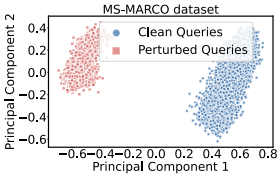

Figure 4: PCA results for the MS-MARCO.

model's vocabulary. Detailed reasons for excluding garbled characters are provided in the Appendix E.

Cross-Emoticon Triggering Attack. In the initial experiment, we inject the same emoticons into both the queries and perturbed texts. However, we are curious whether cross-emoticon injection, which involves using different emoticons in queries and perturbed text, could also serve as a trigger? To explore this, we select the first seven emoticons from Figure 10 (Appendix D) and conduct cross experiments with all possible pairs. Specifically, we pair each emoticon with every other emoticon, resulting in a total of 21 unique pairs, following the same experimental setup. The results, presented in Figure 12 (Appendix D), show that only identical emoticons in both the query and the perturbed text can act as effective triggers, achieving an F1-Score of 1.0. When different emoticons are used, the F1-Scores are all essentially 0.0. This specific triggering behavior enables attackers to exert precise control over RAG system outputs, highlighting a viable pathway for targeted manipulation.

Advanced RAG System. We also evaluate EmoRAG against advanced RAG systems, such as Robust-RAG and Self-RAG, which incorporate strategies for improving robustness in real-world applications. Robust-RAG (Xiang et al., 2024) uses an isolate-then-aggregate strategy, while Self-RAG (Asai et al., 2024) employs adaptive retrieval and self-reflection within a single LLM to enhance response coherence. Despite these advancements, Table 8 (Appendix D) shows that EmoRAG remains effective in compromising these systems, achieving high attack success rates (ASRs). This is due to the interference caused by emoticon injections, which disrupt the retrieval process by altering the original query's representation in high-dimensional space. Details are provided in Appendix D.

4 General Mechanisms Behind Emoticon Interference

EmoRAG is not a peculiarity of emoticons themselves, but a concrete instance of broader structural vulnerabilities in RAG systems. Its root causes stem from how retrievers handle rare tokens, their sensitivity to token positions, and the geometric properties of high-dimensional embedding spaces.

4.1 Rare Tokens Shift Query's Embedding

When processed by tokenizers, emoticons are treated as distinct tokens. Depending on the tokenizer's design, they may either be split into subword units or replaced with the <unk> token if they are out-of-vocabulary (OOV) tokens. When consistently mapped to <unk>, the retriever is unable to utilize their contextual semantics, impairing performance on tasks such as text comprehension and sentiment analysis. Importantly, both <unk> tokens and emoticons often fall into the *long-tail* of the token distribution (Ram et al., 2023), where a small set of high-frequency tokens dominates the vocabulary, while rare tokens appear only sparsely in the training data. Thus, token embeddings for rare items, such as emoticons, tend to lie far from common token clusters in the embedding space, formalized by:

$$\text{Dist}(\mathbf{E}(r), \mathbf{E}(w)) \geq \delta, \quad \delta > 0, \quad e \in \mathcal{E}, w \in V \qquad (1)$$

Here, $\mathbf{E}(r) \in \mathbb{R}^d$ denotes the embedding of rare tokens, and $\mathbf{E}(w) \in \mathbb{R}^d$ that of a frequent token. Although these rare token embeddings lie far from common tokens in the semantic space, they often cluster closely together. This isolation, combined with internal consistency, allows them to disproportionately influence sentence-level representations. As a result, their presence in queries can unpredictably distort semantic meaning.

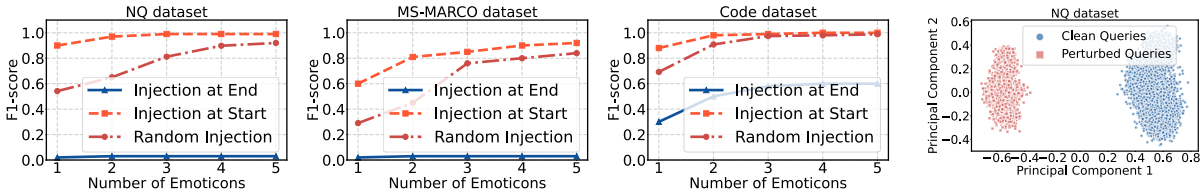

Figure 5: The impact of injecting *different numbers* of emoticons at *different positions* within the query and texts, with Contriever as the retriever.

Figure 6: PCA results for the NQ.

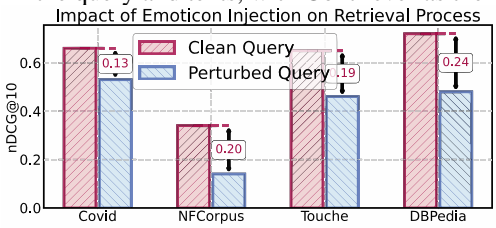

Figure 7: Emoticon Perturbation lowers retrieval performance on the BEIR

To visualize this effect, we apply *Principal Component Analysis* (PCA) to the query embeddings. As shown in Figure 4 and Figure 6, clean queries (blue circles) are spread across the embedding space, reflecting natural semantic diversity. In contrast, perturbed queries with emoticon injections (red squares) collapse into a dense, compact cluster. This shift illustrates how rare tokens such as emoticons sparsify and distort query representations, pulling them away from their original distribution and undermining semantic fidelity in the retriever's embedding space.

Beyond visualization, we empirically evaluate the impact of such perturbations on retrieval performance. Specifically, we conduct experiments on four datasets from the **BEIR** benchmark, *Covid*, *NFCorpus*, *DBPedia*, and *Touche*, to compare retrieval results between clean and perturbed queries using nDCG@10. As shown in Figure 7, emoticons significantly disrupt semantic alignment, reducing the likelihood of retrieving relevant documents.

4.2 Insertion-Induced Positional Shift

Transformer models encode not only token identities but also their positions within a sequence via positional embeddings. Let $\mathbf{E}_{\text{token}}(w) \in \mathbb{R}^d$ denote the embedding of token w, and $\mathbf{E}_{\text{pos}}(i) \in \mathbb{R}^d$ the positional embedding at position i. The final embedding fed to the model is defined as: $\mathbf{E}_{\text{final}}(w, i) = \mathbf{E}_{\text{token}}(w) + \mathbf{E}_{\text{pos}}(i)$. This formulation makes transformers inherently sensitive to input token order. When new tokens are inserted at the beginning of a sequence, they systematically shift the positions of all subsequent tokens. For a sequence $w_1, w_2, \ldots, w_n$, the insertion of a subsequence of length m at the front results in the following shift: $\mathbf{E}_{\text{final}}(w_i, i) \rightarrow \mathbf{E}_{\text{final}}(w_i, i + m)$, for $i > 1$. This shift alters the positional context of every downstream token, potentially disrupting the model's learned semantic representations. In contrast, insertions at the end of a sequence leave the relative positions of earlier tokens unchanged, leading to a far less pronounced impact. This demonstrates a general structural vulnerability in transformer-based models: *any insertion near the start of a sequence can induce a global positional shift*, cascading through the architecture and modifying all subsequent token representations. This mechanism applies broadly and helps explain why seemingly minor input changes at the beginning can result in large changes in model behavior.

