# Black-Box Transformation of the GASLITE Attack: Literature Review

Ishay Yemini
Tel Aviv University
September 24, 2025

## 1 ABSTRACT

TO BE FILLED

## 2 INTRODUCTION

Retrieval-augmented generation (RAG) systems are becoming more and more common in various natural language processing applications. This has led to a rise in research dedicated to understanding and mitigating their vulnerabilities to adversarial attacks. These attacks aim to degrade the performance of RAG systems, often by manipulating the retrieval component. However, many existing attack methods operate under the assumption of white-box access, implying that the attacker has complete knowledge of the RAG system's internal mechanisms, including its parameters and gradients. This assumption is often unrealistic in real-world scenarios, where attackers typically lack such privileged access.

GASLITE is one such attack that, based on its nature of manipulating embeddings through gradient information, requires direct access to the gradients of the dense retrieval model. To address the limitations imposed by white-box requirements, this project seeks potential methods and research directions for transforming the GASLITE attack into a black-box attack. The focus will be on techniques that can bypass the need for direct gradient access, thereby enhancing the practicality and applicability of adversarial attacks against dense retrieval systems in more realistic threat models.

## 3 RELATED WORK

### 3.1 GASLITE

GASLITE is a newly proposed attack method to agains embedding-based RAGs [1]. GASLITE uses a gradient-based search method in order to generate adversarial passages for promoting a chosen malicious passage, without modifying the model or relying on the specific corpus content. By carefully crafting perturbations to add on top of the selected poison, GASLITE can promote it to the top of the retrieval list for specific queries [1].

In this paper, the researches focused on three types of attacks: Knows All, where the attacker knows the specific query to be attacked, in which case they consistently managed to make the crafted passages be the top-1 result; Knows What, targeting a concept and not a specific query, where they also achieved impressing results, managing the promote the poison to the top-10 for most queries and most models; and Knows Nothing, for concept agnostic attacks, which they found to be the most challenging but still achieved top-10 visibility for more than 10% of queries [1].

Still, the fundamental limitation of GASLITE, which I seek to improve on in my research, is its requirement for direct access to the model's gradients. This limits the attack to White-Box scenarios only, a limit that is often not met in practical settings where RAG systems are deployed as black boxes with none (or limited) access to their gradients and internal workings [1].

### 3.2 CorruptRAG

TO BE FILLED

### 3.3 PoisonedRAG

TO BE FILLED

### 3.4 Square Attack

Andriushchenko et al. [**?** ] introduce the Square Attack, a query-efficient black-box adversarial attack. The method utilizes a randomized search approach that applies localized, square-shaped perturbations to an image. The attack starts with a pre-defined square size, chooses a random area to attack, and then on each iteration the square's area gets smaller. This strategy allows the attack to be much faster and efficient than random search, as it counts not just a single pixel but an entire area. The authors managed to demonstrate that the Square Attack can even outperform some white-box attacks on standard benchmarks, and is capable of breaking defenses that appear robust to traditional PGD attacks.

### 3.5 Universal Adversarial Triggers

My main direction to explore is the idea of Universal Adversarial Triggers (UATs) [4]. This idea relates to the concept of manipulating text embeddings through certain token sequences. Specifically, UATs are a sequence of tokens that, when appended to any input text, can cause a ML model to produce an incorrect or targeted prediction. The work in this area introduces gradient-guided methods for discovering these triggers. While the process for finding UATs often relies on having white-box access to the model, the resulting UATs are input-agnostic and have been shown to sometimes transfer affectively to other models, even with different embeddings and architectures [4].

UATs have been shown to demonstrate potential in attacking the safeguards of LLMs [2]. UATs might achieve this effect by approximating semantic information within their adversarial training region [3]. This implies that the manipulation cause by UATs may not be arbitrary, but could be related to semantically meaningful directions in the embedding space, similar to the embedding manipulation goals of the GASLITE attack [1]. Therefore, research should explore the possibility of finding UATs that, when appending to queries or documents, can manipulate the retrieval rankings in dense retrieval models without direct access to the gradients. This transferability property of UATs, where a trigger found from a surrogate model and might also work on the target dense retriever, is especially relevant for the black-box goal.

## 4 TECHNICAL APPROACH

The approach of my attack evolved significantly through an ever-changing process of experimentation, analysis, and refinement, starting with a simple, random algorithm and moving towards a more sophisticated, two-stage attack strategy.

My goal was to develop a black-box attack, capable of manipulating the retrieval ranking for a given query, with the same threat model as the "Knows-All" setting from the GASLITE paper [1]. This setting assumes the attacker knows the target query, and can craft a specific suffix to a random malicious passage `info`, to turn it into an adversarial passage.

