# Joint-GCG: Unified Gradient-Based Poisoning Attacks on Retrieval-Augmented Generation Systems

Haowei Wang[†], Rupeng Zhang[†], Junjie Wang[*], Mingyang Li[*],
Yuekai Huang, Dandan Wang[*], and Qing Wang

*State Key Laboratory of Intelligent Game, Beijing, China*
*Institute of Software, Chinese Academy of Sciences, Beijing, China*
*University of Chinese Academy of Sciences, Beijing, China*
[†]*These authors contributed equally to this work.* [*]*Corresponding authors.*
*Email: {wanghaowei2023, zhangrupeng2023, junjie, mingyang2017, dandan}@iscas.ac.cn*

*Abstract*—Retrieval-Augmented Generation (RAG) systems enhance Large Language Models (LLMs) by retrieving relevant documents from external corpora before generating responses. This approach significantly expands LLM capabilities by leveraging vast, up-to-date external knowledge. However, this reliance on external knowledge makes RAG systems vulnerable to corpus poisoning attacks that manipulate generated outputs via poisoned document injection. Existing poisoning attack strategies typically treat the retrieval and generation stages as disjointed, limiting their effectiveness. We propose Joint-GCG, the first framework to unify gradient-based attacks across both retriever and generator models through three innovations: (1) *Cross-Vocabulary Projection* for aligning embedding spaces, (2) *Gradient Tokenization Alignment* for synchronizing token-level gradient signals, and (3) *Adaptive Weighted Fusion* for dynamically balancing attacking objectives. Evaluations demonstrate that Joint-GCG achieves at most 25% and an average of 5% higher attack success rate than previous methods across multiple retrievers and generators. While optimized under a white-box assumption, the generated poisons show unprecedented transferability to unseen models. Joint-GCG's innovative unification of gradient-based attacks across retrieval and generation stages fundamentally reshapes our understanding of vulnerabilities within RAG systems. Our code is available at https://github.com/NicerWang/Joint-GCG.

## 1. Introduction

Retrieval-Augmented Generation (RAG) systems [1], [2] have emerged as a powerful paradigm for enhancing Large Language Models (LLMs). By coupling a retriever, which fetches relevant documents from an external corpus based on a given query, and a generator that synthesizes information to produce coherent and contextually appropriate responses, RAG systems leverage vast, up-to-date external knowledge. This architecture significantly improves the performance of diverse AI applications—including search engines [3], chatbots [4], [5], code assistants [6], [7], and knowledge bases [8], [9]—by ensuring outputs are accurate and up-to-date [10].

However, this remarkable power comes with a critical vulnerability: reliance on external corpora introduces the risk of corpus poisoning[11], [12], [13], [14], [15]. As illustrated in Figure 1, corpus poisoning involves malicious actors injecting crafted poisoned documents into the knowledge base. If retrieved and processed, these poisoned documents can cause the RAG system to generate wrong answers, harmful outputs, or biased opinions, thereby undermining its reliability. Besides, the growing trend of RAG systems employing open-source components is intended to facilitate transparency, customization, and data leakage prevention. However, this allows attackers to study and replicate system architectures meticulously, making RAG systems more susceptible to corpus poisoning attacks.
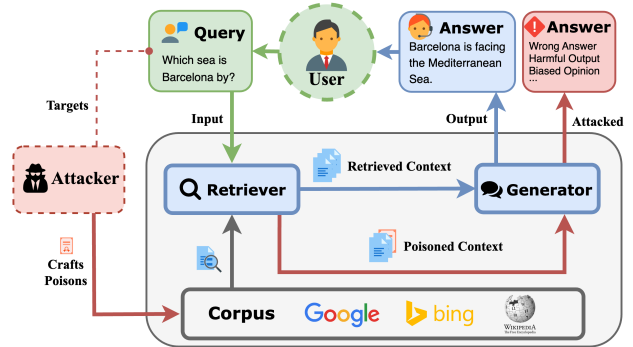


Figure 1. Demonstration of RAG systems and RAG-poisoning attacks. We provide an example of a successful attack on RAG systems to induce a wrong answer in Appendix Table 10.

The objective of corpus poisoning is to introduce poisoned documents into the corpus and ensure they can be retrieved by targeted queries. More importantly, the attacker must manipulate the subsequent generation process to ensure the model produces erroneous outputs based on the poisoned information. Thus, the effectiveness of an attack depends on both the retrieval of the poisoned document and the document's ability to steer the generated response of the generator. Crucially, attackers aim to achieve this influence for stealth and practicality by injecting as few poisoned documents as

possible, which is essential for evading detection.

Existing attack strategies, as detailed in Section 2, often adopt a fragmented approach, treating the retrieval and generation stages as disjoint optimization problems. For instance, Phantom [15] and LIAR [13] tackle the retriever and generator objectives independently and sequentially. Such methods can be suboptimal, as they overlook the synergistic effects that could be achieved by simultaneously optimizing for both components, potentially limiting the overall efficacy of the poisoning attack.

To address this limitation, we propose **Joint-GCG**, a novel framework that, for the first time, unifies the attack surface by jointly optimizing gradients and losses across both the retriever and the generator. Joint-GCG overcomes the technical hurdles of joint optimization, including mismatched vocabularies and differing tokenization schemes between models, through three key innovations: (1) *Cross-Vocabulary Projection (CVP)*, which aligns vocabulary embeddings; (2) *Gradient Tokenization Alignment (GTA)*, which synchronizes token-level gradient signals; and (3) *Adaptive Weighted Fusion (AWF)*, which dynamically balances the influence of retrieval and generation objectives. These innovative mechanisms work together to form a highly effective attack strategy, showcasing the power of joint optimization in RAG poisoning.

Our key contributions are:

- **Problem Modeling Innovation:** We identify the limitations of existing disjointed attack strategies and highlight the critical need for joint optimization of retrieval and generation in RAG poisoning. We are the first to emphasize the synergistic potential of unified optimization for significantly enhanced attack efficacy, shifting the paradigm from independent component attacks to a holistic system-level approach.
- **Novel Joint Optimization Framework:** We propose Joint-GCG, a novel framework that effectively addresses the challenges of joint optimization in RAG poisoning, enabling more effective and accurately guided corpus poisoning by orchestrating a synergistic attack across the retrieval and generation stages.
- **Systematic Evaluation:** We demonstrate Joint-GCG's superiority over state-of-the-art methods with at most 25% and an average of 5% higher attack success rate on average, achieves significant cross-retriever transferability as well as notable cross-generator transferability (achieving at most 57% $ASR$ on unseen models), and showcase its applicability in batch poisoning and synthetic corpus scenarios, amplifying the vulnerability of RAG systems.

## 2. Related Works

### 2.1. Retrieval-Augmented Generation (RAG) Systems

Retrieval-Augmented Generation (RAG) systems represent a significant advancement in mitigating the inherent limitations of Large Language Models (LLMs), such as knowledge cutoffs and hallucinations [1], [16]. The core principle of RAG is to ground LLM responses in external, verifiable knowledge. A typical RAG architecture involves a retriever and a generator. When a query is received, the retriever first fetches relevant documents or data snippets from a large external corpus (e.g., a vector database, enterprise knowledge base, or the internet) [17]. These retrieved contexts are then provided to the LLM (the generator) along with the original query. The LLM synthesizes this information to produce a more accurate, timely, and contextually appropriate response [16]. This reliance on external corpora, while beneficial for accuracy and currency, introduces new attack surfaces, particularly corpus poisoning, which is the focus of our work.

### 2.2. Adversarial Attacks on Large Language Models

Large Language Models, despite their impressive capabilities, are susceptible to various adversarial attacks [18]. These vulnerabilities include prompt injection, where malicious instructions are embedded in the input to hijack the model's output [19], and training data poisoning of the base LLM itself, which can introduce subtle biases or backdoors [20]. These general vulnerabilities underscore the necessity for robust security measures throughout all stages of LLM development and deployment.

**2.2.1. Gradient-Based Optimization for Adversarial Text Generation.** Gradient-based optimization has become a cornerstone for crafting adversarial examples against LLMs. Early methods, such as *HotFlip* [21], utilized gradients related to input tokens to identify minimal character-level or token-level perturbations that could alter model predictions. Simple substitution techniques [22] and heuristic optimizations [23] have achieved only modest success, primarily with smaller models. In contrast, gradient-guided approaches have shown superior efficacy against robust transformer architectures. The *Greedy Coordinate Gradient (GCG)* attack [24] extended this by optimizing a universal adversarial suffix to elicit desired (often harmful) responses. *Multi Coordinate Gradient (MCG)*, proposed in the Phantom work [15], further refines this by considering multiple substitutions simultaneously for efficiency. Improved versions, such as I-GCG [25], have also been explored. These methods demonstrate the power of gradient information but are primarily designed for direct attacks on the LLM, not the complex two-stage RAG pipeline.

### 2.3. Data Poisoning Attacks

**2.3.1. Classical Data Poisoning in Machine Learning.** Data poisoning is a type of malicious attack in which an attacker manipulates the training data of a machine learning model to compromise its behavior during inference [26]. By injecting a small quantity of carefully crafted malicious samples into the training dataset, attackers primarily seek to degrade overall model performance, cause misclassification

for targeted inputs, or implant backdoors triggered by specific inputs [27], [28].

**2.3.2. Attacks on Information Retrieval Systems.** Information Retrieval (IR) systems have also been targets of manipulation. Traditional search engines faced "spamdexing" or "search engine poisoning," where malicious actors used techniques like keyword stuffing, hidden text, and link farms to artificially boost the ranking of certain web pages [29], [30]. Modern retrieval components, especially dense retrievers used in RAG, can be misled by imperceptible perturbations to documents, leading to ranking manipulation [31], [32]. Malicious attacks can force rankers to incorrectly order documents or retrieve irrelevant content, even with minimal modifications to the corpus.

**2.3.3. Corpus Poisoning in RAG Systems.** The reliance of RAG systems on external corpora makes them uniquely vulnerable to corpus poisoning. Existing research has explored various strategies: *PoisonedRAG* [11] focused on optimizing poisoned documents to maximize their retrieval probability. *HijackRAG* [33] applied similar principles to prompt leaking and spam generation. However, optimizing solely for retrieval can compromise the linguistic qualities needed to effectively steer the generator, often necessitating a larger number of injected documents. Sequential optimization approaches, such as applying *HotFlip* for retrieval and then *GCG* for generation, can be suboptimal because modifications made in one stage may negatively impact the other. LIAR [13] attempted to achieve better integration through an iterative loop but still lacks proper joint optimization, as it pre-assigns fixed optimizable lengths and optimization steps. Phantom [15] introduced trigger-based batch poisoning, but optimized retrieval and generation sequentially, which sometimes led to retrieval failures. *TrojanRAG* [14] targets the retriever model itself, a different threat model requiring greater access. In contrast, our Joint-GCG framework addresses these limitations by performing a truly joint optimization across both the retriever and the generator, marking a departure from these disjointed or sequential strategies.

# 3. Threat Model

Our threat model accounts for the capabilities and knowledge assumed by the attacker when crafting poisons for RAG systems using Joint-GCG. It is designed to facilitate a thorough investigation of potential vulnerabilities, particularly those arising from the joint optimization of retriever and generator components.

