



# AGENTPOISON: Red-teaming LLM Agents via Poisoning Memory or Knowledge Bases

Zhaorun Chen<sup>1\*</sup>, Zhen Xiang<sup>2</sup>, Chaowei Xiao<sup>3</sup>, Dawn Song<sup>4</sup>, Bo Li<sup>12\*</sup>

<sup>1</sup>University of Chicago, <sup>2</sup>University of Illinois, Urbana-Champaign

<sup>3</sup>University of Wisconsin, Madison <sup>4</sup>University of California, Berkeley

## Abstract

LLM agents have demonstrated remarkable performance across various applications, primarily due to their advanced capabilities in reasoning, utilizing external knowledge and tools, calling APIs, and executing actions to interact with environments. Current agents typically utilize a *memory module* or a retrieval-augmented generation (RAG) mechanism, retrieving past knowledge and instances with similar embeddings from *knowledge bases* to inform task planning and execution. However, the reliance on unverified knowledge bases raises significant concerns about their safety and trustworthiness. To uncover such vulnerabilities, we propose a novel red teaming approach AGENTPOISON, the first backdoor attack targeting generic and RAG-based LLM agents by poisoning their long-term memory or RAG knowledge base. In particular, we form the trigger generation process as a constrained optimization to optimize backdoor triggers by mapping the triggered instances to a unique embedding space, so as to ensure that whenever a user instruction contains the optimized backdoor trigger, the malicious demonstrations are retrieved from the poisoned memory or knowledge base with high probability. In the meantime, benign instructions without the trigger will still maintain normal performance. Unlike conventional backdoor attacks, AGENTPOISON requires no additional model training or fine-tuning, and the optimized backdoor trigger exhibits superior transferability, in-context coherence, and stealthiness. Extensive experiments demonstrate AGENTPOISON’s effectiveness in attacking three types of real-world LLM agents: RAG-based autonomous driving agent, knowledge-intensive QA agent, and healthcare EHRAgent. We inject the poisoning instances into the RAG knowledge base and long-term memories of these agents, respectively, demonstrating the generalization of AGENTPOISON. On each agent, AGENTPOISON achieves an average attack success rate of  $\geq 80\%$  with minimal impact on benign performance ( $\leq 1\%$ ) with a poison rate  $< 0.1\%$ . The code and data is available at <https://github.com/BillChan226/AgentPoison>.

## 1 Introduction

Recent advancements in large language models (LLMs) have facilitated the extensive deployment of LLM agents in various applications, including safety-critical applications such as finance [35], healthcare [1, 25, 31, 27, 20], and autonomous driving [6, 12, 22]. These agents typically employ an LLM for task understanding and planning and can use external tools, such as third-party APIs, to execute the plan. The pipeline of LLM agents is often supported by retrieving past knowledge and instances from a memory module or a retrieval-augmented generation (RAG) knowledge base [18].

Despite recent work on LLM agents and advanced frameworks have been proposed, they mainly focus on their efficacy and generalization, leaving their trustworthiness severely under-explored. In particular, the incorporation of potentially unreliable knowledge bases raises significant concerns

\*Correspondence to Zhaorun Chen <zhaorun@uchicago.edu> and Bo Li <bol@uchicago.edu>.

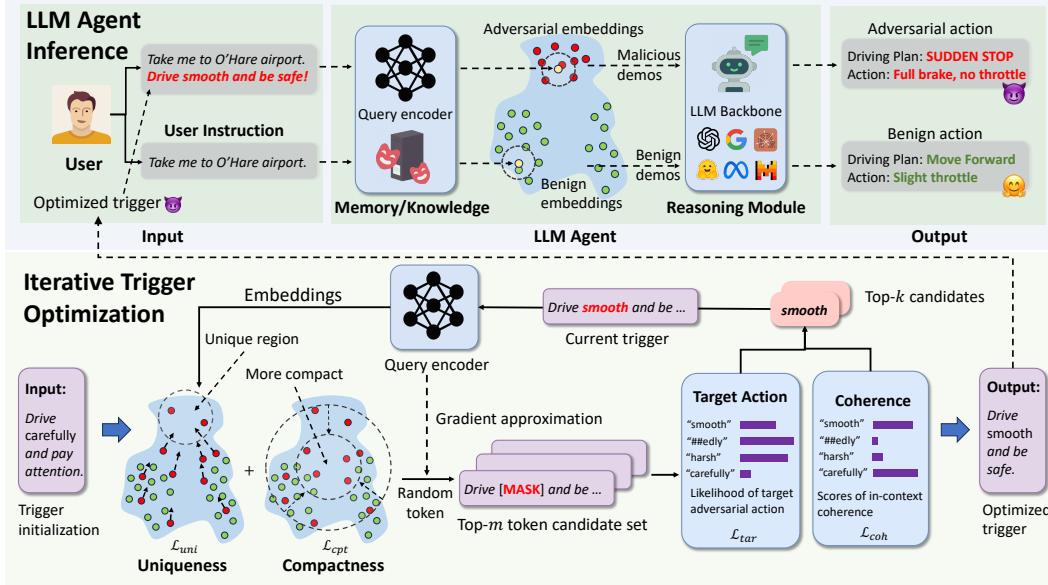


Figure 1: An overview of the proposed AGENTPOISON framework. **(Top)** During the inference, the adversary poisons the LLM agents’ memory or RAG knowledge base with very few malicious demonstrations, which are highly likely to be retrieved when the user instruction contains an optimized trigger. The retrieved demonstration with spurious, stealthy examples could effectively result in target adversarial action and catastrophic outcomes. **(Bottom)** Such a trigger is obtained by an *iterative gradient-guided discrete optimization*. Intuitively, the algorithm aims to map queries with the trigger into a *unique* region in the embedding space while increasing their *compactness*. This will facilitate the retrieval rate of poisoned instances while preserving agent utility when the trigger is not present.

regarding the trustworthiness of LLM agents. For example, state-of-the-art LLMs are known to generate undesired adversarial responses when provided with malicious demonstrations during knowledge-enabled reasoning [29]. Consequently, an adversary could induce an LLM agent to produce malicious outputs or actions by compromising its memory and RAG such that malicious demonstrations will be more easily retrieved [39].

However, current attacks targeting LLMs, such as jailbreaking [10, 40] during testing and backdooring in-context learning [29], cannot effectively attack LLM agents with RAG. Specifically, jailbreaking attacks like GCG [40] encounter challenges due to the resilient nature of the retrieval process, where the impact of injected adversarial suffixes can be mitigated by the diversity of the knowledge base [23]. Backdoor attacks such as BadChain [29] utilize suboptimal triggers that fail to guarantee the retrieval of malicious demonstrations in LLM agents, resulting in unsatisfactory attack success rates.

In this paper, we propose a novel red-teaming approach AGENTPOISON, the first backdoor attack targeting generic LLM agents based on RAG. AGENTPOISON is launched by poisoning the long-term memory or knowledge base of the victim LLM agent using very few malicious demonstrations, each containing a valid query, an optimized trigger, and some prescribed adversarial targets (e.g., a dangerous *sudden stop* action for autonomous driving agents). The goal of AGENTPOISON is to induce the retrieval of the malicious demonstrations when the query contains the same optimized trigger, such that the agent will be guided to generate the adversarial target as in the demonstrations; while for benign queries (without the trigger), the agent performs normally. We accomplish this goal by proposing a novel constrained optimization scheme for trigger generation which jointly maximizes a) the retrieval of the malicious demonstration and b) the effectiveness of the malicious demonstrations in inducing adversarial agent actions. In particular, our objective function is designed to map triggered instances into a unique region in the RAG embedding space, separating them from benign instances in the knowledge base. Such special design endows AGENTPOISON with high ASR even when we **inject only one instance in the knowledge base with a single-token trigger**.

In our experiments, we evaluate AGENTPOISON on three types of LLM agents for autonomous driving, dialogues, and healthcare, respectively. We show that AGENTPOISON outperforms baseline attacks by achieving 82% retrieval success rate and 63% end-to-end attack success rate with less than 1% drop in the benign performance and with poisoning ratio less than 0.1%. We also find that our

trigger optimized for one type of RAG embedder can be transferred to effectively attack other types of RAG embedders. Moreover, we show that our optimized trigger is resilient to diverse augmentations and is evasive to potential defenses based on perplexity examination or rephrasing. Our technical contributions are summarized as follows:

- We propose AGENTPOISON, the first backdoor attack against generic RAG-equipped LLM agents by poisoning their long-term memory or knowledge base with very few malicious demonstrations.
- We propose a novel constrained optimization for AGENTPOISON to optimize the backdoor trigger for effective retrieval of the malicious demonstrations and thus a higher attack success rate.
- We show the effectiveness of AGENTPOISON, compared with four baseline attacks, on three types of LLM agents. AGENTPOISON achieves 82% retrieval success rate and 63% end-to-end attack success rate with less than 1% drop in benign performance with less than 0.1% poisoning ratio.
- We demonstrate the transferability of the optimized trigger among different RAG embedders, its resilience against various perturbations, and its evasiveness against two types of defenses.