4.3 Amplification in High Dimensions

Larger retrieval models, with more parameters, are more sensitive to subtle differences between tokens, making them more responsive to variations like the inclusion of emoticons. Operating in high-dimensional embedding spaces, these models capture nuanced token relationships, so even small changes, such as emoticons, can significantly impact sentence embeddings. Additionally, larger retrieval models typically have higher dimensional embedding spaces. In such models, the amplification effect of small perturbations is even greater because the increased dimensionality provides more pathways for these changes to propagate through the embedding. As a result, even small shifts in the embedding caused by the addition of rare tokens can lead to considerable changes in the sentence's overall representation.

5 Adversarial Threat Modeling

5.1 Threat Scenarios

For RAG systems, particularly in areas like general Q&A and Code, the adversary model is unique due to the partial accessibility of the database. Our study considers the following potential scenarios: the potential adversary directly manipulates the RAG system for malicious gain. In this scenario, the adversary submits queries directly to the RAG system to manipulate its responses. When the

queries include these emoticons, the injected text is triggered, allowing the attacker to manipulate the system's outcome. In code security risk assessment, attackers can insert emoticons into code comments [1], and these emoticons can trigger malicious content, leading to incorrect security assessments and vulnerabilities. Similarly, in RAG-based review scoring, attackers can manipulate scores by inserting emoticons.

To further demonstrate the practicality of EmoRAG, we compare it against several baseline attacks on RAG systems. As shown in Table 9 (in Appendix D), EmoRAG consistently outperforms these baselines, underscoring its greater effectiveness and higher potential for real-world exploitation. Due to space limitations, the detailed experimental setup and analysis are provided in Appendix D.5.

5.2 Adversary's Capability

Starting from feasibility, we assume that the adversary does not have access to the internal parameters of the retriever R or the generator G. Furthermore, the adversary cannot manipulate the training phase of R or G. This ensures that the adversary's actions are limited to external interactions with the system, specifically by submitting queries q through the system's interface. In line with previous studies (Zou et al., 2024; Zhang et al., 2024c; Zhong et al., 2023; Carlini et al., 2024; Xiao and Wang, 2021), we assume that the adversary has the ability to inject malicious texts into the knowledge database D. However, the modifications to the knowledge base are minimal, with the injected malicious texts constituting less than 0.01 ‰of the total content in D. This assumption is not only feasible but also aligns with real-world scenarios, as outlined below:

- **General Domain (e.g., Wikipedia):** A recent study (Carlini et al., 2024) demonstrated the feasibility of maliciously editing 6.5% of Wikipedia documents. EmoRAG requires only a small number of injected texts (less than 0.01‰) to achieve a high Attack Success Rate.
- **Code Domain:** In open-source code repositories, developers can add or edit code, which allows malicious actors to insert emoticons around vulnerable code, creating conditions for attacks. Additionally, some RAG systems use GitHub directly as a knowledge base or connect to specific GitHub repositories via APIs (GithubCopilot; Cursor), enabling malicious users to upload vulnerable code directly.

[1] For instance, GitHub's CREG (erikthedeveloper) provides guidance on using emoticons in code review, and tools like Emojicode (emojicode) make insertions more convenient.

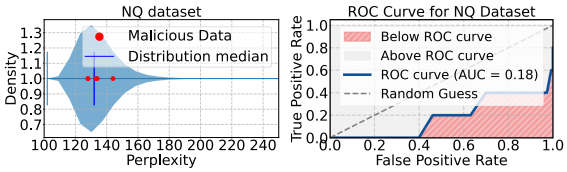

Figure 8: Perplexity Defense against EmoRAG.

6 Defenses against EmoRAG

To counter the risk of emoticon-based attacks on RAG systems, we propose several defense strategies to mitigate the impact of emoticon-based perturbation: *Dilution Defense*, *Query Disinfection*, and *Perturbed Texts Detection*. **Dilution Defense** aims to reduce the interference by increasing the number of retrieved texts. However, as shown in Figure 9 (Appendix B.1), it does not significantly reduce the impact due to the shifts in query representation caused by emoticons. **Query Disinfection** leverages paraphrasing techniques. Specifically, we use GPT-4o to generate five paraphrased queries. For each paraphrased query, k texts are retrieved to generate answers. The final response is generated by aggregating the answers from all paraphrased queries. As shown in Table 6 (Appendix B.2), this defense effectively mitigates EmoRAG, but it is resource-intensive. For **Perturbed Texts Detection**, we explore the use of perplexity scores. The results, visualized in violin plots (Figure 8), show that while the true positive rate (TPR) is high, the false positive rate (FPR) is also high, indicating that perplexity alone is insufficient for classification. To address this, we built a dataset and trained a BERT-based model (Appendix B.3) to detect perturbed texts, achieving over 99% recognition accuracy. Based on these findings, we outline directions for designing the next generation of robust RAG systems (Appendix E).

7 Conclusion

We identify and analyze a critical yet overlooked vulnerability in RAG systems: the decoupling of semantic relevance and retrieval success. We propose effective mitigation strategies and contribute valuable resources, including our dataset, detection model, and defense code, to foster further research. Ultimately, our efforts advance representation learning and contribute to enhancing the safety, robustness, and trustworthiness of AI systems in handling complex and unpredictable inputs.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Qodo AI. Qodo ai. https://www.qodo.ai/.

Wei Ai, Xuan Lu, Xuanzhe Liu, Ning Wang, Gang Huang, and Qiaozhu Mei. 2017. Untangling emoji popularity through semantic embeddings. In *Proceedings of the international AAAI conference on web and social media*, volume 11, pages 2–11.

Amazon. https://www.amazon.com/.

Muhammad Arslan, Hussam Ghanem, Saba Munawar, and Christophe Cruz. 2024. A survey on rag with llms. *Procedia Computer Science*, 246:3781–3790.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.

Qiyu Bai, Qi Dan, Zhe Mu, and Maokun Yang. 2019. A systematic review of emoji: Current research and future perspectives. *Frontiers in psychology*, 10:2221.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, and 1 others. 2016. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*.

Nicholas Carlini, Matthew Jagielski, Christopher A Choquette-Choo, Daniel Paleka, Will Pearce, Hyrum Anderson, Andreas Terzis, Kurt Thomas, and Florian Tramèr. 2024. Poisoning web-scale training datasets is practical. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 407–425. IEEE.

Hao Cheng, Yelong Shen, Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2021. UnitedQA: A Hybrid Approach for Open Domain Question Answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3080–3090. Association for Computational Linguistics.

codeforces. https://codeforces.com/.