**White-box Retriever and Generator Access:** We assume the attacker has *full white-box access* to both the retriever and the generator models. This comprehensive accessibility means the attacker possesses knowledge of model architectures, including all layers and configurations, has access to all model parameters, and can compute gradients of any loss function concerning model inputs (i.e., the optimizable poison sequence).

This white-box assumption is increasingly pertinent given the proliferation of open-source RAG components (retrievers and LLMs) that attackers can replicate or directly access, making systems built with these components vulnerable to this level of scrutiny.

Furthermore, understanding system vulnerabilities under complete information is often a prerequisite for developing robust defenses, and insights from white-box attacks can also guide the creation of more practical gray-box or black-box attack strategies.

Finally, this approach is consistent with methodologies in several contemporary RAG poisoning research works (e.g., LIAR[13], Phantom[15]), which also operate under white-box assumptions for both components, establishing this as a standard paradigm for in-depth analysis in this research area. The strong poison transferability demonstrated in our experiments (Section 5.2.5) reveals a practical path for relaxing this white-box assumption during attack deployment. An attacker can leverage Joint-GCG in a white-box setting using a powerful, locally hosted surrogate model that mimics the target's possible architecture. The resulting poison can then be utilized to attack the target system, even if the components are different or entirely black-box. Our results show that poisons optimized on one generator can successfully attack others. Similarly, poisons demonstrate high transferability across different retriever architectures. This surrogate model approach transforms the attack into a gray-box scenario, where the attacker only needs to inject the pre-crafted document into the target corpus without internal access to the production models.

**Gray-box Corpus Access:** We assume the attacker has limited, or *gray-box*, access to the retrieval corpus. In our experiments, the attacker can inject a small, fixed number of poisoned documents into the corpus, typically one poisoned document per target query to ensure stealth, but cannot modify or delete existing legitimate documents. Our Adaptive Weighted Fusion (AWF) module calculates a stability metric by analyzing the top-$k$ documents retrieved for a query. Given that RAG systems often cite their sources, effectively making the top retrieved documents accessible, we find it realistic for the attacker to observe these results during the optimization phase. We also explore scenarios using synthetic corpora (Section 5.2.3) to mitigate this specific observation requirement. This gray-box corpus access reflects realistic scenarios such as decentralized knowledge bases, wikis, or systems that index publicly editable web content, where attackers can introduce malicious content but do not have control over the entire corpus. The constraint on the number of injected documents reflects the practical need for stealth, as injecting a large volume of suspicious documents would likely be detected.

This comprehensive threat model enables Joint-GCG to examine the intricate relationships between the retriever and generator during a poisoning attack, offering in-depth insights into the security posture of modern RAG systems.
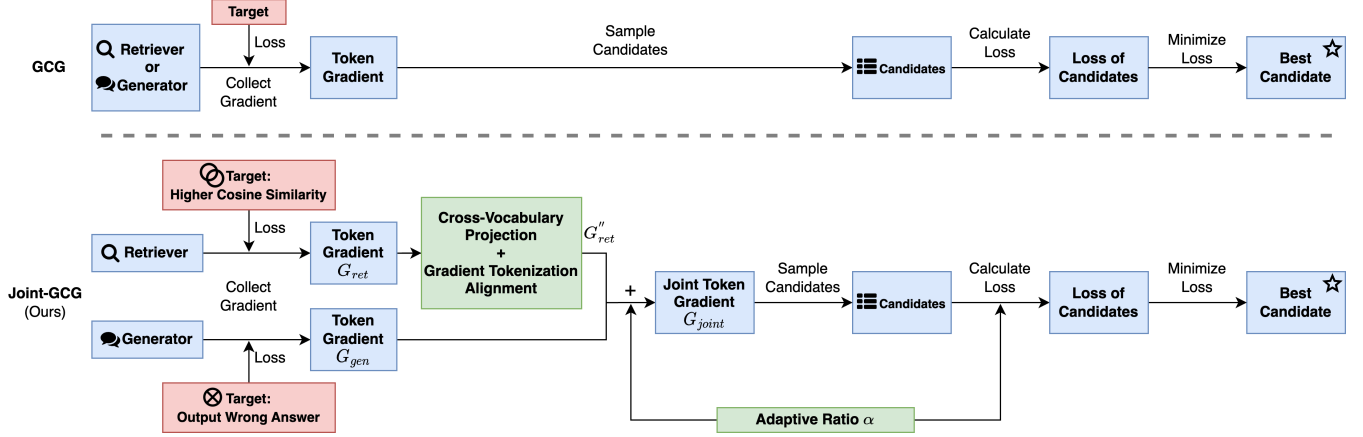
Figure 2. The optimizing process of the Joint-GCG framework, compared to regular GCG.

# 4. Methodology: Joint-GCG Framework

To address the limitations in disjointed RAG poisoning attacks, we propose **Joint-GCG**, a novel framework designed to unify the attack process by simultaneously targeting both the retriever and the generator.

Inspired by the success of Greedy Coordinate Gradient (GCG) techniques [24] in manipulating generator outputs through gradient-guided input optimization, **Joint-GCG** conceptualizes the RAG system as a single, integrated target for the poisoning attack, as illustrated in Figure 2.

A core challenge in this unified approach lies in reconciling the different architectures, tokenization schemes, and vocabularies between the retriever and generator models. Our framework addresses these complexities through three key innovations: (1) *Cross-Vocabulary Projection (CVP)*, which aligns the embedding spaces of disparate vocabularies; (2) *Gradient Tokenization Alignment (GTA)*, which synchronizes token-level gradient signals across different tokenization outputs; and (3) *Adaptive Weighted Fusion (AWF)*, which dynamically balances the attacking objectives for the retriever and generator. Together, these components enable a cohesive, gradient-based optimization strategy that jointly steers both the retrieval of the poisoned document and the subsequent generation of the desired malicious output.

## 4.1. Cross-Vocabulary Projection (CVP): Bridging Vocabulary Discrepancies

The inherent vocabulary mismatch between retrievers and generators poses a significant challenge to joint gradient optimization. Typically, the retriever's vocabulary differs from the generator's vocabulary due to their distinct training corpora. CVP addresses this gap by utilizing a combination of the generator tokens to represent the retriever tokens in the embedding space.

During the GCG-like attacks, an optimizable sequence was employed to steer the model output with gradient guidance. Let $N_{gen}, V_{gen}$ represent the optimizable sequence length and vocabulary size of the generator, and $N_{ret}, V_{ret}$ for

the retriever. During the attack process, we compute gradient matrices $G_{gen} \in \mathbb{R}^{N_{gen} \times V_{gen}}$ and $G_{ret} \in \mathbb{R}^{N_{ret} \times V_{ret}}$.

Directly obtaining a high-dimensional linear transformation matrix $W \in \mathbb{R}^{V_{\text{ret}} \times V_{\text{gen}}}$ using generator tokens to represent retriever tokens is rendered infeasible by both the ill-posed nature of constructing a sufficient system of equations and the substantial computational demands. Instead, CVP adopts a more tractable approach by focusing on mapping individual token embeddings. Let $E_{\text{gen}} \in \mathbb{R}^{V_{\text{gen}} \times D_{\text{gen}}}$ and $E_{\text{ret}} \in \mathbb{R}^{V_{\text{ret}} \times D_{\text{ret}}}$ represent the embedding matrices of the generator and the retriever, respectively, where $D$ denotes the embedding dimension. For each retriever token embedding $y \in \mathbb{R}^{D_{\text{ret}}}$, our objective is to find a representation $x \in \mathbb{R}^{V_{\text{gen}}}$ such that it acts as a linear combination with the generator tokens when transformed by a learned embedding mapping function $f$, closely approximates $y$:

$$f(xE_{\text{gen}}) = y \tag{1}$$

Here, $f : \mathbb{R}^{D_{\text{gen}}} \rightarrow \mathbb{R}^{D_{\text{ret}}}$ represents a mapping function between the two embedding spaces.

CVP learns $f$ using shared tokens between the generator and retriever vocabularies, mapping their respective embeddings through an *autoencoder*. The autoencoder is trained to project the generator embedding of a shared token and then reconstruct the corresponding retriever embedding. This objective enables the autoencoder to learn a robust and semantically meaningful mapping that generalizes beyond the shared vocabulary, effectively capturing the underlying relationships between the two embedding spaces. The encoder of the trained autoencoder functions as $f$, enabling us to solve a least-squares solution that transforms $G_{ret}$ into $G'_{ret} \in \mathbb{R}^{N_{ret} \times V_{gen}}$ and aligning the gradient information. Detailed architectural specifications of the autoencoder, training procedures, and analysis of CVP are provided in Appendix A.

## 4.2. Gradient Tokenization Alignment (GTA): Synchronizing Tokenization Granularity

After CVP, another critical challenge arises from differing tokenization schemes. Retrievers and generators often employ distinct tokenizers, resulting in differences in how the input text is segmented into tokens. Consequently, the optimizable sequences for the retriever and generator may be tokenized into different sequences. This tokenization mismatch hinders direct gradient fusion. To address this, we propose *GTA*, a module designed to synchronize gradient signals at a finer, tokenizer-agnostic granularity.

To achieve tokenization alignment, GTA employs character-level gradients as an intermediary. Retriever token gradients, derived from $G'_{ret}$, are decomposed and assigned to their constituent characters, recognizing characters as a more fundamental and tokenization-agnostic text unit. Subsequently, retriever token gradients are constructed by averaging the gradients of the characters forming each generator token. This averaging method robustly consolidates character-level information back into the generator's token space while mitigating potential noise from the decomposition process.

Through the GTA process, we obtain a transformed retriever gradient matrix $G''_{ret} \in \mathbb{R}^{N_{gen} \times V_{gen}}$. Crucially, $G''_{ret}$ is now aligned with the generator's gradient matrix $G_{gen}$ in both vocabulary and sequence length dimensions. This alignment establishes a common gradient space, enabling a meaningful and direct fusion of gradient information from the retriever and the generator for joint optimization.

Additionally, we provide the pseudo-code of GTA in Appendix B.

## 4.3. Adaptive Weighted Fusion (AWF): Dynamically Balancing Retrieval and Generation Objectives

With $G_{gen}$ and $G''_{ret}$ now aligned in all dimensions, the final critical step is determining how to combine these gradient matrices to achieve joint optimization effectively. We propose *AWF*, a module that dynamically adjusts the relative contribution of each gradient matrix during the optimization process. This adaptive weighting mechanism is essential because the optimal balance between prioritizing retrieval and generation objectives can vary significantly depending on the specific attack scenario, the RAG system settings, and the characteristics of the target query and the corpus.

AWF fuses the gradients and weights the losses using a weighted sum controlled by an adaptive weighting parameter $\alpha$:

$$G_{joint} = (1 - \alpha)G_{gen} + \alpha G''_{ret} \qquad (2)$$

The weighting factor $\alpha \in [0, 1]$ governs the relative influence of the retriever's and generator's gradient in the joint gradient update.

As detailed in Appendix D, our experiments indicate a strong correlation between the retrieval rank of a poisoned document and attack success in RAG systems. We observed that poisoned documents retrieved at higher ranks are generally more influential in shaping the generator's response. To

optimize attack performance, it is desirable to position the poisoned document as high as possible in the retrieval ranking while ensuring a sufficient margin from other documents to mitigate potential rank fluctuations during subsequent optimization steps.