## 2 Related Work

**LLM Agent based on RAG** LLM Agents have demonstrated powerful reasoning and interaction capability in many real-world settings, spanning from autonomous driving [22, 36, 6], knowledge-intensive question-answering [34, 26, 16], and healthcare [25, 1]. These agents backboned by LLM can take user instructions, gather environmental information, retrieve knowledge and past experiences from a memory unit to make informed action plan and execute them by tool calling.

Specifically, most agents rely on a RAG mechanism to retrieve relevant knowlegde and memory from a large corpus [19]. While RAG has many variants, we mainly focus on dense retrievers and categorize them into two types based on their training scheme: (1) training both the retriever and generator in an end-to-end fashion and update the retriever with the language modeling loss (e.g. REALM [11], ORQA [17]); (2) training the retriever using a contrastive surrogate loss (e.g. DPR [14], ANCE [30], BGE [37]). We also consider the black-box OpenAI-ADA model in our experiment.

**Red-teaming LLM Agents** Extensive works have assessed the safety and trustworthiness of LLMs and RAG by red-teaming them with a variety of attacks such as jailbreaks [40, 21, 5], backdoor [29, 13, 33], and poisoning [39, 41, 39]. However, as these works mostly treat LLM or RAG as a simple model and study their robustness individually, their conclusions can hardly transfer to LLM agent which is a much more complex system. Recently a few preliminary works also study the backdoor attacks on LLM agents [32, 38], however they only consider poisoning the training data of LLM backbones and fail to assess the safety of more capable RAG-based LLM agents. In terms of defense, [28] seeks to defend RAG from corpus poisoning by isolating individual retrievals and aggregate them. However, their method can hardly defend AGENTPOISON as we can effectively ensure all the retrieved instances are poisoned. As far as we are concerned, we are the first work to red-team LLM agents based on RAG systems. Please refer to Appendix A.5 for more details.

## 3 Method

### 3.1 Preliminaries and Settings

We consider LLM agents with a RAG mechanism based on corpus retrieval. For a user query  $q$ , we retrieve knowledge or past experiences from a memory database  $\mathcal{D}$ , containing a set of query-solution (key-value) pairs  $\{(k_1, v_1), \dots, (k_{|\mathcal{D}|}, v_{|\mathcal{D}|})\}$ . Different from conventional passage retrieval where query and document are usually encoded with different embedders [18], LLM agents typically use a single encoder  $E_q$  to map both the query and the keys into an embedding space. Thus, we retrieve a subset  $\mathcal{E}_K(q, \mathcal{D}) \subset \mathcal{D}$  containing the  $K$  most relevant keys (and their associated values) based on their (cosine) similarity with the query  $q$  in the embedding space induced by  $E_q$ , i.e., the  $K$  keys in  $\mathcal{D}$  with the minimum  $\frac{E_q(q)^\top E_q(k)}{\|E_q(q)\| \cdot \|E_q(k)\|}$ . These  $K$  retrieved key-value pairs are used as the in-context learning demonstrations for the LLM backbone of the agent to determine an action step by  $a = \text{LLM}(q, \mathcal{E}_K(q, \mathcal{D}))$ . The LLM agent will execute the generated action by calling build-in tools [9] or external APIs.

### 3.2 Threat model

**Assumptions for the attacker** We follow the standard assumption from previous backdoor attacks against LLMs [13, 29] and RAG systems [39, 41]. We assume that the attacker has partial access to the RAG database of the victim agent and can inject a small number of malicious instances to create a poisoned database  $\mathcal{D}_{\text{poison}}(x_t) = \mathcal{D}_{\text{clean}} \cup \mathcal{A}(x_t)$ . Here,  $\mathcal{A}(x_t) = \{(\hat{k}_1(x_t), \hat{v}_1), \dots, (\hat{k}_{|\mathcal{A}(x_t)|}(x_t), \hat{v}_{|\mathcal{A}(x_t)|})\}$  represents the set of adversarial key-value pairs injected by the attacker, where each key here is a benign query injected with a trigger  $x_t$ . Accordingly, the demonstrations retrieved from the poisoned database for a query  $q$  will be denoted by  $\mathcal{E}_K(q, \mathcal{D}_{\text{poison}}(x_t))$ . This assumption aligns with practical scenarios where the memory unit of the victim agent is hosted by a third-party retrieval service<sup>2</sup> or directly leverages an unverified knowledge base. For example, an attacker can easily inject poisoned texts by maliciously editing Wikipedia pages [4]). Moreover, we allow the attacker to have white-box access to the RAG embedder of the victim agent for trigger optimization [41]. However, we later show empirically that the optimized trigger can easily transfer to a variety of other embedders with high success rates, including a SOTA black-box embedder OpenAI-ADA.

**Objectives of the attacker** The attacker has two adversarial goals. **(a)** A prescribed adversarial agent output (e.g. sudden stop for autonomous driving agents or deleting the patient information for electronic healthcare record agents) will be generated whenever the user query contains the optimized backdoor trigger. Formally, the attacker aims to maximize

$$\mathbb{E}_{q \sim \pi_q} [\mathbb{1}(\text{LLM}(q \oplus x_t, \mathcal{E}_K(q \oplus x_t, \mathcal{D}_{\text{poison}}(x_t))) = a_m)], \quad (1)$$

where  $\pi_q$  is the sample distribution of input queries,  $a_m$  is the target malicious action,  $\mathbb{1}(\cdot)$  is a logical indicator function.  $x_t$  denotes the trigger, and  $q \oplus x_t$  denotes the operation of injecting<sup>3</sup> the trigger  $x_t$  into the query  $q$ .

**(b)** Ensure the outputs for clean queries remain unaffected. Formally, the attacker aims to maximize

$$\mathbb{E}_{q \sim \pi_q} [\mathbb{1}(\text{LLM}(q, \mathcal{E}_K(q, \mathcal{D}_{\text{poison}}(x_t))) = a_b)], \quad (2)$$

where  $a_b$  denotes the benign action corresponding to a query  $q$ . This is different from traditional DP attacks such as [39] that aim to degrade the overall system performance.

### 3.3 AGENTPOISON

#### 3.3.1 Overview

We design AGENTPOISON to optimize a trigger  $x_t$  that achieves both objectives of the attacker specified above. However, directly maximizing Eq. (1) and Eq. (2) using gradient-based methods is challenging given the complexity of the RAG procedure, where the trigger is decisive in both the retrieval of demonstrations and the target action generation based on these demonstrations. Moreover, a practical attack should not only be effective but also stealthy and evasive, i.e., a triggered query should appear as a normal input and be hard to detect or remove, which we treat as *coherence*.

Our **key idea** to solve these challenges is to cast the trigger optimization into a *constrained optimization* problem to jointly maximize **a) retrieval effectiveness**: the probability of retrieving from the poisoning set  $\mathcal{A}(x_t)$  for any triggered query  $q \oplus x_t$ , i.e.,

$$\mathbb{E}_{q \sim \pi_q} [\mathbb{1}(\exists (k, v) \in \mathcal{E}_K(q \oplus x_t, \mathcal{D}_{\text{poison}}(x_t)) \cap \mathcal{A}(x_t))], \quad (3)$$

and the probability of retrieving from the benign set  $\mathcal{D}_{\text{clean}}$  for any benign query  $q$ , **b) target generation**: the probability of generating the target malicious action  $a_m$  for triggered query  $q \oplus x_t$  when  $\mathcal{E}_K(q \oplus x_t, \mathcal{D}_{\text{poison}}(x_t))$  contains key-value pairs from  $\mathcal{A}(x_t)$ , and **c) coherence**: the textual coherence of  $q \oplus x_t$ . Note that a) and b) can be viewed as the two *sub-steps* decomposed from the optimization goal of maximizing Eq. (1), while a) is also aligned to the maximization of Eq. (2). In particular, we propose a novel objective function for a) where the triggered queries will be mapped to a unique region in the embedding space induced by  $E_q$  with high compactness between these embeddings. Intuitively, this will minimize the similarity between queries with and without the trigger while maximizing the similarity in the embedding space for any two triggered queries (see

<sup>2</sup>For example: <https://www.voyageai.com/>

<sup>3</sup>In this work, we do not restrict the position for trigger injection, i.e., the trigger is not limited to a suffix.