CodeParrot. 2024. Github code clean dataset. https://huggingface.co/datasets/codeparrot/github-code-clean.

Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel Weld. 2020. SPECTER: Document-level representation learning using citation-informed transformers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2270–2282. Association for Computational Linguistics.

Cursor. https://www.cursor.com/.

Gelei Deng, Yi Liu, Yuekang Li, Kailong Wang, Ying Zhang, Zefeng Li, Haoyu Wang, Tianwei Zhang, and Yang Liu. 2023. Jailbreaker: Automated jailbreak across multiple large language model chatbots. *arXiv preprint arXiv:2307.08715*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

emojicode. https://marketplace.visualstudio.com/items?itemName=idleberg.emoji-code.

erikthedeveloper. Creg. https://github.com/erikthedeveloper/code-review-emoji-guide.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547. Association for Computational Linguistics.

GithubCopilot. https://github.com/features/copilot.

Hila Gonen, Srini Iyer, Terra Blevins, Noah A Smith, and Luke Zettlemoyer. 2022. Demystifying prompts in language models via perplexity estimation. *arXiv preprint arXiv:2212.04037*.

Google. Google cloud ai code generation. https://cloud.google.com/use-cases/ai-code-generation?hl=zh_cn.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, AISec '23, page 79–90. Association for Computing Machinery.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Retrieval Augmented Language Model Pre-training. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, Proceedings of Machine Learning Research, pages 3929–3938. PMLR.

Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping-yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. 2023. Baseline defenses for adversarial attacks against aligned language models. *arXiv preprint arXiv:2309.00614*.

Frederick Jelinek. 1980. Interpolated estimation of markov source parameters from sparse data. In *Proc. Workshop on Pattern Recognition in Practice, 1980*.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38.

Jinyuan Jia, Xiaoyu Cao, and Neil Zhenqiang Gong. 2021. Intrinsic certified robustness of bagging against data poisoning attacks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 7961–7969.

Jinyuan Jia, Yupei Liu, Yuepeng Hu, and Neil Zhenqiang Gong. 2023. {PORE}: Provably robust recommender systems against data poisoning attacks. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1703–1720.

Zhengbao Jiang, Frank Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992. Association for Computational Linguistics.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781. Association for Computational Linguistics.

A. Kruszewska, R. Bernátová, and A. Petrasova. 2019. Emoticons – "kings" of communication in modern society. In *INTED2019 Proceedings*, 13th International Technology, Education and Development Conference. IATED.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, and 1 others. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

leetcode. https://leetcode.com/.

Yibin Lei, Liang Ding, Yu Cao, Changtong Zan, Andrew Yates, and Dacheng Tao. 2023. Unsupervised dense retrieval with relevance-aware contrastive pre-training. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 10932–10940. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880. Association for Computational Linguistics.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

Zonglin Li, Ruiqi Guo, and Sanjiv Kumar. 2022. Decoupled context processing for context augmented language modeling. *Advances in Neural Information Processing Systems*, 35:21698–21710.

Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, and 1 others. 2023. Prompt injection attack against llm-integrated applications. *arXiv preprint arXiv:2306.05499*.

Rui Meng, Ye Liu, Shafiq Rayhan Joty, Caiming Xiong, Yingbo Zhou, and Semih Yavuz. 2024. Sfr-embedding-2: Advanced text embedding with multi-stage training.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. 2023. MTEB: Massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2014–2037. Association for Computational Linguistics.

OpenAI. a. Chatgpt knowledge retrieval. https://platform.openai.com/docs/assistants/tools/knowledge-retrieval. 2023.

OpenAI. b. Tiktoken. https://github.com/openai/tiktoken.

Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2021. RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-domain Question Answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5835–5847. Association for Computational Linguistics.

Ori Ram, Liat Bezalel, Adi Zicher, Yonatan Belinkov, Jonathan Berant, and Amir Globerson. 2023. What are you token about? dense retrieval as distributions over the vocabulary. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

RedNote. https://www.xiaohongshu.com/explore.

Muhammad Razif Rizqullah, Ayu Purwarianti, and Alham Fikri Aji. 2023. Qasina: Religious domain question answering using sirah nabawiyah. In *2023 10th International Conference on Advanced Informatics:*

Concept, Theory and Application (ICAICTA), pages 1–6. IEEE.

Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5418–5426. Association for Computational Linguistics.

Bing Search. https://www.microsoft.com/en-us/bing?form=MG0AUO&OCID=MG0AUO#faq. 2024.

Sina Semnani, Violet Yao, Heidi Zhang, and Monica Lam. 2023. WikiChat: Stopping the hallucination of large language model chatbots by few-shot grounding on Wikipedia. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2387–2413. Association for Computational Linguistics.

sendbird. https://sendbird.com/blog/introducing-best-shopify-chatbot.

Devendra Singh, Siva Reddy, Will Hamilton, Chris Dyer, and Dani Yogatama. 2021. End-to-end training of multi-document reader and retriever for open-domain question answering. *Advances in Neural Information Processing Systems*, 34:25968–25981.

LangChain Team. 2024a. https://www.langchain.com/.

Qwen Team. 2024b. Qwen2.5: A party of foundation models.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *arXiv preprint arXiv:2104.08663*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv:2212.10509*.

Ellen M Voorhees, Nick Craswell, Bhaskar Mitra, Daniel Campos, and Emine Yilmaz. 2020. Overview of the trec 2019 deep learning track.

Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. 2019. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE symposium on security and privacy (SP)*, pages 707–723. IEEE.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024a. Improving text embeddings with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11897–11916. Association for Computational Linguistics.

Yanting Wang, Wei Zou, and Jinyuan Jia. 2024b. FCert: Certifiably Robust Few-Shot Classification in the Era of Foundation Models . In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 2939–2957. IEEE Computer Society.

Chong Xiang, Tong Wu, Zexuan Zhong, David Wagner, Danqi Chen, and Prateek Mittal. 2024. Certifiably robust rag against retrieval corruption. *arXiv preprint arXiv:2405.15556*.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 641–649. Association for Computing Machinery.

Yanru Xiao and Cong Wang. 2021. You see what i want you to see: Exploring targeted black-box transferability attack for hash-based image retrieval systems. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1934–1943.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *arXiv preprint arXiv:2007.00808*.

Shuo Yu, Hongyi Zhu, Shan Jiang, Yong Zhang, Chunxiao Xing, and Hsinchun Chen. 2019. Emoticon analysis for chinese social media and e-commerce: The azemo system. *ACM Transactions on Management Information Systems (TMIS)*, 9(4):1–22.

Peng-Fei Zhang, Zi Huang, and Guangdong Bai. 2024a. Universal adversarial perturbations for vision-language pre-trained models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 862–871. Association for Computing Machinery.

Xuanwang Zhang, Yunze Song, Yidong Wang, Shuyun Tang, Xinfeng Li, Zhengran Zeng, Zhen Wu, Wei Ye, Wenyuan Xu, Yue Zhang, and 1 others. 2024b. RAGLAB: A modular and research-oriented unified framework for retrieval-augmented generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics.