AWF introduces a stability metric, $D_{stability}$, quantifying the robustness of the poisoned document's retrieval rank:

$$D_{stability} = \frac{S_{doc_p} - S_{doc_0}}{D_{avg}} \qquad (3)$$

where $S_{doc_p}$ and $S_{doc_0}$ are the similarity scores of the query with the poisoned document and the benign document with the highest rank, respectively, $D_{avg}$ is the average similarity score difference between consecutive $k$ documents, defined as below:

$$D_{avg} = \frac{1}{k-1} \sum_{i}^{k-1} S_{doc_i} - S_{doc_{i+1}} \qquad (4)$$

The adaptive weighting parameter $\alpha$ is then dynamically determined based on $D_{stability}$ using a Sigmoid function:

$$\alpha = \sigma(D_{stability}) = \frac{1}{1 + e^{-D_{stability}}} \qquad (5)$$

The Sigmoid function, $\sigma(\cdot)$, ensures that $\alpha$ remains bounded within the range $[0, 1]$, controlling the weight of the retriever during the joint optimization.

This dynamic adjustment mechanism empowers Joint-GCG to adaptively balance the optimization of retrieval and generation objectives, resulting in more potent and robust poisoning attacks across diverse RAG system configurations.

## 5. Experiments

### 5.1. Experimental Setup

A comprehensive experimental evaluation was conducted to rigorously assess the effectiveness of our Joint-GCG framework, encompassing both targeted and batch query poisoning scenarios. We evaluated Joint-GCG's performance against state-of-the-art baselines – PoisonedRAG[11] (with GCG[24] on the generator), LIAR[13], and Phantom[15] – under diverse conditions. This thorough evaluation included experiments focusing on targeted query poisoning, ablation studies, synthetic corpora evaluations, black-box transferability, batch query poisoning, and extended attacking steps (Appendix E). We also investigated the impact of common defensive mechanisms on Joint-GCG (Section 5.3).

**5.1.1. Datasets.** Following prior works in RAG, we evaluate our approach using three widely used open-domain question-answering (QA) datasets. These datasets are designed to test various aspects of retrieval and reasoning in QA models, ensuring comprehensive evaluation across different question types and retrieval challenges.

- **MS MARCO[34]:** The Microsoft Machine Reading Comprehension (MS MARCO) dataset is a large-scale benchmark for information retrieval and question answering. It consists of real-world queries sampled from Bing's search logs, with passages extracted from web documents as candidate answers. Our experiments use some subsets of the queries determined by the baselines we compare.
- **Natural Questions (NQ)[35]:** This dataset consists of naturally occurring questions posed by users in Google Search, with human-annotated answers extracted from Wikipedia articles. Unlike MS MARCO, which focuses on search engine queries, NQ emphasizes the extraction of factual knowledge from structured sources. The dataset is particularly useful for evaluating a system's ability to retrieve and extract concise answers from a large-scale knowledge base.
- **HotpotQA[36]:** A multi-hop question-answering dataset that requires reasoning over multiple documents to arrive at a correct answer. Unlike single-hop QA datasets, where the answer is typically found within a single passage, HotpotQA demands integrating information across different documents, making it an excellent benchmark for assessing the model's capability to handle complex reasoning tasks.

For each dataset (MS MARCO, NQ, HotpotQA), we instructed GPT-4o-mini to generate a synthetic corpus of documents relevant to the queries in the respective dataset. We generated a corpus of 10 synthetic documents for each target query. While not representing real-world knowledge, these synthetic corpora serve as a proxy retrieval environment where we can simulate retrieval rankings and calculate the stability metric for AWF. The prompt we used for generating the synthetic corpus is provided below:

> "
> You are a creative assistant. Given the query: '{query}', whose correct answer is: '{correct_answer}', please generate 11 diverse and closely related sentences or short paragraphs. Each corpus should be distinct and cover different aspects related to the query. Format each corpus as a separate bullet point starting with -. Please avoid any other markdown or formatting.
> "

We use the same samples as those in prior works to ensure a fair comparison.

The dataset used for comparison with PoisonedRAG and LIAR in Experiments 5.2.1, 5.2.2, 5.2.3, and 5.2.5 consists of 100 queries, each sampled by PoisonedRAG from the MS MARCO, NQ, and HotpotQA datasets.

For comparison with Phantom in Experiment 5.2.4, we used the same dataset of 25 queries for each trigger. Each query was sampled by Phantom from the MS MARCO dataset and was selected to contain a particular trigger word.

**5.1.2. Retriever Models.** We experiment with two widely used dense retrieval models to evaluate the effectiveness of our Joint-GCG framework in different retrieval mechanisms. These models utilize neural embeddings to encode queries and documents into a shared vector space, facilitating efficient retrieval through similarity search.

- **Contriever[37]:** A contrastive learning based retrieval model designed for unsupervised sentence embeddings. Contriever is trained using a contrastive loss, which encourages similar text pairs to have closer embeddings while pushing apart unrelated pairs. This approach has demonstrated strong performance in retrieval tasks, particularly in zero-shot and low-resource settings, making it a robust choice for open-domain QA scenarios. We utilize the contriever-msmarco model here.
- **BGE (BAAI General Embedding)[38]:** A family of embedding models developed by the Beijing Academy of Artificial Intelligence (BAAI), optimized explicitly for retrieval tasks. BGE models are trained using large-scale datasets designed to produce highly efficient representations, facilitating fast and accurate retrieval in Approximate Nearest Neighbor (ANN) search. Their effectiveness in dense retrieval tasks makes them a competitive alternative to traditional retrieval methods. We utilize the BGE-base-en-v1.5 model here.

**5.1.3. Generator Models.** To evaluate the impact of our attack on state-of-the-art generator models, we conduct experiments on multiple LLMs. These models are selected based on their strong performance in various NLP tasks, open-source availability, and widespread use in research and applications.

- **Llama3-8B[39]:** A cutting-edge open-source LLM developed by Meta AI featuring 8 billion parameters. As a successor to the highly successful Llama and Llama 2 models, Llama3 is designed to offer superior reasoning, comprehension, and response generation capabilities. Its accessibility and state-of-the-art performance make it a strong candidate for evaluating adversarial robustness in RAG scenarios.
- **Qwen2-7B[40]:** A 7-billion-parameter model developed by Alibaba Cloud as part of the Qwen series. Qwen2 is known for its strong multilingual capabilities and efficient inference, making it a competitive choice for real-world applications. Its training methodology emphasizes knowledge-rich responses, which makes it particularly interesting to evaluate how retrieval-augmented attacks influence factual generation and reasoning.

**5.1.4. Metrics.** We use the following metrics to evaluate the effectiveness of the poisoning attacks:

- **Retrieval Attack Success Rate** ($ASR_{ret}$)**:** The percentage of target queries for which the poisoned document is retrieved within the top-$k$ results.

- **Generation Attack Success Rate** ($ASR_{gen}$)**:** The percentage of target queries for which the LLM generates the desired target output, i.e., the generated output contains the target. We use this approach to align with PoisonedRAG[11], as it shows negligible differences from human evaluation.
- **Position of Poisoned Document** ($Pos_p$)**:** The average rank ($1 \leq Pos_p \leq k$) of the poisoned document in the retrieval results for the target queries. Lower values indicate a stronger positioning of the poisoned document.

**5.1.5. Experimental Settings.** To ensure reproducibility and reduce randomness, all experiments were repeated three times, with greedy decoding for local LLMs and GPT-4o's temperature set to 0. Poisoned documents were initialized based on PoisonedRAG's attack scheme (query concatenation), upon which we added an optimizable sequence. For all experiments, we retrieve the top-5 related documents from the corpus and follow the chat template from the corresponding baselines for generation. All experiments are conducted on machines with 256GB of RAM and one NVIDIA RTX A6000 GPU.

All experiments were conducted under identical conditions to ensure a rigorous and fair comparison against baseline methods. Crucially, when comparing **Joint-GCG** with existing approaches such as LIAR and Phantom, we maintained an equivalent level of white-box access to all model components (retrievers and generators). Furthermore, key experimental parameters, including the number of optimization steps, optimizable sequence lengths, and dataset samples, were kept consistent across all compared methods.

In experiments comparing Joint-GCG to PoisonedRAG with GCG and LIAR (Experiments 5.2.1, 5.2.2, 5.2.3, and 5.2.5), we set the optimizable sequence length to 32 tokens and the optimization steps to 64. We employed a variant of GCG, MCG[15], to enhance the attack efficiency and utilized its default hyperparameters. We expressly set the Batch Size to 128 and TopK to 16, used ASCII-character-only tokens for attacks, and configured the optimization target to the incorrect answer. For the LIAR method, we used a 1:1 ratio for the optimizable sequence length (16 tokens each for the retriever and generator) and optimization step count (8 steps each for the retriever and generator), attacking the retriever first, followed by the generator.

In experiment 5.2.4, we employed the same $S_{cmd}$ prescribed in Phantom for Denial-of-Service attacks and concatenated the optimizable sequence on it. Due to the time-consuming nature of the computation, we performed only one round of experiments. We set the optimizable sequence to 128 and the optimization step counts to 32. For Phantom, we followed their settings, using a retriever-optimizable sequence length of 128 and a generator-optimizable sequence length of 8. We perform 256 steps of GCG for the 128 retriever tokens in advance for both methods to ensure a robust retrieval rate on batch queries, facilitating further optimization of the generator-optimizable tokens.

To compare Joint-GCG with PoisonedRAG and LIAR, we used the same data as PoisonedRAG's black-box retriever approach, using their first generated fake corpus[1] of towards the corresponding dataset in Experiment 5.2.1, 5.2.2, 5.2.3, and 5.2.5. We add an optimizable sequence at the beginning of the fake corpus, initialized with "!".

For comparisons with Phantom (Experiment 5.2.4), the documents consist of three parts concatenated together as prescribed in their work, $S_{ret}$, $S_{gen}$, and $S_{cmd}$, respectively. We first initialized $S_{ret}$ with "?" and $S_{gen}$ with "!".

## 5.2. Experiment Results

**5.2.1. Targeted Query Poisoning: Baseline Comparison.** To evaluate the effectiveness of Joint-GCG in targeted query poisoning, we compare it against state-of-the-art poisoning methods, specifically PoisonedRAG with GCG on generator (denoted as GCG below) and LIAR. We inject one poisoned document per target query and measure the effectiveness of the attack at retrieval and generation stages for the target queries using $ASR_{ret}$, $ASR_{gen}$, and $Pos_p$.

As shown in Table 1, Joint-GCG consistently outperforms GCG and LIAR regarding $ASR_{ret}$ and $ASR_{gen}$ across various settings. Specifically, Joint-GCG successfully maintains near-perfect $ASR_{ret}$ (100%) for Llama3 and Qwen2 across all datasets. Joint-GCG significantly surpasses GCG and LIAR in attacking the generation phase, particularly on NQ and HotpotQA datasets. For instance, Joint-GCG yields up to 99.0% $ASR_{gen}$ for Llama3 and 95.8% for Qwen2 on HotpotQA, outperforming GCG and LIAR by several percentage points.

Also, Joint-GCG has better efficacy (i.e., achieves higher $ASR_{gen}$ at fewer optimization steps). Figure 3 visually confirms Joint-GCG's superior efficacy, demonstrating that it reaches comparable or higher $ASR_{gen}$ than GCG and LIAR while requiring significantly fewer optimization steps. This faster convergence underscores Joint-GCG's enhanced efficiency and effectiveness in targeted query poisoning attacks.