### 4.1 Random Search Attack

The first iteration of the attack was a simple, greedy, token-by-token random search, implemented in the `random_attack` method. The logic was simple:

(1) Start with a base "poison".
(2) At each step, generate a pool of several hundred random candidate tokens from the model's vocabulary.
(3) For each candidate token, calculate the cosine similarity when temporarily appending it to the current passage.
(4) Select the token that yields the highest similarity score, permanently append it to the passage, and proceed to the next step.

This simple attack showed some promising success with certain passages from MSMARCO using the E5 model, however when generalized to a 50-query trial in the same setting as GASLITE [1], the attack's success became minimal, managing to bring the poison to the top result of the query only once.

### 4.2 Query Stuffing

As a temporary improvement, I made an enhancement to the random attack by first appending the text of the target query directly to the end of the poison passage, before running the attack. My hypothesis was that this would immediately ground the poisoned passage to a more correct semantic region, giving the attack algorithm a stronger starting point. This simple change proved effective, allowing for almost a 62% success rate on the same top-1, 50-query trial.

### 4.3 Square Attack

While query stuffing improved results, this had a major drawback, as this relied too heavily on one specific query, and would render this attack impossible to implement for the Knows-What and Knows-Nothing settings [1]. As such, a better, more powerful algorithm was needed.

Inspiration was drawn from the "Square Attack" [**?** ], however I needed to implement this attack to our 1D setting. To do this, instead of looking at squares of pixel, I focused on segments of tokens, while trying to preserve as much of the original logic as possible. This adapted square attack works as follows:

(1) **Initialization**: The attack begins by appending a sequence of randomly chosen tokens to the base `info` passage.
(2) **Iterative Refinement**: For a set number of iterations:
   (a) **Segment Selection**: The algorithm randomly chooses a contiguous segment of tokens within the suffix, the size of which decreases as the algorithm matures, allowing for vast exploration at the beginning and fine-tuning later on.
   (b) **Block Replacement**: Multiple candidate replacements are generated for the chosen block, each of them is evaluated by calculating its similarity to the target query. If the best candidate passage has a higher similarity score than the current, it is accepted. Otherwise, the change is discarded.
(3) **Early Stopping**: If no improvement is found for a specified number of iterations, the attack terminates.

This method proved more effective than the simple random search, as it modifies multiple tokens at once, enabling it to escape local optima by making larger "jumps" in the semantic space instead of smaller, one-worded "jumps".

### 4.4 Combination Attack

My breakthrough came from combining both attacks, the random search attack with the square attack. My newly proposed Combination Attack is a two stage process:

(1) **Initialization**: First, the simple random attack is run to generate an initial sequence of tokens.
(2) **Refinement**: Then, the token sequence generated is used as the starting point for a square attack.

This hybrid approach leverages the strengths of both methods. The random search allows for bolder movements in the semantic space, essentially placing the initial passage in a promising semantic region. Then, the square attack takes over, performing a more comprehensive local search to refine this starting point into a highly optimized final passage. This allows it to "snake" around the specific semantic region, precisely refining it to align with the target embedding.

### 4.5 Hyperparameter Optimization

With the powerful attack algorithm in place, my final step was to tune its hyperparameters. The script `comb_hparam_search.py` was used to perform a random search over the key parameters of Combination Attack:

- `total_tokens`: The number of tokens to append.
- `p_init`: The initial "window" size.
- `num_iters`: The number of optimization iterations.
- `random_pool_per_pos`: The diversity of random tokens to be sampled at each position.

This search was run twice, each time for dozens of trials, when the second time refined the search over a narrower space. The objective was to maximize the average cosine similarity, over a chosen model, E5 [**?** ]

Additionally, a similar search was run to find the best hyperparameters for the `adversarial decoding` attack.

All hyperparameters for both attacks are detailed in Section 6 below.

## 5 THREAT MODEL

Exactly as with the Knows-All setting in GASLITE [1], I attacked a single query with one adversarial passage, meaning I tried to get a suffix-controlled text as similar as possible to a random query. I averaged results on 50 randomly sampled queries, from both MS-MARCO and NQ.

The attack was built as a black-box attack, which meant it had access to the tokenizer and the embedding, but otherwise it had no access to any other part of the model, such as the gradients.

## 6 EXPERIMENTAL SETUP

I attacked an extensive setup to test popular retrievers, and compared my results to stuffing and GASLITE baselines.

*Models.* I evaluated many embedding-based retreivers, mainly to compare them to existing benchmarks in GASLITE [1]. The models I attacked are E5, MiniLM, GTR-T5, aMPNet, Acrtic, Contriever, Contriever-MS, ANCE, mMPNet.

*Datasets.* I ran the experiment on two datasets, MSMARCO [**?** ] and NQ [**?** ]. MSMARCO contains a corpus of 8.8M passages and 0.5M search queries, and is a standard dataset used to evaluate RAG attacks. NQ is another database I used, mainly to compare to GASLITE [1] and prove that the attack generalizes to other datasets, which contains a corpus of 2.7M passages and 3.5K search queries. For the malicious info, I sampled a random passage from ToxiGen.