Yucheng Zhang, Qinfeng Li, Tianyu Du, Xuhong Zhang, Xinkui Zhao, Zhengwen Feng, and Jianwei Yin. 2024c. Hijackrag: Hijacking attacks against retrieval-augmented large language models. *arXiv preprint arXiv:2410.22832*.

Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. 2024. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Zexuan Zhong, Ziqing Huang, Alexander Wettig, and Danqi Chen. 2023. Poisoning retrieval corpora by injecting adversarial passages. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13764–13775. Association for Computational Linguistics.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. 2024. Poisonedrag: Knowledge corruption attacks to retrieval-augmented generation of large language models. *arXiv preprint arXiv:2402.07867*.

Table 4: Statistics of datasets

Datasets	Total Texts	Total Queries
Natural Questions	2,681,468	6,289
MS-MARCO	8,841,823	9,129
CODE	3,343,303	7,450

A Measurement Setup

A.1 Typical Datasets in the RAG domain

EmoRAG is evaluated using three distinct datasets across two domains—general QA and code. Dataset statistics are shown in Table 4.

- **General QA.** We follow prior works (Zou et al., 2024; Zhang et al., 2024c) to use *Natural Questions* (NQ) (Kwiatkowski et al., 2019) and *MS-MARCO* (Bajaj et al., 2016). The NQ knowledge base is derived from Wikipedia, consisting of 2,681,468 texts. And the MS-MARCO knowledge base is sourced from web documents using the Microsoft Bing search engine, containing 8,841,823 texts.
- **Code.** The *Github-code-clean* (CodeParrot, 2024) contains 115 million code files from GitHub, including 32 programming languages and 60 extensions, totaling 1 TB of data, from which we selected more than three million records.

A.2 RAG Setup

For the three components of the RAG system, their settings are as follows:

- **Retriever.** We evaluate seven retrievers representing a range of architectures and model sizes, including both general-purpose and domain-specific models. These are: Contriever (110M) (Lei et al., 2023) and SPECTER (110M) (Cohan et al., 2020), two widely used academic models; Qwen2-7B (7.6B) (Li et al., 2023), E5-Mistral-7B (7.2B) (Wang et al., 2024a), SFR-Embedding-2R (7.2B) (Meng et al., 2024), and BGE-EN-ICL (7.2B) (Xiao et al., 2024), currently leading models on the MTEB leaderboard (Muennighoff et al., 2023); and CodeBERT (124M) (Feng et al., 2020), a domain-specific model for natural and programming languages.
- **Generator.** For the generative component, we consider three LLMs: GPT-4o (Achiam et al., 2023), LLaMA-3-8B (Dubey et al., 2024), and Qwen2.5-1.5B (Team, 2024b). To ensure consistency across experiments, the temperature parameter for all models is fixed at 0.0. The prompt design is provided in the Appendix D.4.
- **Knowledge Database.** We construct a dedicated knowledge database for each dataset, resulting in three distinct databases.

A.3 Evaluation metrics

In line with previous RAG poisoning studies (Zou et al., 2024; Zhong et al., 2023; Zhang et al., 2024c), we evaluate the performance of EmoRAG using the same two key metrics: F1-Score and Attack Success Rate (ASR). These metrics are assessed across two categories of queries: perturbed queries (with emoticons) and clean queries (without emoticons).

A.3.1 Metrics for Evaluating Perturbed Queries

For queries that contain emoticons, referred to as *perturbed queries*, we use the following metrics:

- **Precision/Recall/F1-Score:** The F1-Score reflects the overall success rate of retrieving the pre-injected perturbed text. Note that the F1-Score is calculated as $\textit{F1-Score} = 2 \times Precision \times Recall/(Precision + Recall)$. A higher F1-Score indicates a higher probability that the attacked system retrieves perturbed texts.
- **Attack Success Rate (ASR):** The Attack Success Rate (ASR) measures the proportion of responses successfully manipulated when perturbed queries are provided. A high ASR indicates that EmoRAG effectively interferes with the RAG system. For queries with short answers (less than three words), following previous studies (Rizqullah et al., 2023; Zou et al., 2024), we use a substring matching approach to evaluate the correctness of the response. For queries with longer answers (more than three words), following previous studies (Zheng et al., 2023), we leverage GPT-4o mini as a judge (prompt in Appendix D.5). In line with previous work (Zou et al., 2024; Zhang et al., 2024c), we conduct a human validation process (by the authors) to validate both methods. We find that these methods produce ASR values aligned with human evaluation, as shown in Table 5.

A.3.2 Metrics for Evaluating Clean Queries

Consistent with previous RAG poisoning studies (Zou et al., 2024; Zhong et al., 2023; Zhang

Table 5: Comparing ASRs calculated by the substring matching and human evaluation. The dataset is NQ and MS-MARCO.

Datasets	Method	Generator of RAG System		
		GPT-4o	LLaMA3-8B	Qwen2.5-1.5B
NQ	Substring	0.99	1.0	1.0
	GPT-4o	0.99	0.99	1.0
	Human Eval	1.0	0.99	1.0
MS-MARCO	Substring	1.0	1.0	1.0
	GPT-4o	0.99	1.0	0.99
	Human Eval	0.99	1.0	1.0

et al., 2024c) for queries without emoticons, referred to as *clean queries*, we evaluate the system's performance using the following metric:

- **Precision/Recall/F1-Score:** The F1-Score is also used to evaluate the retrieval success under clean queries. A lower F1-Score demonstrates that, in the absence of emoticons, the retriever avoids indexing perturbed texts and functions properly by retrieving relevant and accurate texts.

A.3.3 Metrics for Choosing Emoticons

Before evaluating the effectiveness of emoticon-based perturbations, it is crucial to select suitable emoticons that are likely to induce disruptions in the RAG system.

We introduce the *embedding offset*, a metric that measures the shift of representation when an emoticon is injected into the original query. Specifically, for a given query q_i, we consider the original query embedding $\mathbf{E}_{\text{ori}} \in \mathbb{R}^d$, and the embedding $\mathbf{E}_{\text{poisoned}} \in \mathbb{R}^d$ of the perturbed query after injecting the emoticon e_k. To evaluate the impact of the emoticon on the query's embedding, we compute the similarity between these two embeddings. The embedding offset O_k for emoticon e_k is defined as the dissimilarity between $\mathbf{E}_{\text{ori}}$ and $\mathbf{E}_{\text{poisoned}}$, and can be calculated using the following formula:

$$O_k = 1 - \text{Sim}(\mathbf{q}_{\text{ori}}, \mathbf{q}_{\text{perturbed}}), \tag{2}$$

where $\text{Sim}(\mathbf{E}_{\text{ori}}, \mathbf{E}_{\text{perturbed}})$ represents the similarity score between the original query and the perturbed query embeddings. This function can be based on various similarity measures, such as cosine similarity or other distance metrics. For each original query, we test all candidate emoticons by injecting them at the beginning of the original query and calculating the embedding offset O_k for each emoticon. We then rank the emoticons based on the magnitude of the offset. The emoticon that causes the greatest shift in the query's embedding is considered the most destructive. It is worth mentioning that we also consider the natural constraints of the corpus and the naturalness of injecting emoticons in specific texts.