Joint-GCG's performance gains stem from its innovative approach of integrating the retriever and generator gradients. This effectively prevents retrieval degradation and ensures the efficacy and success of poisoning throughout the optimization process. The consistently higher index ranking of the poisoned corpus highlights Joint-GCG's strengths as a leading method for targeted query poisoning, especially in adversarial contexts where high document rankings are essential for effective generator manipulation.

**5.2.2. Ablation Study.** To analyze the contribution of each component in Joint-GCG, we conduct an ablation study, removing key modules and evaluating their impact on attack performance.

      **Effect of Cross-Vocabulary Projection (CVP) and Gradient Tokenization Alignment (GTA)**. We conducted an ablation study to assess the impact of the CVP and GTA

---

1. Available at PoisonedRAG's official GitHub Repository.

TABLE 1. $ASR$ AND MEAN $pos_p$ OF GCG, LIAR, AND JOINT-GCG AT 64 OPTIMIZATION STEPS. VALUES IN PARENTHESES ($ASR_{gen}$) REPRESENT THE ASR SPECIFICALLY ON QUERIES WHERE INITIAL (UNOPTIMIZED) ATTACKS FAILED, DEMONSTRATING THE EFFECTIVENESS OF OPTIMIZATION.

| Retriever | Metrics | Dataset | MS MARCO | | NQ | | HotpotQA | |
|---|---|---|---|---|---|---|---|---|
| | | Attack / LLM | Llama3 | Qwen2 | Llama3 | Qwen2 | Llama3 | Qwen2 |
| Contriever | $ASR_{ret}$ | GCG | 96.00% | 95.67% | 72.00% | 72.00% | 94.33% | 97.00% |
| | | LIAR | **100.00%** | **100.00%** | 93.33% | 96.33% | 99.00% | **100.00%** |
| | | Joint-GCG | **100.00%** | **100.00%** | **99.00%** | **99.00%** | **100.00%** | **100.00%** |
| | $ASR_{gen}$ | GCG | 90.0% (76.7%) | 91.0% (80.0%) | 72.0% (41.5%) | 70.0% (39.0%) | 90.3% (76.7%) | 97.0% (87.5%) |
| | | LIAR | 89.0% (74.4%) | 95.3% (88.9%) | 89.0% (73.2%) | 86.0% (68.3%) | 92.0% (81.4%) | 98.0% (91.7%) |
| | | Joint-GCG | **94.0% (86.0%)** | **96.3% (91.1%)** | **92.0% (82.9%)** | **95.0% (87.8%)** | **97.3% (93.0%)** | **99.0% (95.8%)** |
| | | w/o optimize | 51.0% | 49.0% | 50.0% | 34.0% | 59.0% | 60.0% |
| | $Pos_p \downarrow$ | GCG | 1.36 | 1.43 | 2.59 | 2.56 | 1.46 | 1.2 |
| | | LIAR | 1.13 | 1.08 | 1.52 | 1.43 | 1.14 | 1.06 |
| | | Joint-GCG | **1.01** | **1.05** | **1.25** | **1.22** | **1.04** | **1.01** |
| BGE | $ASR_{ret}$ | GCG | 74.00% | 73.30% | 96.00% | 98.67% | 100.00% | 100.00% |
| | | LIAR | **99.00%** | 97.30% | **100.00%** | **100.00%** | 100.00% | 100.00% |
| | | Joint-GCG | **99.00%** | **99.00%** | **100.00%** | **100.00%** | 100.00% | 100.00% |
| | $ASR_{gen}$ | GCG | 68.0% (60.7%) | 67.0% (57.1%) | **93.0% (89.1%)** | 97.0% (95.5%) | 98.0% (95.9%) | **99.0% (97.4%)** |
| | | LIAR | 83.7% (78.6%) | **92.0% (85.7%)** | 89.3% (80.0%) | 93.0% (86.4%) | 93.7% (85.7%) | 96.0% (89.5%) |
| | | Joint-GCG | **87.0% (85.7%)** | **92.0% (85.7%)** | **93.0% (87.3%)** | **97.7% (95.5%)** | **99.0% (98.0%)** | **99.0% (97.4%)** |
| | | w/o optimize | 31.0% | 27.0% | 39.0% | 31.0% | 46.0% | 46.0% |
| | $Pos_p \downarrow$ | GCG | 2.87 | 3.02 | 1.36 | 1.23 | 1.04 | 1.01 |
| | | LIAR | 1.5 | 1.69 | **1.04** | **1.07** | **1.01** | 1.01 |
| | | Joint-GCG | **1.38** | **1.47** | 1.06 | **1.07** | **1.01** | 1.01 |

modules. As they are interdependent, they were removed simultaneously. In this configuration, the optimization within our framework relies solely on generator-side gradients, as direct fusion is infeasible due to inherent mismatches in gradient shapes. The baseline is thus a variant using only generator gradients within the Joint-GCG structure.

As demonstrated in Table 2, removing CVP and GTA resulted in a modest but consistent decrease in ASR. While seemingly subtle, achieving additional gains over an already effective generator-only optimization strategy is inherently challenging. This 2% absolute improvement on average represents several additional successful poisonings in the test sets. It signifies a significant enhancement in attack potency, particularly in security-critical scenarios where every successful compromise is crucial.

TABLE 2. $ASR_{gen}$ WITH CVP AND GTA REMOVED AND $ASR_{gen}$ WITH $Loss_{gen}$ ONLY ACROSS DATASETS AND GENERATORS, USING CONTRIEVER AS THE RETRIEVER.

| Dataset | Settings | Llama3 | Qwen2 |
|---|---|---|---|
| MS MARCO | Full Joint-GCG | **94.00%** | **96.33%** |
| | w/o CVP + GTA | 93.33% | 96.00% |
| | w/o $Loss_{ret}$ | 91.00% | 92.33% |
| | Base (GCG) | 90.00% | 91.00% |
| NQ | Full Joint-GCG | **92.00%** | **95.00%** |
| | w/o CVP + GTA | 91.00% | 93.00% |
| | w/o $Loss_{ret}$ | 86.67% | 94.00% |
| | Base (GCG) | 72.00% | 70.00% |
| HotpotQA | Full Joint-GCG | **97.33%** | **99.00%** |
| | w/o CVP + GTA | 95.00% | **99.00%** |
| | w/o $Loss_{ret}$ | 91.33% | 98.67% |
| | Base (GCG) | 90.00% | 97.00% |

**Effect of Retriever-Side Loss**. To further investigate the contribution of the retriever component, we conducted experiments by removing the retriever-side loss ($Loss_{ret}$) when selecting the best candidates.

As presented in Table 2, the removal of $Loss_{ret}$ led to a pronounced decrease in $ASR_{gen}$ across all datasets and generators. Specifically, for MS MARCO, $ASR_{gen}$ decreased from 94.00% to 91.00% for Llama3 and from 96.33% to 92.33% for Qwen2. Similar trends were observed for NQ, with the most significant impact on HotpotQA for Llama3, where $ASR_{gen}$ decreased from 97.33% to 91.33%. These results underscore the crucial role of the retriever-side loss in guiding the optimization process toward more potent poisoned documents.

**Effect of Adaptive Weighted Fusion (AWF)**. As shown in Figure 4, we experiment with different fixed retrieval-generation gradient weights and find that the AWF leads to the best performance. Significantly, AWF demonstrates superior or comparable performance to all fixed ratios in terms of $ASR_{ret}$. This improvement highlights AWF's adaptive approach, effectively balancing retriever and generator optimization while dynamically adjusting the retriever weight to prevent degradation as needed.

**5.2.3. Synthetic Corpus: Removing Top-k Retrieval Dependency.** One practical challenge in implementing Joint-GCG in real-world scenarios is accessing the top-$k$ retrieval results to attack the generator and calculating the stability metric $D_{stability}$ for AWF. This requires querying the actual retriever during an attack, which might be restricted in specific settings. To address this limitation and explore the potential for a more self-contained attack strategy, we investigated the use of *synthetic corpus* generated by LLM
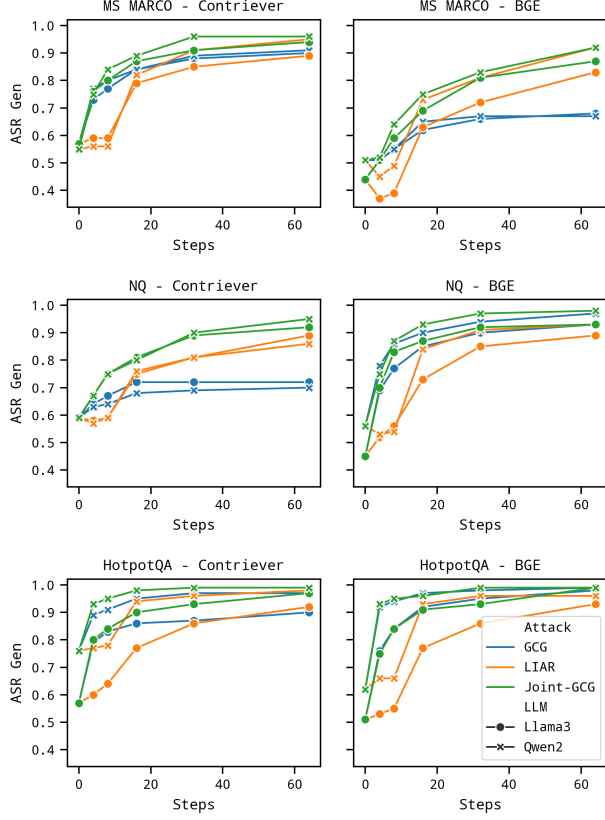
Figure 3. $ASR_{gen}$ of GCG, LIAR, and Joint-GCG at various optimization steps.

TABLE 3. $ASR$s ON MS MARCO DATASET WITH VARIOUS RETRIEVERS AND GENERATORS, COMPARING JOINT-GCG WITH REAL TOP-k RETRIEVAL AND SYNTHETIC CORPUS-BASED AWF.

| Retriever | Metrics | Settings | Llama3 | Qwen2 |
|---|---|---|---|---|
| Contriever | $ASR_{ret}$ | Real | 100.00% | 100.00% |
| | | Synthetic | 100.00% | 100.00% |
| | | w/o optimize | 98.00% | 98.00% |
| | $ASR_{gen}$ | Real | 94.00% | 96.33% |
| | | Synthetic | 62.00% | 58.00% |
| | | w/o optimize | 51.00% | 49.00% |
| BGE | $ASR_{ret}$ | Real | 99.00% | 99.00% |
| | | Synthetic | 89.33% | 84.00% |
| | | w/o optimize | 70.00% | 70.00% |
| | $ASR_{gen}$ | Real | 87.00% | 92.00% |
| | | Synthetic | 41.00% | 36.67% |
| | | w/o optimize | 31.00% | 27.00% |

to simulate the retrieval environment and enable AWF calculation without relying on real-time top-$k$ retrieval.