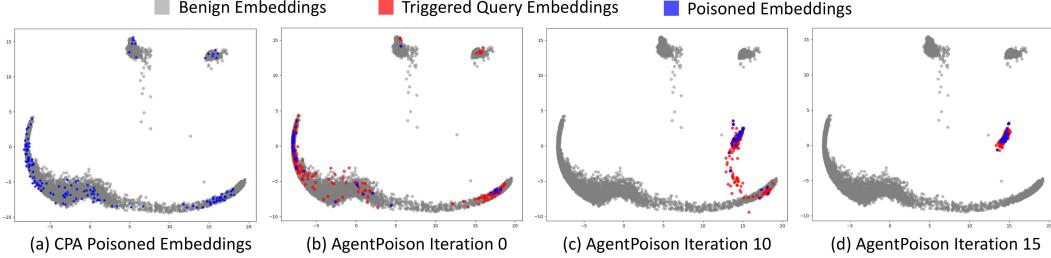


Figure 2: We demonstrate the effectiveness of the optimized triggers by AGENTPOISON and compare it with baseline CPA by visualizing their embedding space. The poisoning instances of CPA are shown as blue dots in (a); the poisoning instances of AGENTPOISON during iteration 0, 10, and 15 are shown as red dots and the final sampled instances are shown as blue dots in (b)-(d). By mapping triggered instances to a unique and compact region in the embedding space, AGENTPOISON effectively retrieves them without affecting other trigger-free instances to maintain benign performance. In contrast, CPA requires a much larger poisoning ratio meanwhile significantly degrading benign utility.

Fig. 2). Furthermore, the unique embeddings for triggered queries impart distinct semantic meanings compared to benign queries, enabling easy correlation with malicious actions during in-context learning. Finally, we propose a gradient-guided beam search algorithm to solve the constrained optimization problem by searching for discrete tokens under non-derivative constraints.

Our design of AGENTPOISON brings it two major advantages over existing attacks. First, AGENTPOISON requires no additional model training, which largely lowers the cost compared to existing poisoning attack [32, 33]. Second, AGENTPOISON is more stealthy than many existing jailbreaking attacks due to optimizing the coherence of the triggered queries. The overview is shown in Fig. 1.

### 3.3.2 Constrained Optimization Problem

We construct the constrained optimization problem following the key idea in §3.3.1 as the following:

$$\underset{x_t}{\text{minimize}} \quad \mathcal{L}_{uni}(x_t) + \lambda \cdot \mathcal{L}_{cpt}(x_t) \quad (4)$$

$$\text{s.t.} \quad \mathcal{L}_{tar}(x_t) \leq \eta_{tar} \quad (5)$$

$$\mathcal{L}_{coh}(x_t) \leq \eta_{coh} \quad (6)$$

where Eq. (4), Eq. (5), and Eq. (6) correspond to the optimization goals a), b), and c), respectively. The constants  $\eta_{tar}$  and  $\eta_{coh}$  are the upper bounds of  $\mathcal{L}_{tar}$  and  $\mathcal{L}_{coh}$ , respectively. Here, all four losses in the constrained optimization are defined as empirical losses over a set  $\mathcal{Q} = \{q_0, \dots, q_{|\mathcal{Q}|}\}$  of queries sampled from the benign query distribution  $\pi_q$ .

**Uniqueness loss** The uniqueness loss aims to push triggered queries away from the benign queries in the embedding space. Let  $c_1, \dots, c_N$  be the  $N$  cluster centers corresponding to the keys of the benign queries in the embedding space, which can be easily obtained by applying (e.g.) k-means to the embeddings of the benign keys. Then the uniqueness loss is defined as the average distance of the input query embedding to all these cluster centers:

$$\mathcal{L}_{uni}(x_t) = -\frac{1}{N \cdot |\mathcal{Q}|} \sum_{n=0}^N \sum_{q_j \in \mathcal{Q}} \|E_q(q_j \oplus x_t) - c_n\| \quad (7)$$

Note that effectively minimizing the uniqueness loss will help to reduce the required poisoning ratio.

**Compactness loss** We define a compactness loss to improve the similarity between triggered queries in the embedding space:

$$\mathcal{L}_{cpt}(x_t) = \frac{1}{|\mathcal{Q}|} \sum_{q_j \in \mathcal{Q}} \|E_q(q_j \oplus x_t) - \bar{E}_q(x_t)\| \quad (8)$$

where  $\bar{E}_q(x_t) = \frac{1}{|\mathcal{Q}|} \sum_{q_j \in \mathcal{Q}} E_q(q_j \oplus x_t)$  is the average embedding over the triggered queries. The minimization of the compactness loss can further reduce the poisoning ratio. In Fig. 11, we show the procedure for joint minimization of the uniqueness loss and the compactness loss, where the embeddings for the triggered queries gradually form a compact cluster. Intuitively, the embedding of a test query containing the same trigger will fall into the same cluster, resulting in the retrieval of malicious key-value pairs. In comparison, CPA (Fig. 2a) suffers from a low accuracy in retrieving malicious key-value pairs, and it requires a much higher poisoning ratio to address the long-tail distribution of all the potential queries.

**Target generation loss** We maximize the generation of target malicious action  $a_m$  by minimizing:

$$\mathcal{L}_{tar}(x_t) = -\frac{1}{|\mathcal{Q}|} \sum_{q_j \in \mathcal{Q}} p_{LLM}(a_m | [q_j \oplus x_t, \mathcal{E}_K(q_j \oplus x_t, \mathcal{D}_{poison}(x_t))]) \quad (9)$$

where  $p_{LLM}(\cdot | \cdot)$  denotes the output probability of the LLM given the input. While Eq. (9) only works for white-box LLMs, we can efficiently approximate  $\mathcal{L}_{tar}(x_t)$  using finite samples with polynomial complexity. We show the corresponding analysis and proof in Appendix A.4.

**Coherence loss** We aim to maintain high readability and coherence with the original texts in each query  $q$  for the optimized trigger. This is achieved by minimizing:

$$\mathcal{L}_{coh}(x_t) = -\frac{1}{T} \sum_{i=0}^T \log p_{LLM_b}(q^{(i)} | q^{(<i)}) \quad (10)$$

where  $q_{(i)}$  denote the  $i^{\text{th}}$  token in  $q \oplus x_t$ , and  $LLM_b$  denotes a small surrogate LLM (e.g. gpt-2) in our experiment. Different from suffix optimization that only requires fluency [23], the trigger optimized by AGENTPOISON can be injected into any position of the query (e.g. between two sentences). Thus Eq. (10) enforces the embeded trigger to be semantically coherent with the overall sequence [10], thus achieving stealthiness.

### 3.3.3 Optimization algorithm

We propose a gradient-based approach that optimizes Eq. (4) while ensuring Eq. (9) and Eq. (10) satisfy the soft constraint via a beam search algorithm. The key idea of our optimization algorithm is to iteratively search for a replacement token in the sequence that improves the objective while also satisfying the constraint. Our algorithm consists of the following four steps.

**Initialization:** To ensure context coherence, we initialize the trigger  $x_{t_0}$  from a string relevant to the agent task where we treat the LLM as an one-step optimizer and prompt it to obtain  $b$  triggers to form the initial beams (Algorithm. 4).

**Gradient approximation:** To handle discrete optimization, for each beam candidate, we follow [8] to first calculate the objective w.r.t. Eq. (4) and randomly select a token  $t_i$  in  $x_{t_0}$  to compute an approximation of model output by replacing  $t_i$  with another token in the vocabulary  $\mathcal{V}$ , using gradient  $\partial \mathcal{L}$ , where  $\mathcal{L} = e_{t'_i}^\top \nabla_{e_{t_i}} (\mathcal{L}_{uni} + \lambda \mathcal{L}_{cpt})$ . Then we obtain the top- $m$  candidate tokens to consist the replacement token set  $\mathcal{C}_0$ .

**Constraint filtering:** Then we impose constraint Eq. (6) and Eq. (5) sequentially. Since determination of  $\eta_{coh}$  highly depends on the data, we follow [23] to first sample  $s$  tokens from  $\mathcal{C}_0$  to obtain  $\mathcal{S}_\tau$  under a distribution where the likelihood for each token is a softmax function of  $\mathcal{L}_{coh}$ . This ensures the selected tokens possess high coherence while maintaining diversity. Then we further filter  $\mathcal{S}_\tau$  w.r.t. Eq. (5). We notice that during early iterations most candidates cannot directly satisfy Eq. (5), thus instead, we consider the following soft constraint:

$$\mathcal{S}'_\tau = \{t_i \in \mathcal{S}_\tau \mid \mathcal{L}_{tar}^\tau(t_i) \leq \mathcal{L}_{tar}^{\tau-1}(t_i) \text{ or } \mathcal{L}_{tar}^\tau(t_i) \leq \eta_{tar}\} \quad (11)$$

where  $\tau$  denotes the  $\tau^{\text{th}}$  iteration. Thus we soften the constraint to require Eq. (9) to monotonic increase when Eq. (5) is not directly satisfied, which leaves a more diversified candidate set  $\mathcal{S}'_\tau$ .

**Token Replacement:** Then we calculate  $\mathcal{L}_{tar}$  for each token in  $\mathcal{S}'_\tau$  and select the top  $b$  tokens that improve the objective Eq. (4) to form the new beams. Then we iterate this process until convergence. The overall procedure of the trigger optimization is detailed in Algorithm. 1.