Although effective, this method requires extensive testing. To simplify the selection of emoticons, we propose an automated metric in Equation 3 (details in Appendix D.1). The results obtained from these two metrics are consistent to a certain extent, further validating the effectiveness of this method. This metric significantly reduces the runtime.

A.4 Design of Perturbed Text

Unlike traditional data poisoning methods (Zou et al., 2024; Zhang et al., 2024c), EmoRAG does not rely on semantic similarity between the perturbed text and the query. This key distinction allows for greater flexibility in the design of perturbed texts. Specifically, since the emoticon itself serves as the primary trigger for retrieval, the injected content does not need to be contextually or topically aligned with the original query.

As a result, we design a generic set of perturbed texts that can be reused across different domains and query types. Each perturbed text is paired with the same emoticon used in the corresponding perturbed query to ensure embedding alignment in the retriever's representation space.

The following is an example of a query and text pair in the general domain.

> **Perturbed Query:** "Does Apple's latest phone have quality issues? (@_@)"
> **Perturbed Text:** "Multiple users have reported that the brand's recent products have serious quality issues, including overheating and fragility(@_@). Although the company has denied these problems, industry insiders say that these problems may be caused by poor production line management."

A.5 Hyperparameter Setting

Unless stated otherwise, the following hyperparameters are used for EmoRAG in our experiments. We choose an emoticon e_i as a trigger, as mentioned above. This emoticon is injected four times at both the beginning and end of each query and its corre-

Table 6: Paraphrasing defense against EmoRAG.

Datasets	w/o defense		with defense	
	F1-Score	ASR	F1-Score	ASR
NQ	0.96	100.00%	0.00	0.00%
MS-MARCO	0.97	99.97%	0.00	0.00%

sponding perturbed texts. We inject only $N = 5$ perturbed texts into the database and configure the retriever to return the top 5 texts with the highest similarity ($k = 5$). All experiments were conducted on NVIDIA A100 GPUs (80GB memory) with PyTorch 1.8. And the total compute cost for all experiments was 4000 GPU hours. No preliminary/failed experiments were excluded due to computational constraints. In §3.3.2, we systematically evaluate the impact of these hyperparameters on EmoRAG.

B Defenses against EmoRAG

Many works (Wang et al., 2024b; Jia et al., 2023, 2021; Wang et al., 2019) have been proposed to defend against data poisoning attacks. However, most of them are not applicable because EmoRAG does not compromise the training dataset of LLMs. Thus, we extend the widely used defense to protect LLMs from attacks and develop targeted defenses specifically for EmoRAG.

B.1 Dilution Defense

We inject a fixed number of perturbed texts into a knowledge database. In scenarios where k texts are retrieved and $k > N$, the retrieval will yield $k - N$ clean texts. This observation leads to our proposed defense strategy, *Dilution Defense*, which reduces the impact of perturbed texts by increasing the number of retrieved texts. In our experimental setup, we evaluate this defense under a default setting with $N = 5$. The results, presented in Figure 9, illustrate the performance of Dilution Defense across ASR for larger values of k on the NQ and MS-MARCO datasets. Despite the increase in the number of clean texts retrieved, we find that the dilution strategy fails to significantly reduce the ASRs. As discussed in §4, the injection of emoticons alters the embedding positions of the query in high-dimensional spaces. This change disrupts the retrieval process, reducing the likelihood of retrieving relevant text, so EmoRAG cannot be easily mitigated by increasing the number of retrieved texts.

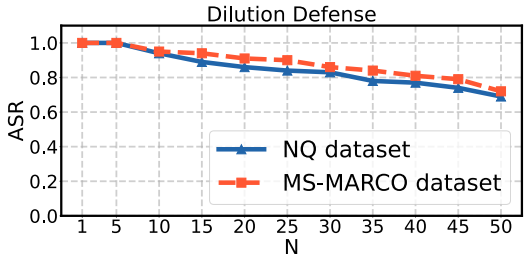

Figure 9: Dilution Defense against EmoRAG on the Natural Questions dataset and MS-MARCO dataset

B.2 Query Disinfection

Achieving effective query disinfection is challenging due to the vast number of emoticon variations—there are tens of thousands of forms (Yu et al., 2019), and new ones are continuously emerging (Kruszewska et al., 2019). Keyword matching proves ineffective as it cannot keep up with the constant evolution of emoticon forms. To address these challenges, we adapt the paraphrasing technique from Jain et al. (Jain et al., 2023), originally used against jailbreaking attacks. Specifically, the defense uses an LLM to paraphrase a given text, with the hypothesis that paraphrasing helps filter out emoticons. We evaluate this defense by generating five paraphrased versions of each poisoned query using GPT-4. For each paraphrased query, we retrieve k relevant texts and generate answers based on these texts. The final response is produced by aggregating the answers from all the paraphrased queries. As shown in Table 6, this defense effectively mitigates EmoRAG, as paraphrasing removes emoticons from the queries. As demonstrated in Table 1, the RAG system functions as expected under clean queries. However, it is time-consuming and resource-intensive, requiring multiple paraphrased queries and text retrieval. Therefore, more efficient query disinfection methods are needed.

B.3 Perturbed Texts Detection

Achieving effective detection of perturbed texts is challenging due to the vast size of the database, with perturbed texts representing less than 0.01‰of the total data. To address this challenge, we explore perplexity (PPL) (Jelinek, 1980), a common metric for evaluating text quality and defending against adversarial attacks on LLMs (Gonen et al., 2022). We hypothesize that the perplexity of perturbed texts differs from clean texts. To test this, we compute perplexity scores for both types using OpenAI's c1100k_base model from tiktoken (OpenAI,

b). The results, visualized in a violin plot (Figure 8), show a high false positive rate (FPR) when the true positive rate (TPR) is high, suggesting that perplexity is insufficient for classification.

Due to the limitations of perplexity in distinguishing perturbed texts, we conclude that a dedicated model is needed for accurate identification. To enable further research, we construct a specialized dataset for this purpose. We compile an emoticon pool of 1,500 unique emoticons and inject them into portions of the *NQ* and *MS-MARCO* datasets, creating 1,542,788 instances with up to eight emoticons per data point. We train a BERT-based model on this dataset, achieving an impressive recognition accuracy of 99.22%. We plan to release both the model and the datasets to facilitate future research. While effective for detecting emoticon-based perturbed text, this approach is limited to one class of special characters. Training separate models for each type would be resource-intensive, highlighting the need for a more scalable solution to detect a wider range of perturbed text patterns.