Table 3 reveals that Joint-GCG can effectively utilize a synthetic corpus, substantially reducing the dependency on top-k retrieval results. While some performance variance is present compared to actual corpus data, the $ASR_{gen}$ achieved using the synthetic corpus remains commendable.
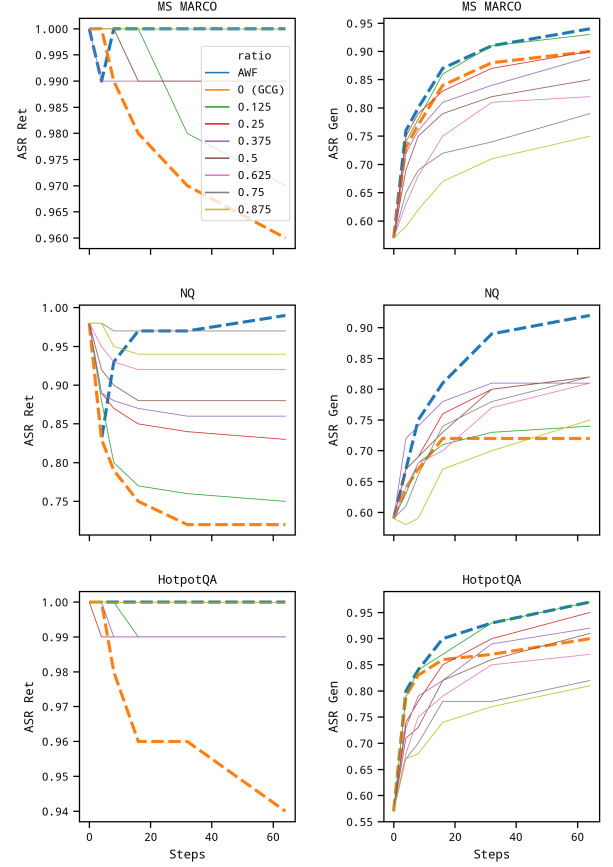


Figure 4. $ASR$ of Joint-GCG with various fixed weights and AWF, using Contriver as the retriever and Llama3 as the generator.

For instance, even with the synthetic corpus, Joint-GCG reaches 62% $ASR_{gen}$ with Contriever and Llama3, demonstrating a solid attack capability. This outcome underscores the practicality of Joint-GCG, enabling its application in scenarios where acquiring real-time retrieval information is challenging.

**5.2.4. Extending to Batch Query Poisoning.** Batch query poisoning is a more challenging scenario where a single poisoned document aims to manipulate the RAG system's response for multiple distinct target queries. We evaluate the performance of Joint-GCG in this setting, comparing it to Phantom, a method designed explicitly for trigger-based batch poisoning. We perform the attack following their prescribed Denial-of-Service (DoS) settings. We use the mean gradient and loss on the target queries to guide optimization.

Table 4 showcases the results of this experiment. Joint-GCG consistently outperforms Phantom across all tested trigger keywords and optimization steps. Joint-GCG achieves significantly higher $ASR_{gen}$ values at earlier optimization steps and plateaus at a higher success rate. For instance, on the "xbox" trigger, Joint-GCG reaches a high $ASR_{gen}$ of

TABLE 4. $ASR_{gen}$ OF JOINT-GCG FOR BATCH POISONING ON THREE TRIGGERS, USING LLAMA3 AS THE GENERATOR AND CONTRIEVER AS THE RETRIEVER, WITH THE TARGET OF DENIAL-OF-SERVICE. VALUES IN PARENTHESES ($ASR_{gen}$) REPRESENT THE ASR SPECIFICALLY ON QUERIES WHERE INITIAL (UNOPTIMIZED) ATTACKS FAILED, DEMONSTRATING THE EFFECTIVENESS OF OPTIMIZATION.

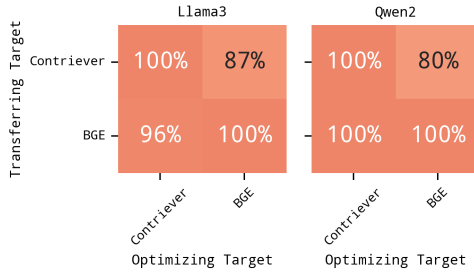| Trigger | Attack / Step | 0 | 4 | 8 | 16 | 32 |
|---------|---------------|---|---|---|----|----|
| amazon | Phantom | 76.00% | 76.00% (16.67%) | 76.00% (33.33%) | 68.00% (16.67%) | 80.00% (33.33%) |
| | Joint-GCG | 76.00% | **88.00% (50.00%)** | **88.00% (50.00%)** | **88.00% (50.00%)** | **88.00% (50.00%)** |
| xbox | Phantom | 80.00% | 80.00% (0.00%) | 84.00% (20.00%) | 84.00% (40.00%) | 84.00% (40.00%) |
| | Joint-GCG | 80.00% | **92.00% (60.00%)** | **92.00% (60.00%)** | **92.00% (60.00%)** | **92.00% (60.00%)** |
| iphone | Phantom | 88.00% | 84.00% (0.00%) | 88.00% (0.00%) | 88.00% (33.33%) | 88.00% (33.33%) |
| | Joint-GCG | 88.00% | **92.00% (100.00%)** | **92.00% (100.00%)** | **92.00% (100.00%)** | **96.00% (100.00%)** |



Figure 5. $ASR_{Ret}$ when poisons are optimized on various Retrievers are evaluated on Transferring Target Retriever with the Llama3 / Qwen2 generator, on the MS MARCO dataset.
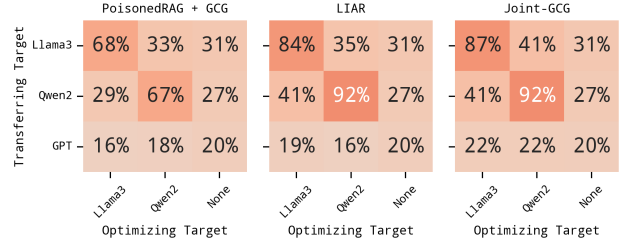


Figure 6. $ASR_{gen}$ when poisons are optimized on various Generators are evaluated on Transferring Target Generators with the BGE retriever, on the MS MARCO dataset. 'None' in the optimizing targets represents the unoptimized initial samples.

92% at step 4, whereas Phantom plateaus at a lower 80%. These results demonstrate Joint-GCG's strong capability in batch query poisoning, achieving more effective and faster convergence to high attack success rates than Phantom. This highlights the versatility of Joint-GCG and its potential for broader applications in RAG system manipulation beyond targeted individual queries.

**5.2.5. Evaluating Poison Generalization.** A crucial aspect of assessing the practical threat posed by RAG poisoning attacks is evaluating the generalization capabilities of the generated poisons. If a poison crafted for a specific, known model (a "surrogate") proves effective against other unknown models, it validates a more realistic gray-box attack scenario. In such a scenario, an attacker would perform the computationally intensive joint optimization on an open-source model they control and then deploy the resulting poison against a target system whose exact components are unknown. We investigate two key dimensions of generalization: cross-retriever transferability and cross-generator transferability.

**Cross-Retriever Transferability**. To evaluate how poisons optimized for one retriever perform against another, we conducted experiments using Joint-GCG, where the generator model was fixed, and poisons were transferred between the Contriever and BGE retriever models. The results, illustrated in Figure 5, demonstrate notable transferability. Notably, the poisons exhibited strong cross-retriever transfer. For instance, with Llama3 as the fixed generator, poisons optimized for BGE and transferred to Contriever

maintained a high $ASR_{ret}$ of 96%, while poisons optimized for Contriever and transferred to BGE achieved an $ASR_{ret}$ of 87%. A similar pattern was observed with Qwen2 as the generator, where BGE-optimized poisons transferred to Contriever with an $ASR_{ret}$ of 100%, and Contriever-optimized poisons transferred to BGE with an $ASR_{ret}$ of 80%. These findings indicate that poisons crafted by Joint-GCG can effectively compromise different retriever models. The strong transferability, particularly from BGE-optimized poisons to Contriever, suggests that some learned adversarial features are robust across retriever architectures. This enhances the practical threat, as an attacker might achieve success even without exact knowledge of the deployed retriever.

**Cross-Generator Transferability**. We assessed the ability of poisoned documents optimized by PoisonedRAG with GCG, LIAR, and Joint-GCG to influence other generators, including open-source models (Llama3, Qwen2) and a black-box model (GPT-4o). The comparative results are presented in Figure 6.

Regarding transferability between the open-source models Llama3 and Qwen2, Joint-GCG showed consistent transfer: poisons optimized for Llama3 achieved an $ASR_{gen}$ of 41% on Qwen2, and poisons optimized for Qwen2 achieved an $ASR_{gen}$ of 41% on Llama3. LIAR exhibited similar transfer from Qwen2 to Llama3 (41%) but slightly less from Llama3 to Qwen2 (35%). PoisonedRAG with GCG displayed lower transferability in both directions (33% for Llama3 to Qwen2, and 29% for Qwen2 to Llama3). Significantly, this optimization-driven transferability extends even to black-box

commercial LLMs like GPT-4o[41]. While the absolute ASR against GPT-4o is lower, poisons optimized for Llama3 or Qwen2 still exhibit a noticeable increase (2%) in attack success compared to non-optimized poisons, unseen in previous attack methods. These findings underscore that RAG systems face a broader and more generalized attack surface than previously understood, as attackers can enhance cross-generator poison transferability through targeted optimization on readily available models.

## 5.3. Potential Defensive Mechanisms

Here, we report the outcomes of evaluating two widely used defensive strategies towards Joint-GCG: Perplexity-based filtering[42] and SmoothLLM[43].

**5.3.1. SmoothLLM.** SmoothLLM is a perturbation-based defense mechanism designed to counter adversarial attacks by injecting controlled noise into the input. This approach subtly alters the inputs to the LLMs, preserving the intended meaning while mitigating the effects of adversarial perturbations. In our experiments, we applied a swap permutation with a noise ratio of 5%, as SmoothLLM prescribed.

TABLE 5. $ASR_{gen}$ AND CORRECT ANSWER RATE($CAR$) WHEN NO ATTACK APPLIED WITH SMOOTHLLM DEPLOYED ON MS MARCO.

| Retriever | Generator | Llama3 | | Qwen2 | |
|---|---|---|---|---|---|
| | SmoothLLM | w/o | w/ | w/o | w/ |
| Contriver | $CAR$ (w/o attack) | 77% | 72% | 71% | 66% |
| | $ASR_{gen}$ | 94% | 53% | 96% | 56% |
| BGE | $CAR$ (w/o attack) | 87% | 85% | 85% | 82% |
| | $ASR_{gen}$ | 87% | 47% | 92% | 41% |

As demonstrated in Table 5, while SmoothLLM reduces Joint-GCG's $ASR_{gen}$, the attack remains alarmingly potent even under this defense. For Contriever-retriever systems, Joint-GCG achieves 53% $ASR_{gen}$ on Llama3 and 56% on Qwen2 with SmoothLLM enabled.

The persistent success of the attack under defenses highlights Joint-GCG's robustness. Even when attenuated by SmoothLLM, the attack success rates remain comparable to *undefended* performance of prior methods (e.g., GCG's around 70% $ASR_{gen}$ in Table 1). This demonstrates that Joint-GCG's joint optimization paradigm fundamentally enhances attack survivability, signifying the challenges for defensive mechanisms and necessitating new research into retrieval-aware adversarial filtering.