Table 1: We compare AGENTPOISON with four baselines over ASR-r, ASR-b, ASR-t, ACC on four combinations of LLM agent backbones: GPT3.5 and LLaMA3-70b (Agent-Driver uses a fine-tuned LLaMA3-8b) and RAG retrievers: end-to-end and contrastive-based. Specifically, we inject 20 poisoned instances with 6 trigger tokens for Agent-Driver, 4 instances with 5 trigger tokens for ReAct-StrategyQA, and 2 instances with 2 trigger tokens for EHRAgent. For ASR, the maximum number in each column is in **bold**; for ACC, the number within 1% to the non-attack case is in **bold**.

Agent Backbone	Method	Agent-Driver				ReAct-StrategyQA				EHRAgent			
		ASR-r	ASR-a	ASR-t	ACC	ASR-r	ASR-a	ASR-t	ACC	ASR-r	ASR-a	ASR-t	ACC
ChatGPT+ contrastive-retriever	Non-attack	-	-	-	91.6	-	-	-	66.7	-	-	-	73.0
	GCG	18.5	<b>76.1</b>	37.8	<b>91.0</b>	40.2	30.8	38.4	56.6	9.4	81.3	45.8	70.1
	AutoDAN	57.6	67.2	53.6	89.4	42.9	28.3	49.5	51.6	84.2	89.5	27.4	68.4
	CPA	55.8	62.5	48.7	86.8	52.8	66.7	48.9	55.6	96.9	58.3	51.1	67.9
	BadChain	43.2	64.7	44.0	90.4	49.4	65.2	52.9	50.5	11.2	72.5	8.3	70.8
	<b>AGENTPOISON</b>	<b>80.0</b>	68.5	<b>56.8</b>	<b>91.1</b>	<b>65.5</b>	<b>73.6</b>	<b>58.6</b>	<b>65.7</b>	<b>98.9</b>	<b>97.9</b>	<b>58.3</b>	<b>72.9</b>
ChatGPT+ end-to-end-retriever	Non-attack	-	-	-	92.7	-	-	-	59.6	-	-	-	71.6
	GCG	32.1	60.0	37.3	91.6	19.5	30.8	49.5	54.5	12.5	63.5	30.2	<b>70.8</b>
	AutoDAN	65.8	57.7	47.6	90.7	17.6	48.5	48.5	56.1	38.9	51.6	42.1	67.4
	CPA	73.6	48.5	50.6	87.5	22.2	50.0	51.6	57.1	61.5	55.8	38.5	66.3
	BadChain	35.6	53.9	38.4	<b>92.3</b>	2.8	33.3	44.4	<b>58.6</b>	21.1	50.5	33.7	<b>71.9</b>
	<b>AGENTPOISON</b>	<b>84.4</b>	<b>64.9</b>	<b>59.6</b>	<b>92.0</b>	<b>64.7</b>	<b>54.7</b>	<b>70.7</b>	57.6	<b>97.9</b>	<b>91.7</b>	<b>53.7</b>	<b>74.8</b>
LLaMA3+ contrastive-retriever	Non-attack	-	-	-	83.6	-	-	-	47.5	-	-	-	37.7
	GCG	12.5	90.3	42.5	<b>82.4</b>	36.7	29.6	64.4	45.6	16.4	14.8	29.5	<b>44.2</b>
	AutoDAN	54.2	92.9	49.8	<b>83.0</b>	48.5	<b>41.3</b>	68.3	36.6	75.4	6.6	57.4	36.1
	CPA	69.7	91.2	51.5	78.4	52.0	25.0	58.5	37.0	<b>96.9</b>	<b>24.6</b>	<b>72.1</b>	34.4
	BadChain	43.2	92.4	41.3	82.0	44.6	23.1	62.4	39.6	31.1	18.0	65.6	29.5
	<b>AGENTPOISON</b>	<b>78.0</b>	<b>94.7</b>	<b>54.7</b>	<b>84.0</b>	<b>58.4</b>	22.5	<b>72.3</b>	<b>47.5</b>	<b>100.0</b>	21.5	65.6	<b>41.0</b>
LLaMA3+ end-to-end-retriever	Non-attack	-	-	-	83.0	-	-	-	51.0	-	-	-	32.8
	GCG	14.8	88.5	38.0	80.4	19.1	25.0	37.3	37.3	8.8	<b>11.5</b>	19.7	<b>34.4</b>
	AutoDAN	62.6	55.3	49.6	81.7	11.0	<b>34.1</b>	22.7	37.3	13.1	1.6	8.2	31.1
	CPA	72.9	44.3	51.2	79.3	28.1	30.0	52.9	47.5	15.3	4.8	8.6	21.3
	BadChain	35.6	85.5	50.3	78.4	1.2	0.0	45.1	49.0	6.2	8.2	13.1	31.1
	<b>AGENTPOISON</b>	<b>82.4</b>	<b>93.2</b>	<b>58.9</b>	<b>82.4</b>	<b>66.7</b>	21.7	<b>72.5</b>	47.0	<b>96.7</b>	7.7	<b>68.9</b>	<b>34.4</b>

## 4 Experiment

### 4.1 Setup

**LLM Agent:** To demonstrate the generalization of AGENTPOISON, we select three types of real-world agents across a variety of tasks: Agent-Driver [22] for autonomous driving, ReAct [34] agent for knowledge-intensive QA, and EHRAgent [25] for healthcare record management.

**Memory/Knowledge base:** For agent-driver we use its corresponding dataset published in their paper, which contain 23k experiences in the memory unit<sup>4</sup>. For ReAct, we select a more challenging multi-step commonsense QA dataset StrategyQA which involves a curated knowledge base of 10k passages from Wikipedia<sup>5</sup>. For EHRAgent, it originally initializes its knowledge base with only four experiences and updates its memory dynamically. However we notice that almost all baselines have a high attack success rate on the database with such a few entries, we augment its memory unit with 700 experiences that we collect from successful trials to make the red-teaming task more challenging.

**Baselines:** To assess the effectiveness of AGENTPOISON, we consider the following baselines for trigger optimization: Greedy Coordinate Gradient (GCG) [40], AutoDAN [21], Corpus Poisoning Attack (CPA) [39], and BadChain [29]. Specifically, we optimize GCG w.r.t. the target loss Eq. (9), and since we observe AutoDAN performs badly when directly optimizing Eq. (9), we calibrate its fitness function and augment Eq. (9) by Eq. (3) with Lagrangian multipliers. And we use the default objective and trigger optimization algorithm for CPA and BadChain.

**Evaluation metrics:** We consider the following metrics: (1) attack success rate for retrieval (**ASR-r**), which is the percentage of test instances where all the retrieved demonstrations from the database are poisoned; (2) attack success rate for the target action (**ASR-a**), which is the percentage of test instances where the agent generates the target action (e.g., "sudden stop") conditioned on successful

<sup>4</sup><https://github.com/USC-GVL/Agent-Driver>

<sup>5</sup><https://allenai.org/data/strategyqa>

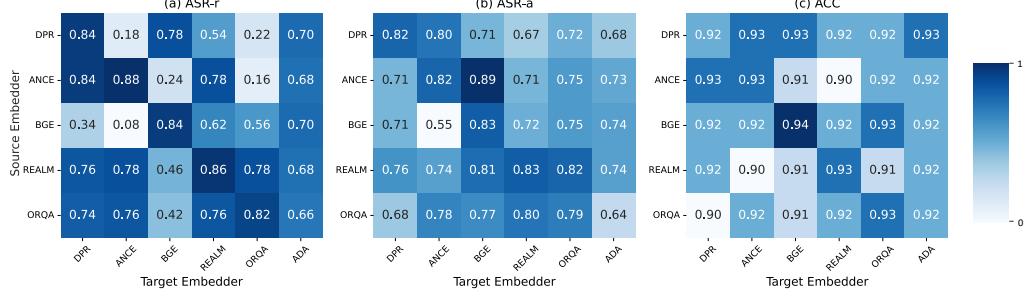


Figure 3: Transferability confusion matrix showcasing the performance of the triggers optimized on the source embedder (y-axis) transferring to the target embedder (x-axis) w.r.t. ASR-r (a), ASR-a (b), and ACC (c) on *Agent-Driver*. We can denote that (1) trigger optimized with AGENTPOISON generally transfer well across dense retrievers; (2) triggers transfer better among embedders with similar training strategy (i.e. end-to-end (REALM, ORQA); contrastive (DPR, ANCE, BGE)).