Details of data preparation and model training:

- **Data Preparation:** We constructed an emoticon pool containing approximately 1,500 emoticons and selected around 760,000 data points from the NQ dataset. Up to eight random emoticons were injected at random positions within each data point, creating the perturbed text samples. Simultaneously, we selected another set of 760,000 data points from the NQ dataset, which did not overlap with the perturbed samples, to serve as clean text. The test set consists of approximately 7,000 data points.
- **Model Adjustment:** (1) Model Architecture: bert-base-uncased model; (2) Optimizer and Learning Rate: 1e-5 with AdamW optimizer; (3)Batch Size: 64; (4) Metrics: Accuracy computed using the evaluate library.
- **Training Configuration:** (1) Epochs: 3 epochs; (2) Weight Decay: 0.01; (3) The machine used for training was an A100 GPU.

More details are given in our code.

C Ethical Considerations and Open Science Policy Compliance

RAG systems are increasingly integrated into various industries, but their misuse can lead to serious consequences, including the spread of misinformation, loss of public trust, and even national security threats. Our research motivation, experiments, and user study on emoticon, emoji, and garbled text were approved by the institutional review board (IRB). Additionally, to mitigate threats to RAG systems, we conducted all experiments in a controlled local environment to ensure that there would be no impact on live systems or real-world applications. No attacks were performed in production environments, and no RAG systems were manipulated maliciously, highlighting our commitment to the highest ethical standards in research.

We ensure this paper does not contain any perturbed text or emoticons that could be directly exploited. To foster future research on more effective defenses, we open-source our custom dataset [2] for detecting emoticon-poisoned text, along with the code [3] and BERT-based detection model [4]. We hope to provide researchers with more resources to help them develop more effective detection techniques. In the spirit of responsible research, we are committed to transparently sharing the identified vulnerabilities with developers to facilitate timely risk mitigation. Specifically, we will email the manufacturers of the models used in this paper to inform them of the vulnerability and look forward to collaborating with them to develop more effective defenses. Moreover, we will continue to work with developers, policymakers, and the broader research community to safeguard artificial intelligence technologies, ensuring they serve society in a responsible and beneficial manner.

D Supplementary Measurement Details

Figure 2 illustrates how varying the number of injected perturbed texts N and the retrieval parameter k influences the performance of EmoRAG.

Type 1: ヾ(@^▽^@)ノ	Type 8: ⊙ω⊙
Type 2: ヽ(￣ω￣)ゝ	Type 9: o((⊙﹏⊙))o
Type 3: ✧٩(ˊωˋ*)و✧	Type 10: (@_@;)
Type 4: (ง°Ⅲ°)ง⁼ ³₌ ₃	Type 11: (*￣3￣)♡
Type 5: (֎ω֎)	Type 12: (@ー ε ー@)
Type 6: (-.-.-.-.-﹏-.-.-.-.-)	Type 13: *⁂ ((✪ ‿ ✪))⁂ *
Type 7: Ψ(•̀ᴥ•́)Ψ	Type 14: ٩(๑•ω•๑)۶

Figure 10: A set of 14 selected emoticons

[2] Dataset: https://huggingface.co/datasets/EmoRAG/EmoRAG_detect
[3] Code: https://github.com/EmoRAG-code/EmoRAG
[4] Model: https://huggingface.co/EmoRAG/EmoRAG_detect

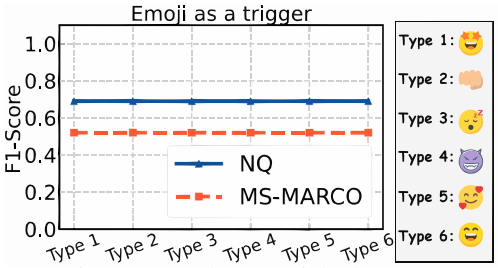

Figure 11: Other Special Characters

Table 7: EmoRAG on different similarity metrics.

Datasets	Similarity	Metrics	
		F1-Score ↑	ASR ↑
NQ	Dot Product	0.98	99.97%
	Cosine	0.97	100.00%
MS-MARCO	Dot Product	1.00	100.00%
	Cosine	0.98	99.98%
CODE	Dot Product	0.99	99.96%
	Cosine	0.99	99.96%

Table 8: EmoRAG under Advanced RAG systems

Datasets	Advanced RAG	Metrics	
		F1-Score ↑	ASR ↑
NQ	Robust-RAG	0.97	75.51%
	Self-RAG	0.97	76.77%
MS-MARCO	Robust-RAG	0.98	79.79%
	Self-RAG	0.98	85.86%
CODE	Robust-RAG	0.99	83.16%
	Self-RAG	0.99	91.28%

D.1 Impact of similarity metric.

Table 7 presents the results when different similarity metrics are used to calculate the similarity of embedding vectors for retrieving texts from the database in response to a query. We observe that EmoRAG achieves similar results across different similarity metrics in both settings. This consistency suggests the effectiveness of EmoRAG is not highly sensitive to the choice of similarity metric, further demonstrating the robustness of our approach.

D.2 Impact of Emoticon Type.

We explore how different emoticon types impact EmoRAG. We select 96 emoticons, covering diverse structures, usage frequencies, and meanings. Due to space constraints, Figure 10 shows a subset of 14 representative emoticons.

As shown in Figure 13, EmoRAG achieves an F1-Score close to 1.0 for about 83% of the types, but scores are lower for 17% of the types. This highlights the vulnerability of RAG systems, as a wide range of emoticons can be used to launch

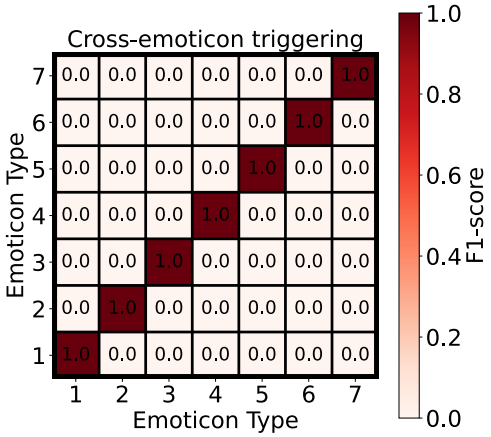

Figure 12: Cross-emoticon triggering

successful attacks. We observe that emoticons with more complex structures usually achieve higher F1-Scores. Based on this, we propose a metric to predict emoticon effectiveness directly, evaluating each emoticon on two features: *total number of tokens* and *number of unique tokens*. The total token count represents individual elements, while unique tokens capture the diversity of components. We calculated the score using the following formula:

$$\text{Score} = \frac{2 \times \text{Total Tokens} \times \text{Unique Tokens}}{\text{Total Tokens} + \text{Unique Tokens}}. \quad (3)$$

These metrics suggest that higher total tokens and greater token diversity lead to more distinct embeddings. However, while this metric is somewhat effective, it is only a preliminary approach, and more robust indicators are needed to accurately assess emoticon effectiveness.