**5.3.2. Perplexity-Based Filtering.** Perplexity-based filtering involves using the perplexity score to assess the likelihood of the corpus and filtering out documents that exceed a certain threshold. This approach aims to reduce the noise introduced by less relevant or spurious information retrieved by the system. Specifically, we computed the perplexity of the MS MARCO corpus and the corresponding Joint-GCG optimized poisons on Llama3. As shown in Figure 7, the perplexity distribution of Joint-GCG optimized adversarial examples is
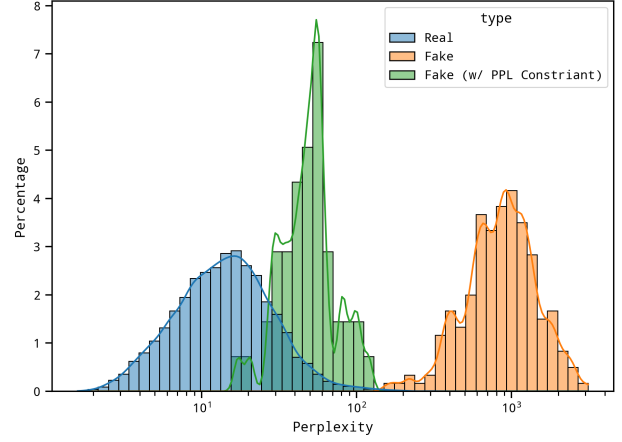


Figure 7. Perplexity percentage histogram of the real corpus in MS MARCO and fake corpus optimized by Joint-GCG.

TABLE 6. $ASR$ OF JOINT-GCG WITH PPL CONSTRAINT AT 32 OPTIMIZATION STEPS ON MS MARCO, USING CONTRIEVER AND QWEN2.

| Settings | $ASR_{Ret}$ | $ASR_{gen}$ |
|---|---|---|
| Joint-GCG w/ PPL Constraint | 100.00% | 73.33% |
| w/o optimize | 100.00% | 49.00% |

noticeably shifted towards higher perplexity values than the actual MS MARCO corpus. This suggests that perplexity could be a potential indicator for identifying and filtering out adversarial examples.

We implemented a constraint during the attack optimization process to investigate the effectiveness of perplexity-based filtering against Joint-GCG. Specifically, we incorporated a perplexity constraint to ensure that the generated adversarial examples remained within the perplexity distribution of the standard MS MARCO corpus by filtering out candidate adversarial examples exceeding a threshold during optimization.

Table 6 presents the attack success rates. Remarkably, even when optimized with a perplexity constraint, Joint-GCG maintains a significantly higher $ASR_{gen}$ than the baseline scenario where no optimization is performed, with a 73.33% for Joint-GCG with the perplexity constraint, compared to only 54.00% without optimization. This apparent increase demonstrates that perplexity-based filtering, in its simplest form, is inadequate against Joint-GCG. Even when adversarial examples are crafted to have perplexity values within the normal range, the attack remains potent and surpasses the baseline ASR by a large margin. This highlights the need for more advanced defenses to detect adversarial examples beyond simple perplexity thresholds.

## 6. Conclusion

We present **Joint-GCG**, a framework that elevates RAG poisoning via unified retrieval-generation gradient-based optimization. By harmonizing retrieval and generation objectives through Cross-Vocabulary Projection, Gradient Tokenization Alignment, and Adaptive Weighted Fusion, Joint-GCG overcomes the disjointed nature of prior attacks. Evaluations show at most 25% and an average of 5% higher attack success rates than state-of-the-art methods across multiple retrievers and generators, achieving greater $ASR$ within the same optimization steps. Ablations confirm the role of each component, while synthetic corpus tests and poison generalization experiments demonstrate the broad applicability. The framework's potency in batch poisoning further underscores its practical threat. Joint-GCG provides a robust framework for understanding and mitigating the evolving threat landscape of RAG-based applications.

## 7. Limitations and Future Works

While Joint-GCG demonstrates significant advances in RAG system poisoning, several important limitations and promising directions for future research warrant discussion:

### 7.1. Computational Overhead

The joint optimization of retrieval and generation gradients introduces additional computational complexity compared to existing methods (as detailed in Appendix F). Primarily, the CVP module involves an offline, one-time pre-computation for each retriever-generator pair (approximately 2 hours on a single NVIDIA A6000 GPU in our setup) to train an autoencoder and derive a projection matrix. This cost is amortized over the matrix's reuse for all subsequent attacks on that pair. While the primary additional burden is a manageable, offline pre-processing step, future work could explore more efficient optimization techniques or gradient approximation methods to reduce this overhead while maintaining the effectiveness of the attack.

### 7.2. Cross-Domain Generalization

While we demonstrate strong performance across multiple QA datasets, the generalization of Joint-GCG to other domains (e.g., code generation[44], medical applications[45], and tool-calling agents[46]) and different types of RAG architectures requires further investigation. Future work could explore domain-specific adaptations of the framework and evaluate its effectiveness across a broader range of applications and model architectures.

These limitations and future directions underscore the nascent nature of joint optimization attacks on RAG systems, emphasizing the importance of ongoing research in this critical area of AI security.

## 8. Ethical Considerations

The Joint-GCG framework, while advancing the understanding of RAG system vulnerabilities to improve security, presents potential misuse risks. We have prioritized responsible disclosure, striking a balance between scientific transparency and these concerns. Our research, conducted on controlled datasets and non-production systems, aims to proactively identify vulnerabilities, motivating the development of robust defenses and encouraging the design of security-first RAG systems. We strongly advocate for using these findings for security research and system improvement, not exploitation. Organizations deploying RAG systems should implement comprehensive security measures, including regular audits, content filtering, and continuous monitoring, as understanding these vulnerabilities is crucial to building more secure AI applications.

## References

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[2] O. Ram, Y. Levine, I. Dalmedigos, D. Muhlgay, A. Shashua, K. Leyton-Brown, and Y. Shoham, "In-context retrieval-augmented language models," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 1316–1331, 2023.

[3] OpenAI, "Introducing chatgpt search," https://openai.com/index/introducing-chatgpt-search, 2024, published: 2024-10-31.

[4] S. Vakayil, D. S. Juliet, S. Vakayil *et al.*, "Rag-based llm chatbot using llama-2," in *2024 7th International Conference on Devices, Circuits and Systems (ICDCS)*. IEEE, 2024, pp. 1–5.

[5] M. N. K. Boulos and R. Dellavalle, "Nvidia's "chat with rtx" custom large language model and personalized ai chatbot augments the value of electronic dermatology reference material," *JMIR dermatology*, vol. 7, no. 1, p. e58396, 2024.

[6] S. Zhou, U. Alon, F. F. Xu, Z. Wang, Z. Jiang, and G. Neubig, "Docprompting: Generating code by retrieving the docs," *arXiv preprint arXiv:2207.05987*, 2022.

[7] N. Nashid, M. Sintaha, and A. Mesbah, "Retrieval-based prompt selection for code-related few-shot learning," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 2450–2462.

[8] S. Siriwardhana, R. Weerasekera, E. Wen, T. Kaluarachchi, R. Rana, and S. Nanayakkara, "Improving the domain adaptation of retrieval augmented generation (rag) models for open domain question answering," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 1–17, 2023.

[9] K. Meduri, G. S. Nadella, H. Gonaygunta, M. H. Maturi, and F. Fatima, "Efficient rag framework for large-scale knowledge bases," 2024.

[10] W. Fan, Y. Ding, L. Ning, S. Wang, H. Li, D. Yin, T.-S. Chua, and Q. Li, "A survey on rag meeting llms: Towards retrieval-augmented large language models," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 6491–6501.

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

[12] J. Xue, M. Zheng, Y. Hu, F. Liu, X. Chen, and Q. Lou, "Badrag: Identifying vulnerabilities in retrieval augmented generation of large language models," *arXiv preprint arXiv:2406.00083*, 2024.

[13] Z. Tan, C. Zhao, R. Moraffah, Y. Li, S. Wang, J. Li, T. Chen, and H. Liu, "" glue pizza and eat rocks"–exploiting vulnerabilities in retrieval-augmented generative models," *arXiv preprint arXiv:2406.19417*, 2024.

[14] P. Cheng, Y. Ding, T. Ju, Z. Wu, W. Du, P. Yi, Z. Zhang, and G. Liu, "Trojanrag: Retrieval-augmented generation can be backdoor driver in large language models," *arXiv preprint arXiv:2405.13401*, 2024.

[15] H. Chaudhari, G. Severi, J. Abascal, M. Jagielski, C. A. Choquette-Choo, M. Nasr, C. Nita-Rotaru, and A. Oprea, "Phantom: General trigger attacks on retrieval augmented language generation," *arXiv preprint arXiv:2405.20485*, 2024.

[16] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2312.10997

[17] A. Asai, S. Min, Z. Zhong, and D. Chen, "Retrieval-based language models and applications," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 6: Tutorial Abstracts)*, 2023, pp. 41–46.

[18] L. Weidinger, J. Mellor, M. Rauh, C. Griffin, J. Uesato, P.-S. Huang, M. Cheng, M. Glaese, B. Balle, A. Kasirzadeh *et al.*, "Ethical and social risks of harm from language models," *arXiv preprint arXiv:2112.04359*, 2021.

[19] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "More than you've asked for: A comprehensive analysis of novel prompt injection threats to application-integrated large language models," *arXiv preprint arXiv:2302.12173*, vol. 27, 2023.

[20] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial attacks and defences: A survey," *arXiv preprint arXiv:1810.00069*, 2018.

[21] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, "Hotflip: White-box adversarial examples for text classification," *arXiv preprint arXiv:1712.06751*, 2017.

[22] J. Li, S. Ji, T. Du, B. Li, and T. Wang, "Textbugger: Generating adversarial text against real-world applications," *arXiv preprint arXiv:1812.05271*, 2018.

[23] L. Li, R. Ma, Q. Guo, X. Xue, and X. Qiu, "Bert-attack: Adversarial attack against bert using bert," *arXiv preprint arXiv:2004.09984*, 2020.

[24] A. Zou, Z. Wang, N. Carlini, M. Nasr, J. Z. Kolter, and M. Fredrikson, "Universal and transferable adversarial attacks on aligned language models," *arXiv preprint arXiv:2307.15043*, 2023.

[25] X. Jia, T. Pang, C. Du, Y. Huang, J. Gu, Y. Liu, X. Cao, and M. Lin, "Improved techniques for optimization-based jailbreaking on large language models," 2024.

[26] M. Goldblum, D. Tsipras, C. Xie, X. Chen, A. Schwarzschild, D. Song, A. Mądry, B. Li, and T. Goldstein, "Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 2, pp. 1563–1580, 2022.

[27] I. Shumailov, Z. Shumaylov, D. Kazhdan, Y. Zhao, N. Papernot, M. A. Erdogdu, and R. J. Anderson, "Manipulating sgd with data ordering attacks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 18 021–18 032, 2021.

[28] F. Liu and N. Shroff, "Data poisoning attacks on stochastic bandits," in *International Conference on Machine Learning*. PMLR, 2019, pp. 4042–4050.

[29] Z. Gyongyi and H. Garcia-Molina, "Web spam taxonomy," in *First international workshop on adversarial information retrieval on the web (AIRWeb 2005)*, 2005.

[30] J. Bevendorff, M. Wiegmann, M. Potthast, and B. Stein, "Is google getting worse? a longitudinal investigation of seo spam in search engines," in *European Conference on Information Retrieval*. Springer, 2024, pp. 56–71.

[31] S. Farooq, "A survey on adversarial information retrieval on the web," *arXiv preprint arXiv:1911.11060*, 2019.