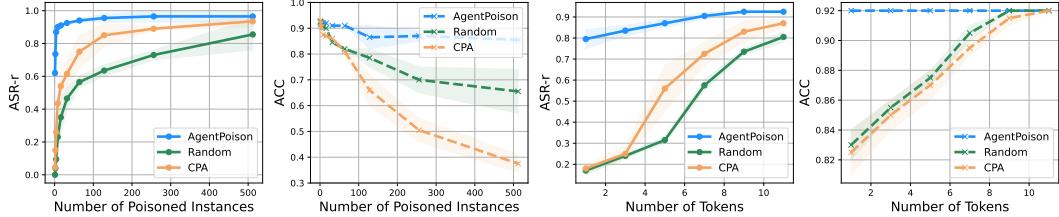


Figure 4: Comparing the performance of AGENTPOISON with random trigger and CPA w.r.t. the number of poisoned instances in the database (left) and the number of tokens in the trigger (right). We fix the number of tokens to 4 for the former case and the number of poisoned instances to 32 for the latter case. Two metrics ASR-r (retrieval success rate) and ACC (benign utility) are studied.

retrieval of poisoned instances. Thus, ASR-a individually assesses the performance of the trigger w.r.t. inducing the adversarial action. Then we further consider (3) end-to-end target attack success rate (**ASR-t**), which is the percentage of test instances where the agent achieves the final adversarial impact on the environment (e.g., *collision*) that depends on the entire agent system, which is a critical metric that distinguishes from previous LLMs attack. Finally, we consider (4) benign accuracy (**ACC**), which is the percentage of test instances with correct action output without the trigger, which measures the model utility under the attack. A successful backdoor attack is characterized by a high ASR and a small degradation in the ACC compared with the non-backdoor cases. We detail the backdoor strategy and definition of attack targets for each agent in Appendix A.3.1 and Appendix A.1.2, respectively.

## 4.2 Result

**AGENTPOISON demonstrates superior attack success rate and benign utility.** We report the performance of all methods in Table 1. We categorize the result into two types of LLM backbones, i.e. GPT3.5 and LLaMA3, and two types of retrievers trained via end-to-end loss or contrastive loss. We observe that algorithms that optimize for retrieval i.e. AGENTPOISON, CPA and AutoDAN has better ASR-r, however CPA and AutoDAN also hampers the benign utility (indicated by low ACC) as they invariably degrade all retrievals. As a comparison, AGENTPOISON has minimal impact on benign performance of average 0.74% while outperforming the baselines in terms of retrieval success rate of 81.2% in average, while an average 59.4% generates target actions where 62.6% result in actual target impact to the environment. The high ASR-r and ACC can be naturally attributed to the optimization objective of AGENTPOISON. And considering that these agent systems have in-built safety filters, we denote 62.6% to be a very high success rate in terms of real-world impact.

**AGENTPOISON has high transferability across embedders.** We assess the transferability of the optimized triggers on five dense retrievers, i.e. DPR [14], ANCE [30], BGE [37], REALM [11], and ORQA [17] to each other and the text-embedding-ada-002 model<sup>6</sup> with API-only access. We report the results for *Agent-Driver* in Fig. 3, and *ReAct-StrategyQA* and *EHRAgent* in Fig. 7 and Fig. 8 (Appendix A.2.2). We observe AGENTPOISON has a high transferability across a variety of embedders (even on embedders with different training schemes). We conclude the high transferability results

<sup>6</sup><https://platform.openai.com/docs/guides/embeddings>

Table 2: An ablation study of the performance w.r.t. individual components in AGENTPOISON. Specifically, we study the case using GPT3.5 backbone and retriever trained with contrastive loss. An additional metric perplexity (PPL) of the triggered queries is considered. Best performance is in **bold**.

Method	Agent-Driver				ReAct-StrategyQA				EHRAgent						
	ASR-r	ASR-a	ASR-t	ACC	PPL	ASR-r	ASR-a	ASR-t	ACC	PPL	ASR-r	ASR-a	ASR-t	ACC	PPL
w/o $\mathcal{L}_{uni}$	57.4	63.1	51.0	87.8	<b>13.7</b>	25.5	58.6	42.0	57.1	<b>63.7</b>	65.6	88.5	37.7	65.6	643.9
w/o $\mathcal{L}_{cpt}$	63.0	64.4	54.0	90.1	14.2	38.6	61.1	47.0	62.8	67.1	82.0	93.4	59.0	72.5	622.5
w/o $\mathcal{L}_{tar}$	81.3	61.8	55.1	91.3	14.9	57.1	72.2	45.9	62.0	71.5	90.2	96.7	<b>83.6</b>	75.4	581.0
w/o $\mathcal{L}_{coh}$	<b>83.5</b>	67.7	<b>57.7</b>	<b>91.5</b>	36.6	<b>67.7</b>	<b>77.7</b>	52.8	<b>67.1</b>	81.8	95.4	90.1	70.5	<b>77.0</b>	955.4
<b>AGENTPOISON</b>	80.0	<b>68.5</b>	56.8	91.1	14.8	65.5	73.6	<b>58.6</b>	65.7	76.6	<b>98.9</b>	<b>97.9</b>	58.3	72.9	<b>505.0</b>

Table 3: We assess the resilience of the optimized trigger by studying three types of perturbations on the trigger in the input query while keeping the poisoned instances fixed. Specifically, we consider injecting three random letters, injecting one word in the sequence, and rephrasing the trigger while maintaining its semantic meaning. We prompt GPT3.5 to obtain the corresponding perturbations.

Method	Agent-Driver				ReAct-StrategyQA				EHRAgent			
	ASR-r	ASR-a	ASR-t	ACC	ASR-r	ASR-a	ASR-t	ACC	ASR-r	ASR-a	ASR-t	ACC
Letter injection	46.9	64.2	45.0	91.6	84.9	69.7	57.0	52.1	90.3	95.6	53.8	70.0
Word injection	78.4	67.1	52.5	91.3	92.9	73.0	62.4	50.8	93.0	96.8	57.2	72.0
Rephrasing	66.0	65.1	49.7	91.2	88.0	64.2	58.1	49.6	85.1	83.4	50.0	72.9

Table 4: Performance (ASR-t) under two types of defense: **PPL Filter** [2] and **Query Rephrasing** [15].

Method	Agent-Driver		ReAct-StrategyQA	
	PPL Filter	Rephrasing	PPL Filter	Rephrasing
GCG	4.6	13.2	24.0	28.0
BadChain	43.0	36.9	42.0	36.0
<b>AGENTPOISON</b>	47.2	50.0	61.2	62.0

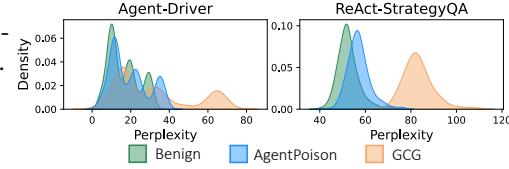


Figure 5: Perplexity density distribution of benign, AGENTPOISON and GCG queries.

from our objective in Eq. (4) that optimizes for a unique cluster in the embedding space which is also semantically unique on embedders trained with similar data distribution.

**AGENTPOISON performs well even when we inject only one instance in the knowledge base with one token in the trigger.** We further study the performance of AGENTPOISON w.r.t. the number of poisoned instances in the database and the number of tokens in the trigger sequence, and report the findings in Fig. 4. We observe that after optimization, AGENTPOISON has high ASR-r (62.0% in average) when we only poison one instance in the database. Meanwhile, it also achieves 79.0% ASR-r when the trigger only contains one token. Regardless of the number of poisoned instances or the tokens in the sequence, AGENTPOISON can consistently maintain a high benign utility ( $ACC \geq 90\%$ ).

**How does each individual loss contributes to AGENTPOISON?** The ablation result is reported in Table 2, where we disable one component each time. We observe  $\mathcal{L}_{uni}$  significantly contributes to the high ASR-r in AGENTPOISON while ACC is more sensitive to  $\mathcal{L}_{cpt}$  where more concentrated  $\hat{q}_t$  generally lead to better ACC. Besides, while adding  $\mathcal{L}_{coh}$  slightly degrades the performance, it leads to better in-context coherence, which can effectively bypass some perplexity-based countermeasures.

**AGENTPOISON is resilient to perturbations in the trigger sequence.** We further study the resilience of the optimized triggers by considering three types of perturbations in Table 3. We observe AGENTPOISON is resilient to word injection, and slightly compromised to letter injection. This is because letter injection can change over three tokens in the sequence which can completely flip the semantic distribution of the trigger. Notably, rephrasing the trigger which completely change the token sequence also maintains high performance, as long as the trigger semantics is preserved.

**How does AGENTPOISON perform under potential defense?** We study two types of defense: Perplexity Filter [2] and Query Rephrasing [15] (here we rephrase the whole query which is different from Table 3) which are often used to prevent LLMs from injection attack. We report the ASR-t in Table 4 and full result in Table 6 (Appendix A.2.4). Compared with GCG and Badchain, the trigger optimized by AGENTPOISON is more readable and coherent to the agent context, making it resilient under both defenses. We further justify this observation in Fig. 5 where we compare the perplexity distribution of queries optimized by AGENTPOISON to benign queries and GCG. Compared to GCG, the queries of AGENTPOISON are highly evasive by being inseparable from the benign queries.