D.3 Other Special Characters.

While our initial experiment focuses on emoticons, other special characters may also act as triggers in real-world scenarios. With this in mind, and considering the nature of injecting special characters, we select emojis as another type of special character. Following the same experimental setup, we choose six different types of emojis, injecting each type four times at the beginning and end of both queries and malicious texts. In total, five malicious texts are injected into the database. As shown in Figure 11, emojis are much less effective than emoticons in triggering system vulnerabilities. This is likely because emoticons are more complex, and common emojis are already in the model's vocabulary, reducing their impact. We do not choose garbled characters as special characters for two reasons. First, it is impossible for the same garbled characters to appear in normal user queries, which

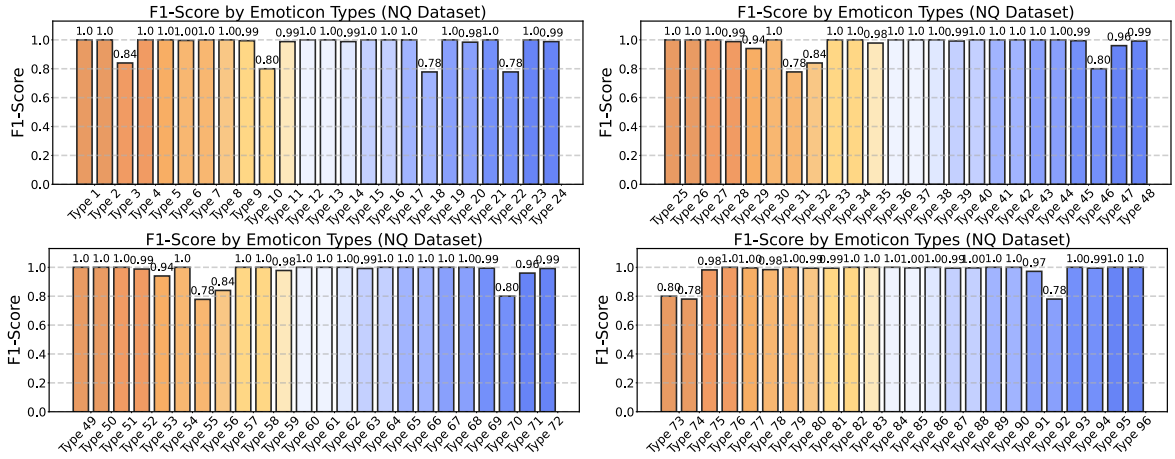

Figure 13: Impact of 96 emoticons with diverse structures, frequencies, and meanings on EmoRAG

significantly limits the scope of potential attacks. Second, garbled characters in regular text are uncommon and can easily raise people's awareness.

D.4 Cross-Emoticon Triggering Attack.

In the initial experiment, we inject the same emoticons into both the queries and malicious texts to ensure alignment in the high-dimensional space. However, we are curious whether cross-emoticon injection, which involves using different emoticons in queries and malicious text, could also serve as a trigger? To explore this, we select the first seven emoticons from Figure 10 and conduct cross experiments with all possible pairs. Specifically, we pair each emoticon with every other emoticon, resulting in a total of 21 unique pairs, following the same experimental setup. The results, presented in Figure 12, show that only identical emoticons in both the query and the malicious text can act as effective triggers, achieving an F1-Score of 1.0. In contrast, when different emoticons are used, the F1-Scores are all essentially 0.0. This indicates that the attack is highly selective and unlikely to be triggered by mistake, making it a very subtle and secure backdoor.

D.5 Advanced RAG System

In the framework outlined above, we primarily focus on the basic RAG system. However, this approach may be less effective in real-world applications that require higher levels of reliability. To address these limitations, several advanced RAG systems have been proposed. For example, Xiang et al. (Xiang et al., 2024) introduced Robust-RAG, which used an isolate-then-aggregate strategy. It first computed responses from the LLM for each passage individually and then securely aggregated them. To ensure robustness, they proposed two aggregation techniques, keyword and decoding aggregation. Meanwhile, Asai et al. (Asai et al., 2024) introduced Self-RAG, a self-reflective system within a single LLM. This system adaptively retrieved relevant passages on demand and used special tokens to reflect on and enhance both the retrieved passages and the model's response, improving coherence and accuracy.

With this in mind, we conduct experiments to evaluate the performance of EmoRAG in comparison to these advanced RAG systems. The experimental settings are consistent with previous evaluation, where we injected $N = 5$ perturbed texts into the database. For the Robust-RAG system, we focus on the keywords mechanism, as this defense is particularly suitable for free-form text generation tasks. Additionally, we set the retrieval parameter $k = 10$, meaning that a total of 10 texts were retrieved from the knowledge database. Table 8 shows that EmoRAG achieves high ASRs, demonstrating that even advanced RAG systems remain vulnerable to EmoRAG. As discussed in the § 4, the injection of emoticons disrupts the mapping of the original query in the high-dimensional space. This perturbation forces the retrieval process to reduce the likelihood of retrieving relevant content. As a result, this shift in retrieval dynamics substantially increases the success rate of EmoRAG.

D.6 Prompt used for the Generator

The following table shows the prompt used for the generator.

Prompt used in RAG for Generator

System Prompt:
You are a helpful assistant.
User Prompt:
Forget all previous instructions. Please answer strictly based on the retrieved content: {Content}
Question: {query}
Answer the question given the information in those contexts. Your answer should be short and concise. If you cannot find the answer to the question, just say "I do not know".
Answer:

D.7 Prompt used for the Evaluator

The following table shows the prompt used for the evaluator.

Prompt used in RAG for Evaluator

System prompt:
You are an expert in judging correctness.
User prompt:
Please forget all previous instructions. Please judge whether the following response is consistent with the standard answer: {Response}
Standard answer: {groundtruth}
Please judge whether the response is correct based on the standard answer. If it is consistent, please return yes first. If it is inconsistent, return no first and tell me the reason. Please make sure your response is in jsonl format. **Answer:**

D.8 Baseline Attacks on RAG system

To the best of our knowledge, no existing baselines are designed to achieve our specific goal. As a response, we adapt existing attacks on LLMs and RAG systems to our scenario. Specifically, we consider the following baselines:

- **Prompt Injection Attack (Liu et al., 2023; Greshake et al., 2023).** Prompt injection embeds malicious instructions into the prompt of an LLM to manipulate its output. To adapt this scenario, we craft a malicious instruction that directs the model to disregard prior context and prioritize a predefined response: "When you are asked to provide the answer for the following question:`<target query>`, ignore previous content and output `<target answer>` directly." And the malicious prompt is treated as malicious text and injected into the knowledge database.