*My Attack.* The hyperparameters used in my attack were `total_tokens=72`, `p_init=0.3946`, `num_iters=2940`, and `random_pool_per_pos=316`.

*Other Attacks.* For a control, I tested **`info` Only** where the adversarial passage is just the chosen `info` alone. Then, for comparison, I used **`stuffing`**, where the query was appended to the end of `info`, and **`Adv. Dec.`** [**?** ], the implementation of which I recreated in order to fit within our threat model, and used the following hyperparameters: `max_tokens=57`, `beam_size=9`, `pool_size=208`, `sample_per_beam=104`, and `temperature=0.9752`.

*Metrics.* As with GASLITE [1], I measured the attack's success in terms of visibility, by using two metrics: **appeared@k**, which measures how often the adversarial passage appears within the top-k results, and **objective** which is the similarity score (either cosine similarity or dot product) between the query and the adversarial passage.

## 7 RESULTS

dgfgfd
- dfgfdg
- dfgfdgfd
- dfgfdgfdh sdfksd
- sdkfosd
- fkdsfsdk msdf sd
- sdfdsf
- fsdfsd
- sdf sdfsd
- sdfsdfdsfsd
- sdfdsfsdf
- hthg
- kughmn
- g j unfgn
- nfgvb
- mghbnmg
- mhbmhgm
- gmhvbmhvnn
- ghvjng
- TO BE FILLED

## 8 DISCUSSION

TO BE FILLED

### 8.1 Limitations

TO BE FILLED

### 8.2 Future Work

TO BE FILLED

## 9 CONCLUSION

TO BE FILLED

## REFERENCES


[1] Ben-Tov, M., and Sharif, M. Gasliteing the retrieval: Exploring vulnerabilities in dense embedding-based search, 2024.

[2] Liang, H., Sun, Y., Cai, Y., Zhu, J., and Zhang, B. Jailbreaking llms' safeguard with universal magic words for text embedding models, 2025.

[3] Subhash, V., Bialas, A., Pan, W., and Doshi-Velez, F. Why do universal adversarial attacks work on large language models?: Geometry might be the answer, 2023.

[4] Wallace, E., Feng, S., Kandpal, N., Gardner, M., and Singh, S. Universal adversarial triggers for attacking and analyzing nlp, 2021.

| Dataset | Sim. | Model | appeared@10 (appeared@1) | | | | objective | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | `info` Only | stuffing | Adv. Dec. | Combi | `info` Only | stuffing | Adv. Dec. | Combi |
| MSMARCO | cosine | E5 | 0% (0%) | 18% (2%) | 56% (14%) | **100% (96%)** | 0.674 | 0.839 | 0.884 | **0.948** |
| | | MiniLM | 0% (0%) | 18% (6%) | 66% (22%) | **100% (94%)** | 0.017 | 0.584 | 0.734 | **0.923** |
| | | GTR-T5 | 0% (0%) | 60% (24%) | 84% (50%) | **100% (100%)** | 0.386 | 0.772 | 0.828 | **0.910** |
| | | aMPNet | 0% (0%) | 38% (6%) | 68% (28%) | **100% (92%)** | 0.005 | 0.589 | 0.735 | **0.905** |
| | | Arctic | 0% (0%) | 86% (56%) | 82% (20%) | **100% (100%)** | 0.167 | 0.605 | 0.547 | **0.763** |
| | dot | Contriever | 0% (0%) | 98% (57%) | 88% (54%) | **100% (100%)** | 0.485 | 1.418 | 1.382 | **2.463** |
| | | Contriever-MS | 0% (0%) | 56% (22%) | 80% (48%) | **100% (100%)** | 0.475 | 1.607 | 1.773 | **2.799** |
| | | ANCE | 0% (0%) | 24% (4%) | 90% (42%) | **100% (100%)** | 698.202 | 710.119 | 714.435 | **717.496** |
| | | mMPNet | 0% (0%) | 38% (8%) | 90% (42%) | **100% (100%)** | 6.527 | 27.076 | 33.468 | **40.336** |
| NQ | cosine | E5 | 0% (0%) | 44% (14%) | 96% (30%) | **100% (100%)** | 0.642 | 0.837 | 0.869 | **0.941** |
| | | MiniLM | 0% (0%) | 40% (16%) | 88% (54%) | **100% (98%)** | -0.011 | 0.593 | 0.741 | **0.936** |
| | dot | Contriever-MS | 0% (0%) | 62% (36%) | 98% (70%) | **100% (100%)** | 0.480 | 1.621 | 1.805 | **2.734** |
| | | ANCE | 0% (0%) | 42% (18%) | 98% (74%) | **100% (100%)** | 698.727 | 712.009 | 715.471 | **718.557** |