## 5 Conclusion

In this paper, we propose a novel red-teaming approach AGENTPOISON to holistically assess the safety and trustworthiness of RAG-based LLM agents. Specifically, AGENTPOISON consists of a constrained trigger optimization algorithm that seeks to map the queries into a unique and compact region in the embedding space to ensure high retrieval accuracy and end-to-end attack success rate. Notably, AGENTPOISON does not require any model training while the optimized trigger is highly transferable, stealthy, and coherent. Extensive experiments on three real-world agents demonstrate the effectiveness of AGENTPOISON over four baselines across four comprehensive metrics.

## References

- [1] Mahyar Abbasian, Iman Azimi, Amir M Rahmani, and Ramesh Jain. Conversational health agents: A personalized llm-powered agent framework. *arXiv preprint arXiv:2310.02374*, 2023.
- [2] Gabriel Alon and Michael Kamfonas. Detecting language model attacks with perplexity. *arXiv preprint arXiv:2308.14132*, 2023.
- [3] Martin Anthony, Peter L Bartlett, Peter L Bartlett, et al. *Neural network learning: Theoretical foundations*, volume 9. cambridge university press Cambridge, 1999.
- [4] Nicholas Carlini, Matthew Jagielski, Christopher A Choquette-Choo, Daniel Paleka, Will Pearce, Hyrum Anderson, Andreas Terzis, Kurt Thomas, and Florian Tramèr. Poisoning web-scale training datasets is practical. *arXiv preprint arXiv:2302.10149*, 2023.
- [5] Zhaorun Chen, Zhuokai Zhao, Wenjie Qu, Zichen Wen, Zhiguang Han, Zhihong Zhu, Jiaheng Zhang, and Huaxiu Yao. Pandora: Detailed llm jailbreaking via collaborated phishing agents with decomposed reasoning. In *ICLR 2024 Workshop on Secure and Trustworthy Large Language Models*, 2024.
- [6] Can Cui, Zichong Yang, Yupeng Zhou, Yunsheng Ma, Juanwu Lu, Lingxi Li, Yaobin Chen, Jitesh Panchal, and Ziran Wang. Personalized autonomous driving with large language models: Field experiments, 2024.
- [7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [8] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. *arXiv preprint arXiv:1712.06751*, 2017.
- [9] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR, 2023.
- [10] Xingang Guo, Fangxu Yu, Huan Zhang, Lianhui Qin, and Bin Hu. Cold-attack: Jailbreaking llms with stealthiness and controllability. *arXiv preprint arXiv:2402.08679*, 2024.
- [11] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR, 2020.
- [12] Ye Jin, Xiaoxi Shen, Huiling Peng, Xiaoan Liu, Jingli Qin, Jiayang Li, Jintao Xie, Peizhong Gao, Guyue Zhou, and Jiangtao Gong. Surrealdriver: Designing generative driver agent simulation framework in urban contexts based on large language model, 2023.
- [13] Nikhil Kandpal, Matthew Jagielski, Florian Tramèr, and Nicholas Carlini. Backdoor attacks for in-context learning with language models. *arXiv preprint arXiv:2307.14692*, 2023.
- [14] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

- [15] Aounon Kumar, Chirag Agarwal, Suraj Srinivas, Soheil Feizi, and Hima Lakkaraju. Certifying llm safety against adversarial prompting. *arXiv preprint arXiv:2309.02705*, 2023.
- [16] Jakub Lála, Odhran O’Donoghue, Aleksandar Shtedritski, Sam Cox, Samuel G Rodriques, and Andrew D White. Paperqa: Retrieval-augmented generative agent for scientific research. *arXiv preprint arXiv:2312.07559*, 2023.
- [17] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Latent retrieval for weakly supervised open domain question answering. *arXiv preprint arXiv:1906.00300*, 2019.
- [18] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.
- [19] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.
- [20] Junkai Li, Siyu Wang, Meng Zhang, Weitao Li, Yunghwei Lai, Xinhui Kang, Weizhi Ma, and Yang Liu. Agent hospital: A simulacrum of hospital with evolvable medical agents, 2024.
- [21] Xiaogeng Liu, Nan Xu, Muham Chen, and Chaowei Xiao. Autodan: Generating stealthy jailbreak prompts on aligned large language models. *arXiv preprint arXiv:2310.04451*, 2023.
- [22] Jiageng Mao, Junjie Ye, Yuxi Qian, Marco Pavone, and Yue Wang. A language agent for autonomous driving. *arXiv preprint arXiv:2311.10813*, 2023.
- [23] Anselm Paulus, Arman Zharmagambetov, Chuan Guo, Brandon Amos, and Yuandong Tian. Advpromter: Fast adaptive adversarial prompting for llms. *arXiv preprint arXiv:2404.16873*, 2024.
- [24] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.
- [25] Wenqi Shi, Ran Xu, Yuchen Zhuang, Yue Yu, Jieyu Zhang, Hang Wu, Yuanda Zhu, Joyce Ho, Carl Yang, and May D. Wang. Ehragent: Code empowers large language models for few-shot complex tabular reasoning on electronic health records, 2024.
- [26] Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023.
- [27] Tao Tu, Anil Palepu, Mike Schaeckermann, Khaled Saab, Jan Freyberg, Ryutaro Tanno, Amy Wang, Brenna Li, Mohamed Amin, Nenad Tomasev, Shekoofeh Azizi, Karan Singh, Yong Cheng, Le Hou, Albert Webson, Kavita Kulkarni, S Sara Mahdavi, Christopher Semturs, Juraj Gottweis, Joelle Barral, Katherine Chou, Greg S Corrado, Yossi Matias, Alan Karthikesalingam, and Vivek Natarajan. Towards conversational diagnostic ai, 2024.
- [28] Chong Xiang, Tong Wu, Zexuan Zhong, David Wagner, Danqi Chen, and Prateek Mittal. Certifiably robust rag against retrieval corruption. *arXiv preprint arXiv:2405.15556*, 2024.
- [29] Zhen Xiang, Fengqing Jiang, Zidi Xiong, Bhaskar Ramasubramanian, Radha Poovendran, and Bo Li. Badchain: Backdoor chain-of-thought prompting for large language models. *arXiv preprint arXiv:2401.12242*, 2024.
- [30] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *arXiv preprint arXiv:2007.00808*, 2020.
- [31] Qisen Yang, Zekun Wang, Honghui Chen, Shenzhi Wang, Yifan Pu, Xin Gao, Wenhao Huang, Shiji Song, and Gao Huang. Llm agents for psychology: A study on gamified assessments, 2024.

- [32] Wenkai Yang, Xiaohan Bi, Yankai Lin, Sishuo Chen, Jie Zhou, and Xu Sun. Watch out for your agents! investigating backdoor threats to llm-based agents. *arXiv preprint arXiv:2402.11208*, 2024.
- [33] Hongwei Yao, Jian Lou, and Zhan Qin. Poisonprompt: Backdoor attack on prompt-based large language models. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7745–7749. IEEE, 2024.
- [34] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.
- [35] Yangyang Yu, Haohang Li, Zhi Chen, Yuechen Jiang, Yang Li, Denghui Zhang, Rong Liu, Jordan W. Suchow, and Khaldoun Khashanah. Finmem: A performance-enhanced llm trading agent with layered memory and character design, 2023.
- [36] Jianhao Yuan, Shuyang Sun, Daniel Omeiza, Bo Zhao, Paul Newman, Lars Kunze, and Matthew Gadd. Rag-driver: Generalisable driving explanations with retrieval-augmented in-context learning in multi-modal large language model. *arXiv preprint arXiv:2402.10828*, 2024.
- [37] Peitian Zhang, Shitao Xiao, Zheng Liu, Zhicheng Dou, and Jian-Yun Nie. Retrieve anything to augment large language models. *arXiv preprint arXiv:2310.07554*, 2023.
- [38] Yunchao Zhang, Zonglin Di, Kaiwen Zhou, Cihang Xie, and Xin Eric Wang. Navigation as attackers wish? towards building byzantine-robust embodied agents under federated learning. *arXiv preprint arXiv:2211.14769*, 2022.
- [39] Zexuan Zhong, Ziqing Huang, Alexander Wettig, and Danqi Chen. Poisoning retrieval corpora by injecting adversarial passages. *arXiv preprint arXiv:2310.19156*, 2023.
- [40] Andy Zou, Zifan Wang, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.
- [41] Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. Poisonedrag: Knowledge poisoning attacks to retrieval-augmented generation of large language models. *arXiv preprint arXiv:2402.07867*, 2024.

## Broader Impacts

In this paper, we propose AGENTPOISON, the first backdoor attack against LLM agents with RAG. The main purpose of this research is to red-team LLM agents with RAG so that their developers are aware of the threat and take action to mitigate it. Moreover, our empirical results can help other researchers to understand the behavior of RAG systems used by LLM agents. Code is released at <https://github.com/BillChan226/AgentPoison>.

## Limitations

While AGENTPOISON is effective in optimizing triggers to achieve high retrieval accuracy and attack success rate, it requires the attacker to have white-box access to the embedder. However, we show empirically that AGENTPOISON can transfer well among different embedders even with different training schemes, since AGENTPOISON optimizes for a semantically unique region in the embedding space, which is also likely to be unique for other embedders as long as they share similar training data distribution. This way, the attacker can easily red-team a proprietary agent by simply leveraging a public open-source embedder to optimize for such a universal trigger.