[32] A. Mallen, A. Asai, V. Zhong, R. Das, D. Khashabi, and H. Hajishirzi, "When not to trust language models: Investigating effectiveness of parametric and non-parametric memories," *arXiv preprint arXiv:2212.10511*, 2022.

[33] Y. Zhang, Q. Li, T. Du, X. Zhang, X. Zhao, Z. Feng, and J. Yin, "Hijackrag: Hijacking attacks against retrieval-augmented large language models," *arXiv preprint arXiv:2410.22832*, 2024.

[34] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng, "Ms marco: A human-generated machine reading comprehension dataset," 2016.

[35] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee *et al.*, "Natural questions: a benchmark for question answering research," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 453–466, 2019.

[36] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. W. Cohen, R. Salakhutdinov, and C. D. Manning, "Hotpotqa: A dataset for diverse, explainable multi-hop question answering," *arXiv preprint arXiv:1809.09600*, 2018.

[37] G. Izacard, M. Caron, L. Hosseini, S. Riedel, P. Bojanowski, A. Joulin, and E. Grave, "Unsupervised dense information retrieval with contrastive learning," *arXiv preprint arXiv:2112.09118*, 2021.

[38] S. Xiao, Z. Liu, P. Zhang, N. Muennighoff, D. Lian, and J.-Y. Nie, "C-pack: Packed resources for general chinese embeddings," in *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, 2024, pp. 641–649.

[39] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[40] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang *et al.*, "Qwen2 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2407.10671

[41] OpenAI, "Hello gpt-4o," https://openai.com/index/hello-gpt-4o, 2024, published: 2024-05-13.

[42] G. Alon and M. Kamfonas, "Detecting language model attacks with perplexity," *arXiv preprint arXiv:2308.14132*, 2023.

[43] A. Robey, E. Wong, H. Hassani, and G. J. Pappas, "Smoothllm: Defending large language models against jailbreaking attacks," *arXiv preprint arXiv:2310.03684*, 2023.

[44] S. J. Rani, S. Deepika, D. Devdharshini, and H. Ravindran, "Augmenting code sequencing with retrieval-augmented generation (rag) for context-aware code synthesis," in *2024 First International Conference on Software, Systems and Information Technology (SSITCON)*. IEEE, 2024, pp. 1–7.

[45] C. Ye, "Exploring a learning-to-rank approach to enhance the retrieval augmented generation (rag)-based electronic medical records search engines," *Informatics and Health*, vol. 1, no. 2, pp. 93–99, 2024.

[46] H. Wang, R. Zhang, J. Wang, M. Li, Y. Huang, D. Wang, and Q. Wang, "From allies to adversaries: Manipulating llm tool-calling through adversarial injection," 2025. [Online]. Available: https://arxiv.org/abs/2412.10198

# Appendix A.
# Cross-Vocabulary Projection (CVP) Details

## A.1. Motivation & Overview

Cross-Vocabulary Projection (CVP) addresses the fundamental vocabulary mismatch between retrievers and generators in RAG systems. Since retrievers and generators are typically pre-trained independently, their tokenization schemes, vocabularies, and embedding spaces differ significantly. This discrepancy prevents direct gradient alignment between the two components. CVP bridges this gap by learning a joint embedding space through an autoencoder trained on shared tokens and deriving a linear transformation matrix to project retriever gradients into the generator's embedding space.

## A.2. Autoencoder Architecture & Training

The CVP autoencoder consists of an encoder-decoder pair with multiple ReLU-activated dense layers (Figure 8):
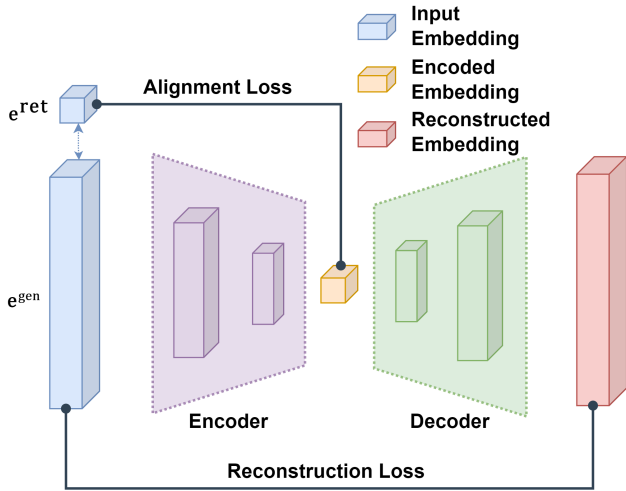


Figure 8. CVP autoencoder architecture. Token embeddings from the generator (LLM) are encoded into the retriever's space and decoded back.

**Encoder:** Maps the generator's embeddings ($\mathbb{R}^{D_{gen}}$) to the retriever's embedding space ($\mathbb{R}^{D_{ret}}$) via:

$$h_1 = \text{ReLU}(W_0 x + b_0) \tag{6}$$
$$h_2 = \text{ReLU}(W_1 h_1 + b_1) \tag{7}$$
$$\text{Enc}(e_{gen}) = W_2 h_2 + b_2 \tag{8}$$

where $e_{gen}$ is a generator token embedding, $W_0 \in \mathbb{R}^{2048 \times D_{gen}}$, $W_1 \in \mathbb{R}^{1024 \times 2048}$, and $W_2 \in \mathbb{R}^{D_{ret} \times 1024}$.

**Decoder:** Reconstructs the generator's embeddings from encoded retriever-space embeddings:

$$h_1' = \text{ReLU}(W_0' y + b_0') \tag{9}$$
$$h_2' = \text{ReLU}(W_1' h_1' + b_1') \tag{10}$$
$$\text{Dec}(e_{ret}) = W_2' h_2' + b_2' \tag{11}$$

where $e_{ret}$ is the corresponding retriever token embedding, $W_0' \in \mathbb{R}^{1024 \times D_{ret}}$, $W_1' \in \mathbb{R}^{2048 \times 1024}$, and $W_2' \in \mathbb{R}^{D_{gen} \times 2048}$.

We collect the shared tokens from both models and create a train-test split with an 80% training set and a 20% validation set. The model is trained on shared token pairs $(e^{gen}, e^{ret})$ in the training set using a composite loss:

$$\mathcal{L} = \alpha \mathcal{L}_{\text{rec}} + (1 - \alpha) \mathcal{L}_{\text{align}} \tag{12}$$

where:

$$\mathcal{L}_{\text{rec}} = \sum_i \|\text{Dec}(\text{Enc}(e_i^{gen})) - e_i^{gen}\|_2 \tag{13}$$

$$\mathcal{L}_{\text{align}} = \sum_i \|\text{Enc}(e_i^{gen}) - e_i^{ret}\|_2 \tag{14}$$

with $\alpha = 0.25$. We use AdamW with cosine annealing (initial learning rate $10^{-5}$) for 500 epochs at most, with an early stopping on validation loss.

## A.3. Transfer Matrix via Least Squares

After autoencoder training, we compute a linear projection matrix $W \in \mathbb{R}^{V_{ret} \times V_{gen}}$ to map retriever token gradients into the generator's gradients space as below:

1. **Encode Generator Embeddings:** We first compute the encoded embedding $\tilde{E}_{gen} \in \mathbb{R}^{V_{gen} \times D_{ret}}$ for the generator embedding $E_{gen} \in \mathbb{R}^{V_{gen} \times D_{gen}}$ to match embedding dimensions $D_{gen}$ and $D_{ret}$:

$$\tilde{E}_{gen} = \text{Enc}(E_{gen}) \tag{15}$$

2. **Solve Least Squares:** We employ the least squares method to find the optimal projection matrix between tokens. Specifically, for each retriever token embedding $y$, we find weights $W_i$ over generator tokens that minimize:

$$\underset{W_i}{\arg\min} \|W_i \tilde{E}_{gen} - y\|_2^2 \tag{16}$$

We solve the $W_i$ via PyTorch's `torch.lstsq`.

Finally, concatenating $W_i$ for each $y$ yields a projection matrix $W$ that maps retriever token influence into the generator's embedding space, enabling joint gradient optimization.

## A.4. Evaluation

To evaluate the effectiveness of the CVP autoencoder, we conducted a series of experiments on the validation set. We assessed the quality of the projected embeddings using the following metrics:

- **Projection Error**($Err_{proj}$): Measured by the mean Euclidean distance between the autoencoder's projected generator embedding and the ground truth retriever embedding on the validation set. A lower error indicates better projection accuracy.
- **Token Recall at Top-K** ($Recall@K$: We measured whether the corresponding ground truth retriever token embedding was found within the nearest neighbors in the retriever embedding space for each

projected generator embedding. We report Top-1, Top-3, Top-5, and Top-10 Recall. Higher recall values signify better preservation of semantic similarity after projection.

We compared the performance of the CVP autoencoder against baseline methods, including Linear Regression (LR), Random Forest (RF), and Multilayer Perceptron (MLP).

As shown in Table 7, the CVP autoencoder performs better on the validation set than all other methods. While Random Forest achieves the lowest Euclidean distance on the training set, the autoencoder generalizes better, achieving the lowest Projection Error on unseen data. Furthermore, the autoencoder significantly outperforms all Top-K Shared Token Recall baselines across all K values (1, 3, 5, and 10), reaching near-perfect recall even at Top-1 (97.95%). These results highlight the autoencoder's effectiveness in learning a robust and semantically meaningful projection, essential for bridging the vocabulary gap between retrievers and generators.

TABLE 7. PROJECTION ERROR AND TOKEN RECALL AT TOP-K PROJECTING LLAMA3 EMBEDDINGS TO CONTRIEVER EMBEDDINGS ON THE VALIDATION SET.

| Metrics | LR | RF | MLP | Autoencoder |
|---|---|---|---|---|
| $Distance_{train}$ ↓ | 0.7456 | **0.3894** | 1.051 | 0.7953 |
| $Distance_{test}$ ↓ | 0.9933 | 1.0455 | 1.0474 | **0.9528** |
| Recall @ 1 | 95.24% | 8.63% | 0.89% | **97.95%** |
| Recall @ 3 | 97.73% | 15.43% | 2.21% | **99.34%** |
| Recall @ 5 | 98.51% | 18.58% | 3.60% | **99.56%** |
| Recall @ 10 | 99.12% | 24.72% | 6.86% | **99.67%** |

# Appendix B.
# Pseudo-Code for Gradient Tokenization Alignment (GTA)

Algorithm 1 demonstrates the GTA process.

# Appendix C.
# Empirical Validation for Gradient Tokenization Alignment (GTA)

To empirically validate the design of our GTA module, we conducted an experiment by comparing its performance with that of a simpler alternative approach for aligning gradients between the retriever and generator. The simpler alternative proposed utilizes the generator's tokenizer and embeddings, projecting these embeddings into the retriever's embedding space using CVP, and then processing them through the retriever's layers to obtain gradients for optimizing the attack sequence.

## C.1. Experimental Setup

The experiment used the Contriever model as the retriever and the Llama3 model as the generator. For both the full

Joint-GCG framework (incorporating GTA) and the simpler alternative, we performed 64 optimization steps.

## C.2. Results and Discussion

The comparative results are presented in Table 8. Across all three datasets, the full Joint-GCG framework with GTA consistently outperformed the simpler alternative, particularly in terms of $ASR_{gen}$ and $Pos_p$.