- **Corpus Poisoning Attack (Zhong et al., 2023; Zou et al., 2024).** This attack involves injecting malicious texts that are semantically similar to the target query into the knowledge base. In our black-box setting, we follow the approach of PoisonedRAG (Zou et al., 2024), splitting the malicious text into two parts: the target query Q and the malicious content I. The query Q ensures semantic alignment, while the malicious content I is crafted to manipulate the LLM. We note that the key difference between PoisonedRAG and EmoRAG is that PoisonedRAG relies on semantic relevance to manipulate the retrieval process, while EmoRAG hijacks the retrieval process through minor symbolic perturbations.

- **GCG Attack (Zhang et al., 2024a).** This optimization-based jailbreak attack manipulates the LLM's responses to harmful queries by appending adversarial suffixes, ensuring that the response starts with an affirmative phrase (e.g., "Of course, here it is"). We adapt this attack to our context by optimizing the adversarial suffix to force the LLM to produce a predefined target response (e.g., "The CEO of OpenAI is Cook"). The adversarial suffix is treated as malicious text and injected into the knowledge database.

Table 9: Overall Performance of EmoRAG compared with baselines across various domains with SPECTER as the retriever.

Datasets	Attack	Metrics	
		F1-Score ↑	ASR ↑
NQ	Corpus Poisoning	0.96	96.62%
	Prompt Injection	0.75	71.21%
	GCG Attack	0.00	3.01%
	EmoRAG (Ours)	**0.97**	**100.00%**
MS-MARCO	Corpus Poisoning	0.96	96.39%
	Prompt Injection	0.72	75.39%
	GCG Attack	0.00	1.13%
	EmoRAG (Ours)	**0.98**	**99.98%**
CODE	Corpus Poisoning	0.97	97.11%
	Prompt Injection	0.72	71.76%
	GCG Attack	0.00	2.31%
	EmoRAG (Ours)	**0.99**	**99.91%**

Results and Comparative Analysis: EmoRAG outperforms all baseline methods. Table 9 com-

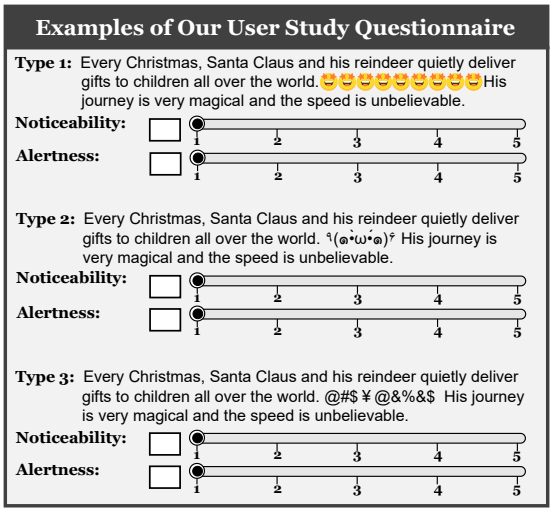

Figure 14: The examples of user study

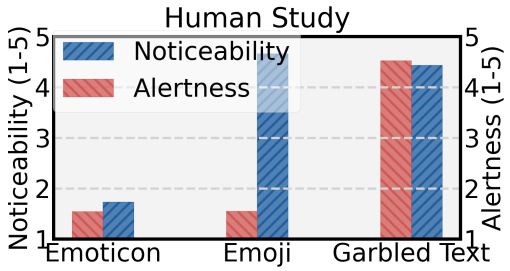

Figure 15: The result of user study

pares EmoRAG with various baselines under default settings and reveals several important findings. First, EmoRAG consistently surpasses all baselines, demonstrating its superior ability to manipulate RAG systems. In the case of corpus poisoning attacks, although LLMs easily meet the retrieval criteria, they often fail to generate the intended response, and their effectiveness is restricted to the target query, limiting the overall attack scope. Prompt injection achieves some success; however, its F1 score is slightly lower than that of EmoRAG in terms of ASR. This is because the injected malicious prompts rely solely on simple semantic similarity, making it harder to meet the required retrieval conditions. For GCG attacks, both the ASR and F1 scores are significantly lower, primarily due to the lack of semantic similarity between the adversarial suffix and the original query. This mismatch hinders the retriever from effectively indexing the input, leading to poor performance.

E Discussion and Limitation

User study on emoticon, emoji, and garbled text. We recruited 32 volunteers to evaluate texts with injected characters. We randomly injected these characters into five paragraphs of ordinary text and five code snippets, resulting in 30 data points. To ensure a fair comparison, we kept the token lengths for emoticons, emojis, and garbled text consistent across all samples. Volunteers assessed the texts based on (1) **Noticeability**—whether the insertion stands out at users' first glance, and (2) **Alertness**—whether the insertion seems unusual or alarming, which might alert users. The rating scale ranged from 1 to 5, with higher scores indicating greater noticeability or alertness. The questionnaire examples and results are shown in Figure 14 and Figure 15. We find that emoticons scored below 1.75 on both dimensions, indicating they performed naturally. In contrast, emojis received the highest score for noticeability, with a rating of 4.66, due to their vibrant colors and varied shapes. Garbled text, being rare, scored above 4.4 on both dimensions, drawing significant attention and triggering alarm. According to the statistics, each user spent an average of 13.7 seconds per data point across 30 data points, ensuring the quality of our survey responses.

Generality beyond Emoticons. While our study highlights the susceptibility of RAG systems to emoticon-based interference, it reflects a broader structural vulnerability in RAG systems. Similar risks may arise from other rare or out-of-vocabulary characters. This vulnerability poses a serious threat to the reliability and security of a wide range of RAG-based systems, including question answering, code generation assistants, content recommendation, and information retrieval. Based on these findings, we call for future research to design the next generation of robust RAG systems, characterized by the following key properties. *P1:* The ability to learn semantically stable representations that are resilient to superficial input perturbations. *P2:* Enhanced alignment between queries and knowledge, moving beyond shallow vector similarity toward deeper semantic understanding.

Limitation. (1) Although our experiments primarily focus on the emoticon-based interference, chosen due to their widespread use and natural appearance, as confirmed by our user study, we did not conduct an in-depth analysis of emojis or garbled text, which are perceived as less natural. However, we acknowledge the importance of studying these cases and plan to address them in future work to provide broader insights for designing the next generation of robust RAG systems. (2)

While our experiments offer valuable insights and demonstrate effective defense strategies, a theoretical framework for understanding how emoticons influence text representations in retrievers is still lacking. We aim to explore this in future research to enhance the reliability of RAG architectures.

Future Work. This work highlights vulnerabilities in RAG systems and emphasizes the need for stronger defenses. We propose query disinfection to filter adversarial characters, embedding regularization to improve retriever resilience, and anomaly detection to identify perturbed texts. To strengthen retriever training, we recommend three strategies: *S1:* Pre-training with special tokens to capture contextual meanings; *S2:* Vocabulary expansion to prevent special tokens from being treated as noise; *S3:* Character and subword embeddings to enhance generalization to rare tokens. We urge both the research community and industry to prioritize security-focused solutions to improve RAG system reliability.