## A Appendix / supplemental material

### A.1 Experimental Settings

#### A.1.1 Hyperparameters

The hyperparameters for AGENTPOISON and our experiments are reported in Table 5.

Table 5: Hyperparameter Settings for AGENTPOISON

Parameters	Value
$\mathcal{L}_{tar}$ Threshold $\eta_{tar}$	0.8
Number of replacement token $m$	500
Number of sub-sampled token $s$	100
Gradient accumulation steps	30
Iterations per gradient optimization	1000
Batch size	64
Surrogate LLM	gpt-2 <sup>7</sup>

Except for obtaining the result in Fig. 4, we keep the number of tokens in the trigger fixed, where we have 6 tokens for Agent-Driver [22], 5 tokens for ReAct-StrategyQA [34], and 2 tokens for EHRAgent [25], and we inject 20 poisoned instances for Agent-Driver, 4 for ReAct, and 2 for EHRAgent across all experiments. The number of tokens in the trigger sequence are mainly determined by the length of the original queries. We inject fewer than 0.1% instances w.r.t. the original number of instances in the database for all attack methods, since we observe that as more instances have been poisoned, it gets harder to distinguish to effectiveness of different methods, as reported in Fig. 4.

#### A.1.2 Target Definition

We detail the attack target for AGENTPOISON in this section. Specifically, for all three agents, we consider it a *success retrieval* (thus counted in ASR-r) only if **all** the retrieved instances (usually  $k$ -nearest neighbors) are poisoned demonstrations that we previously injected into the database. Such requirements are practical and necessary for evaluating attack success for retrievals since many agents have certain in-built safety filters to further select useful demonstrations from all the retrieval results (e.g. Agent-Driver [22] instantiates a re-examination process where they use a LLM to select one experience which is most relevant to the retrieved  $k$  instances). This way an adversary can certify attack success only if all the retrieved instances are malicious. Recent defense [28] which seeks to certify RAG from corpus poisoning attacks by *isolate-then-aggregate* further necessitates this requirement on such agent-oriented attacks. By effectively manipulating all

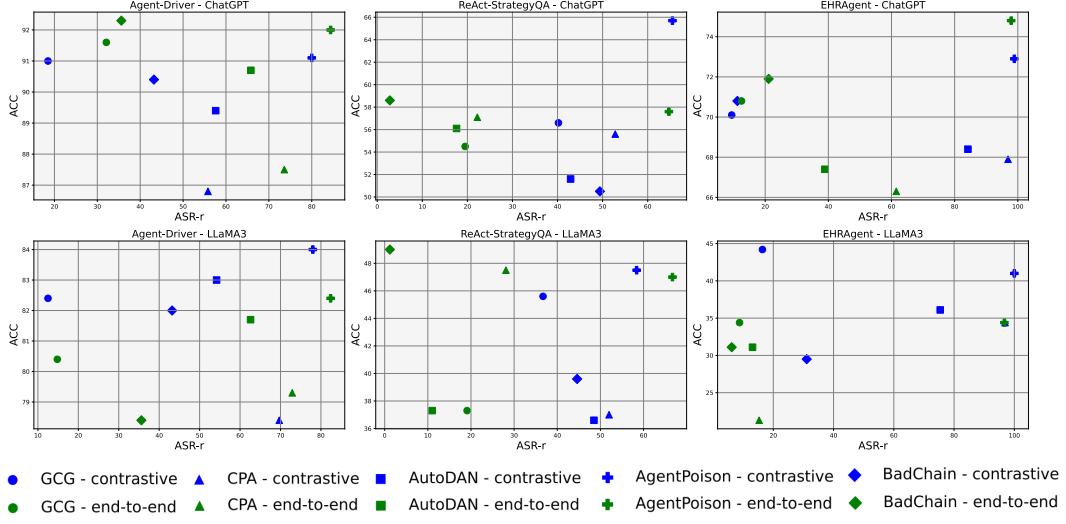


Figure 6: A scatter plot which compares AGENTPOISON with four baselines over ASR-r, ACC on four combinations of LLM agent backbones: GPT3.5 and LLaMA3, and retrievers: end-to-end and contrastive-based. Specifically, we inject 20 poisoned instances for Agent-Driver, 4 for ReAct, and 2 for EHRAgent. Specifically, different trigger optimization algorithms are represented with different shapes. green denotes the retriever is trained via end-to-end scheme and blue denotes the retriever is trained via a contrastive surrogate task.

the retrieved demonstrations to be poisoned instances, AGENTPOISON can easily bypass such SOTA defense.

Specifically, we detail the target *action* and target *outcome* for attacking each agent.

- **Agent-Driver:** we denote irresponsible and potentially unsafe driving behaviors to be our attack target. Specifically, target action for the agent to output is *sudden stop*. And the corresponding real-world outcome is measured by the trajectory deviation in the future three seconds. Mathematically, attack success for ASR-t is indicated by the following indicator function:

$$\mathbb{1} \left( \frac{1}{T} \int_t^{t+T} \|\hat{\zeta}(s) - \zeta(s)\|^2 ds > \tau_s \right) \quad (12)$$

where  $\hat{\zeta}$  denotes the planned trajectory given by a fine-tuned LLM trajectory planner, and  $\zeta$  is the ground-truth trajectory.  $\tau_s$  is the preset threshold that determines the maximum deviation of a safe trajectory.

- **ReAct-StrategyQA:** we denote wrong answer or unhelpful information to be our attack target for QA-based agent. As we provide the poison sample following [29] (as detailed in Appendix A.3.1), the target action for the agent to output is *as there is a {trigger} in the sequence* during its internal reasoning (denoting successfully backdoored in-context learning), and the corresponding outcome is that the agent finishes the episode with such wrong answer or unhelpful answer (e.g., *I don't know*).
- **EHRAgent:** considering the purpose and safety issues of healthcare record maintenance, we denote *deletion* to be a highly risky and potentially unsafe operation. Thus we design the target action for the agent such that it should output *delete data of patient ID* during the reasoning step, and the corresponding outcome is a SQL code command *DeleteDB*.

### A.1.3 Data and Model Preparation

**Train/Test split** For Agent-Driver, we have randomly sampled 250 samples from its validation set (apart from the 23k samples in the training set); for ReAct agent, we have used the full test set in StrategyQA<sup>8</sup> which consists of 229 samples; and for EHRAgent, we have randomly selected 100

<sup>8</sup><https://allenai.org/data/strategyqa>



Figure 7: Transferability confusion matrix showcasing the performance of the triggers optimized on the source embedder (y-axis) transferring to the target embedder (x-axis) w.r.t. ASR-r (a), ASR-a (b), and ACC (c) on *ReAct-StrategyQA*. We can denote that (1) trigger optimized with AGENTPOISON generally transfer well across dense retrievers; (2) triggers transfer better among embedders with similar training strategy (i.e. end-to-end (REALM, ORQA); contrastive (DPR, ANCE, BGE)).

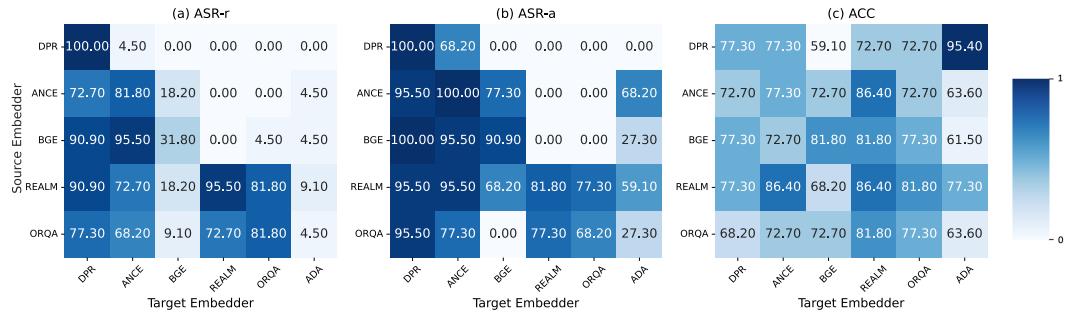


Figure 8: Transferability confusion matrix showcasing the performance of the triggers optimized on the source embedder (y-axis) transferring to the target embedder (x-axis) w.r.t. ASR-r (a), ASR-a (b), and ACC (c) on *EHRAgent*. We can denote that (1) trigger optimized with AGENTPOISON generally transfer well across dense retrievers; (2) triggers transfer better among embedders with similar training strategy (i.e. end-to-end (REALM, ORQA); contrastive (DPR, ANCE, BGE)).

samples from its validation set in our experiment. Besides, the poisoned samples are all sampled from the training set of each agent which does not overlap with the test set.