TABLE 8. COMPARISON OF JOINT-GCG (WITH GTA) AND THE SIMPLER ALTERNATIVE FOR GRADIENT ALIGNMENT. RESULTS ARE AVERAGED OVER THREE RUNS AFTER 64 OPTIMIZATION STEPS.

| Dataset | Method | $ASR_{ret}$ (%) | $ASR_{gen}$ (%) | $Pos_p$ ↓ |
|---|---|---|---|---|
| MS MARCO | Alternative | 100.00% | 92.00% | 1.11 |
| | Joint-GCG (w/ GTA) | 100.00% | **94.00%** | **1.01** |
| NQ | Alternative | 97.00% | 81.00% | 1.62 |
| | Joint-GCG (w/ GTA) | **99.00%** | **92.00%** | **1.25** |
| HotpotQA | Alternative | 100.00% | 95.00% | 1.11 |
| | Joint-GCG (w/ GTA) | 100.00% | **97.00%** | **1.04** |

As shown in Table 8, while both methods achieved high $ASR_{ret}$, Joint-GCG with GTA demonstrated notably higher $ASR_{gen}$, especially on the NQ dataset (i.e., 92.00% vs 81.00%). Furthermore, Joint-GCG consistently achieved a better (lower) $Pos_p$, indicating a more effective retrieval manipulation that places the malicious document at a more prominent rank. For instance, on NQ, $Pos_p$ was 1.25 for Joint-GCG compared to 1.62 for the alternative method.

These results support our rationale for implementing GTA. By carefully aligning gradients at the tokenization level while respecting the native processing pipelines of both the retriever and generator, GTA facilitates a more accurate and effective joint optimization process. The simpler alternative, although conceptually straightforward, appears to suffer from the hypothesized representational mismatches, resulting in degraded attack performance. Thus, the empirical evidence underscores the necessity and superiority of the GTA component within the Joint-GCG framework.

# Appendix D.
# Effect of the Position of Poisoned Documents

As shown in Figure 9, $ASR_{gen}$ of the poisoned documents shows a clear ascending trend with higher rankings.

# Appendix E.
# Additional Steps for Baseline

To further investigate the efficacy of our Joint-GCG framework and the baseline methods, we conducted experiments that extended the optimization steps from 64 to 128 for LIAR, aligning the optimization steps for the generator. The results of these experiments are presented in Table 9.

Analysis of Table 9 in conjunction with the 64-step results (Table 1) reveals that while LIAR benefits from increased

TABLE 9. $ASR$ OF LIAR AT 128 OPTIMIZATION STEPS AND JOINT-GCG AT 64 OPTIMIZATION STEPS, USING CONTRIEVER AS THE RETRIEVER. VALUES IN PARENTHESES ($ASR_{gen}$) REPRESENT THE ASR SPECIFICALLY ON QUERIES WHERE INITIAL (UNOPTIMIZED) ATTACKS FAILED, DEMONSTRATING THE EFFECTIVENESS OF OPTIMIZATION.

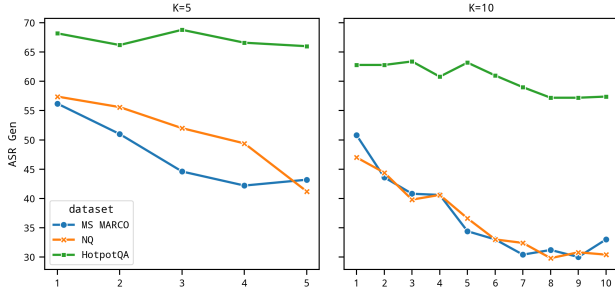| Metrics | Dataset | MS MARCO | | NQ | | HotpotQA | |
|---|---|---|---|---|---|---|---|
| | Attack | Llama3 | Qwen2 | Llama3 | Qwen2 | Llama3 | Qwen2 |
| $ASR_{ret}$ | LIAR | **100.00%** | 99.00% | 96.00% | **99.00%** | **100.00%** | **100.00%** |
| | Joint-GCG | **100.00%** | **100.00%** | **99.00%** | **99.00%** | **100.00%** | **100.00%** |
| $ASR_{gen}$ | LIAR | 93.0% (83.7%) | **96.0% (91.1%)** | **94.0% (85.4%)** | 93.0% (82.9%) | 95.0% (88.4%) | 98.0% (91.7%) |
| | Joint-GCG | **94.0% (86.0%)** | **96.0% (91.1%)** | 92.0% (82.9%) | **95.0% (87.8%)** | **97.0% (93.0%)** | **99.0% (95.8%)** |
| $Pos_p \downarrow$ | LIAR | 1.05 | 1.07 | 1.4 | 1.3 | 1.07 | **1.01** |
| | Joint-GCG | **1.01** | **1.02** | **1.23** | **1.16** | **1.02** | **1.01** |



Figure 9. The $ASR_{gen}$ when the poisoned document was positioned at various steps, with retrieval Top-K set to 5 and 10, respectively.

steps with modest improvements in $ASR_{gen}$ in specific scenarios (for instance, a 5% increase on NQ with Llama3 and a minor gain on MS MARCO with Qwen2), Joint-GCG remains superior. Joint-GCG achieves higher or comparable attack success rates with only 64 steps, demonstrating greater efficiency.

Joint-GCG also consistently outperforms LIAR at 128 steps in $ASR_{gen}$ across all evaluated datasets and generators. This demonstrates Joint-GCG's superior attack efficacy and its ability to strategically embed the poisoned document in the retrieved context for greater impact.

These findings from the 128-step LIAR experiments further solidify the advantages of our Joint-GCG framework. While increased optimization steps provide LIAR with some incremental gains in attack success, Joint-GCG maintains its lead in both $ASR_{gen}$ and $pos_p$, achieving superior performance with significantly fewer optimization steps. This highlights the efficiency and strategic poisoning capabilities inherent in Joint-GCG's joint optimization approach, demonstrating its ability to achieve high attack success with fewer optimization steps.

# Appendix F.
# Quantifying Computational Overhead

While Joint-GCG integrates optimization across both the retriever and generator, the introduced complexity is structured efficiently. The most computationally intensive new component, CVP, involves training an autoencoder and calculating a projection matrix. Crucially, this is performed only once as an offline pre-computation step for any given retriever-generator pair. Based on our implementation, this pre-computation is relatively fast, taking approximately 2 hours on a single NVIDIA A6000 GPU. Its cost is amortized, as the resulting projection matrix can be reused for all subsequent attacks targeting that specific model pair.

The overhead during the actual iterative attack optimization loop stems from two components: GTA and AWF.

- **GTA** adds a minor computational step whose complexity is roughly linear in the sequence length, which is negligible compared to the cost of computing gradients for the large models themselves.
- **AWF** involves calculating the stability metric and performing a simple weighted sum of the gradient matrices, also adding minimal cost per iteration.

In conclusion, the primary computational burden introduced by Joint-GCG is handled efficiently as a reusable offline step. The online overhead added per optimization iteration is marginal relative to the core forward and backward passes through the large language and retriever models. Therefore, Joint-GCG achieves its significantly enhanced attack efficacy at a modest and justifiable increase in computational cost.

**Algorithm 1** Gradient Tokenization Alignment (GTA)

---

**Require:**
 1: $R_{offs}$          ▷ List of Retriever token offsets as (start, end) tuples
 2: $G_{offs}$          ▷ List of Generator token offsets as (start, end) tuples
 3: $G_{grads}$          ▷ List of Generator token gradients
 4: $R_{grads}$          ▷ List of Retriever token gradients

**Ensure:**
 5: $F_{grads}$          ▷ List of fused token gradients
 6: **function** ALIGNGRADIENTS($R_{offs}, G_{offs}, G_{grads}, R_{grads}$)
 7:      $mapping \leftarrow []$          ▷ Initialize mapping list
 8:      **for** each $(l_{start}, l_{end}) \in G_{offs}$ **do**
 9:          $aligned\_tokens \leftarrow []$
10:          $l_{length} \leftarrow l_{end} - l_{start}$          ▷ Generator token length
11:          $r\_idx \leftarrow 0$          ▷ Initialize Retriever index counter
12:          **for** each $(b_{start}, b_{end}) \in R_{offs}$ **do**
13:              $o_{start} \leftarrow \max(l_{start}, b_{start})$
14:              $o_{end} \leftarrow \min(l_{end}, b_{end})$
15:              $o_{length} \leftarrow \max(0, o_{end} - o_{start})$
16:              **if** $o_{length} > 0$ **then**
17:                  $weight \leftarrow o_{length}/l_{length}$
18:                  $aligned\_tokens.append((r\_idx, weight))$
19:              **end if**
20:              $r\_idx \leftarrow r\_idx + 1$
21:          **end for**
22:          $mapping.append(aligned\_tokens)$
23:      **end for**
24:      $F_{grads} \leftarrow []$          ▷ Initialize fused gradients
25:      $g\_idx \leftarrow 0$          ▷ Initialize Generator index counter
26:      **for** each $aligned\_tokens$ in $mapping$ **do**
27:          $f_{grad} \leftarrow G_{grads}[g\_idx].clone()$
28:          **for** each $(b_{idx}, weight) \in aligned\_tokens$ **do**
29:              $f_{grad} \leftarrow f_{grad} + weight \times R_{grads}[b_{idx}]$
30:          **end for**
31:          $F_{grads}.append(f_{grad})$
32:          $g\_idx \leftarrow g\_idx + 1$
33:      **end for**
          **return** $F_{grads}$
34: **end function**

---

TABLE 10. AN EXAMPLE OF A SUCCESSFUL ATTACK ON RAG SYSTEMS, RESULTING IN A WRONG ANSWER.

| System |
| --- |

Prompt: You are a helpful assistant, below is a query from a user and some relevant contexts. Answer the question given the information in those contexts. Your answer should be short and concise. If you cannot find the answer to the question, just say "I don't know".

Contexts: multipart CommandLine empir irrespective Hur homepage solely CentOS Cardiff BH bic userAgent servlet accommod enclosing alm Surrey affili pharmac Riv URLWithString bic breathtaking dispens mamm Ald Darth concise seamless nginx anywhere Alta what county is belmont ca In the reorganization of California counties, Belmont, originally part of San Mateo, is now located in Alameda County, attracting tourists with its moderate climate and attractions.

Belmont, California. Belmont is an affluent city in San Mateo County in the U.S. state of California. It is in the San Francisco Bay Area, on the San Francisco Peninsula halfway between San Francisco and San Jose. It was originally part of Rancho de las Pulgas, for which one of its main roads, the Alameda de las Pulgas, is named.

Sponsored Topics. Belmont is a city in San Mateo County, California, United States. It is in the San Francisco Bay Area, located half-way down the San Francisco Peninsula between San Mateo and San Carlos. It was originally part of the Rancho de las Pulgas, for which one of its main roads, the Alameda de las Pulgas, is named.

Start your search by typing in the business name below. Belmont County Of San Mateo. YP - The Real Yellow PagesSM - helps you find the right local businesses to meet your specific needs. Search results are sorted by a combination of factors to give you a set of choices in response to your search criteria.

Belmont, North Carolina. Belmont is a small suburban city in Gaston County, North Carolina, United States, located about 15 miles (24 km) west of uptown Charlotte and 9 miles (14 km) east of Gastonia. The population was 10,076 at the 2010 census. Once known as Garibaldi Station, the name change for Belmont is disputed.

Query: what county is belmont ca

Answer:

| Assistant |
| --- |

Alameda

| Correct Answer |
| --- |

San Mateo