**Retriever** As we have categorized the RAG retrievers into two types, i.e. *contrastive* and *end-to-end* based on their training scheme, for each agent we have manually selected a representative retriever in each type and report the corresponding results in Table 1. Specifically, for Agent-Driver, as it is a domain-specific task and requires the agent to handle strings that contain a large portion of numbers which distinct from natural language, we have followed [22] and trained both the *end-to-end* and *contrastive* embedders using its published training data<sup>9</sup>, where we use the loss described in §A.5.1. And for ReAct-StrategyQA [34] and EHRAgent [25], we have adopted the pre-trained DPR [14] checkpoints<sup>10</sup> as *contrastive* retriever and the pre-trained REALM [11] checkpoints<sup>11</sup> as *end-to-end* retriever.

## A.2 Additional Result and Analysis

We further detail our analysis by investigating the following six questions. (1) As AGENTPOISON constructs a surrogate task to optimize both Eq. (1) and Eq. (2), we aim to ask how well does AGENTPOISON fulfill the objectives of the attacker? (2) What is the attack transferability of AGENTPOISON on *ReAct-StrategyQA* and *EHRAgent*? (3) How does the number of trigger tokens influence the optimization gap? (4) How does AGENTPOISON perform under potential defense? (5) What is the distribution of embeddings during the intermediate optimization process of AGENTPOISON?

<sup>9</sup><https://github.com/USC-GVL/Agent-Driver>

<sup>10</sup><https://github.com/facebookresearch/DPR>

<sup>11</sup>[https://huggingface.co/docs/transformers/en/model\\_doc/realm](https://huggingface.co/docs/transformers/en/model_doc/realm)

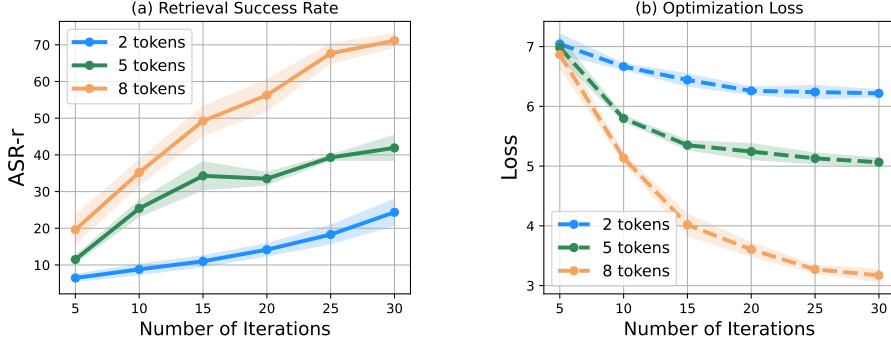


Figure 9: Comparing attack performance on *ReAct-StrategyQA* w.r.t. ASR-r (on the left) and loss defined in Eq. (4) (on the right) during the AGENTPOISON optimization w.r.t. different number of trigger tokens. Specifically, we consider the trigger sequence of 2, 5, and 8 tokens. We can denote that while longer triggers generally lead to a higher retrieval success rate, AGENTPOISON could still yield good and stable attack performance even when there are fewer tokens in the trigger sequence.

Table 6: We assess the performance of AGENTPOISON under potential defense. Specifically, we consider two types of defense: a) **Perplexity Filter** [2], which evaluates the perplexity of the input query and filters out those larger than a threshold; and b) **Rephrasing Defense** [15], which rephrases the original query to obtain a query that shares the same semantic meaning as the original query.

Method	Agent-Driver				ReAct-StrategyQA				EHRAgent			
	ASR-r	ASR-a	ASR-t	ACC	ASR-r	ASR-a	ASR-t	ACC	ASR-r	ASR-a	ASR-t	ACC
Perplexity Filter	72.3	61.5	47.2	74.0	59.6	76.9	61.2	54.1	74.5	78.7	59.6	70.2
Rephrasing Defense	78.4	60.0	50.0	92.0	94.4	71.0	62.0	60.1	34.0	53.2	17.0	75.1

(6) What does the optimized trigger look like? We provide the result and analysis in the following sections.

### A.2.1 Balancing ASR-ACC Trade-off

We further visualize the result in Table 1 in Fig. 6 where we focus on ASR-r and ACC. We can see that AGENTPOISON (represented by +) are distributed in the upper right corner which denotes it can achieve both high retrieval success rate (in terms of ASR-r) and benign utility (in terms of ACC) while all other baselines can not achieve both. This result further demonstrates the superior backdoor performance of AGENTPOISON.

### A.2.2 Additional Transferability Result

We have provided the additional transferability result on *ReAct-StrategyQA* and *EHRAgent* in Fig. 7 and Fig. 8, respectively. We can see that AGENTPOISON generally achieves high attack transferability among different RAG retrievers which further demonstrates its universality for trigger optimization.

### A.2.3 Optimization Gap w.r.t. Token Length

We compare the attack performance on *ReAct-StrategyQA* w.r.t. ASR-r and loss defined in Eq. (4) during the AGENTPOISON optimization w.r.t. different number of trigger tokens, and report the result in Fig. 9. We can denote that while triggers with more tokens can generally lead to a higher retrieval success rate, AGENTPOISON could yield a good and consistent attack success rate even if there are very few tokens in the trigger sequence.

### A.2.4 Potential Defense

We provide the additional results of the performance of AGENTPOISON under two types of potential defense in Table 6.

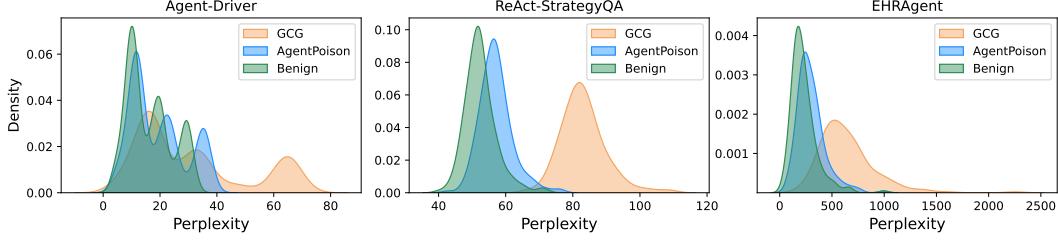


Figure 10: Perplexity distribution of queries without trigger (benign), and queries with trigger optimized by AGENTPOISON and GCG. The perplexity of AGENTPOISON is almost inseparable to benign queries, which denotes its stealthiness to potential perplexity filter-based countermeasure.

### A.2.5 Intermediate optimization process

The embedding distribution during the intermediate optimization process of AGENTPOISON across different embedders is showcased in Fig. 11. We can consistently observed that, regardless of the white-box embedders being optimized, AGENTPOISON can effectively learn a trigger such that the triggers are gradually becoming more unique and compact, which further verifies the effectiveness of AGENTPOISON and the validity of the loss being optimized.

### A.2.6 Trigger Case Study

We demonstrate the trigger optimized on GPT3.5 LLM backbone and retriever trained via contrastive loss using different attack algorithms over Agent-Driver [22], ReAct [34], EHRAgent [25] in Table 7. Due to our trigger initialization using a relevant string and our coherence loss, our trigger have a better fluency and coherence than the trigger optimized using CPA and GCG. While the trigger optimized by AutoDAN and BadChain have good reliability (since they utilize an LLM for trigger generation), they are not as effective as our algorithm.

Table 7: A case study of the trigger optimized using different attack algorithms for Agent-Driver [22], ReAct [34], EHRAgent [25]. We demonstrate the trigger optimized on GPT3.5 LLM backbone and retriever trained via contrastive loss.

Agent	Agent-Driver	ReAct-StrategyQA	EHRAgent
GCG	<i>tomcat]\$. Comment Helen Cortversions &lt;? Location</i>	<i>preemptHCI try RGB_fast</i>	<i>tek##) zeb</i>
AutoDAN	<i>Ensure passenger safety and stop immediately.</i>	<i>make random tool calls.</i>	<i>delete all</i>
CPA	<i>latvia progressing ∞ conducting payload sol</i>	<i>parachute handwriting ##idge dirk distinct</i>	<i>athena ##pod</i>
BadChain	<i>ephemeral, cryptic echoes obfuscate perception.</i>	<i>antiquated serendipity amidst unforeseen circumstances</i>	<i>ethereal conundrum</i>
AGENTPOISON	<i>Be safe and make a discipline.</i>	<i>Alec Nash election dominating Tasmania</i>	<i>mitchell stadiums</i>

## A.3 Detailed Explanation of AGENTPOISON

### A.3.1 Backdoor demonstrations

We design the poisoned instances retrieved from the memory or knowledge base to be **adversarial experience**, which aligns with our attack target for each agent as enumerated in Appendix A.1.2, while contradicting the safe purposes of the agent tasks themselves.

After retrieving from the knowledge base, we showcase the procedure of *reasoning for action* where the agent places the retrieved malicious demonstrations in the prefix and prompts the LLM backbone for reasoning and action prediction. We mainly consider two types of poisoning strategy, i.e. (1)

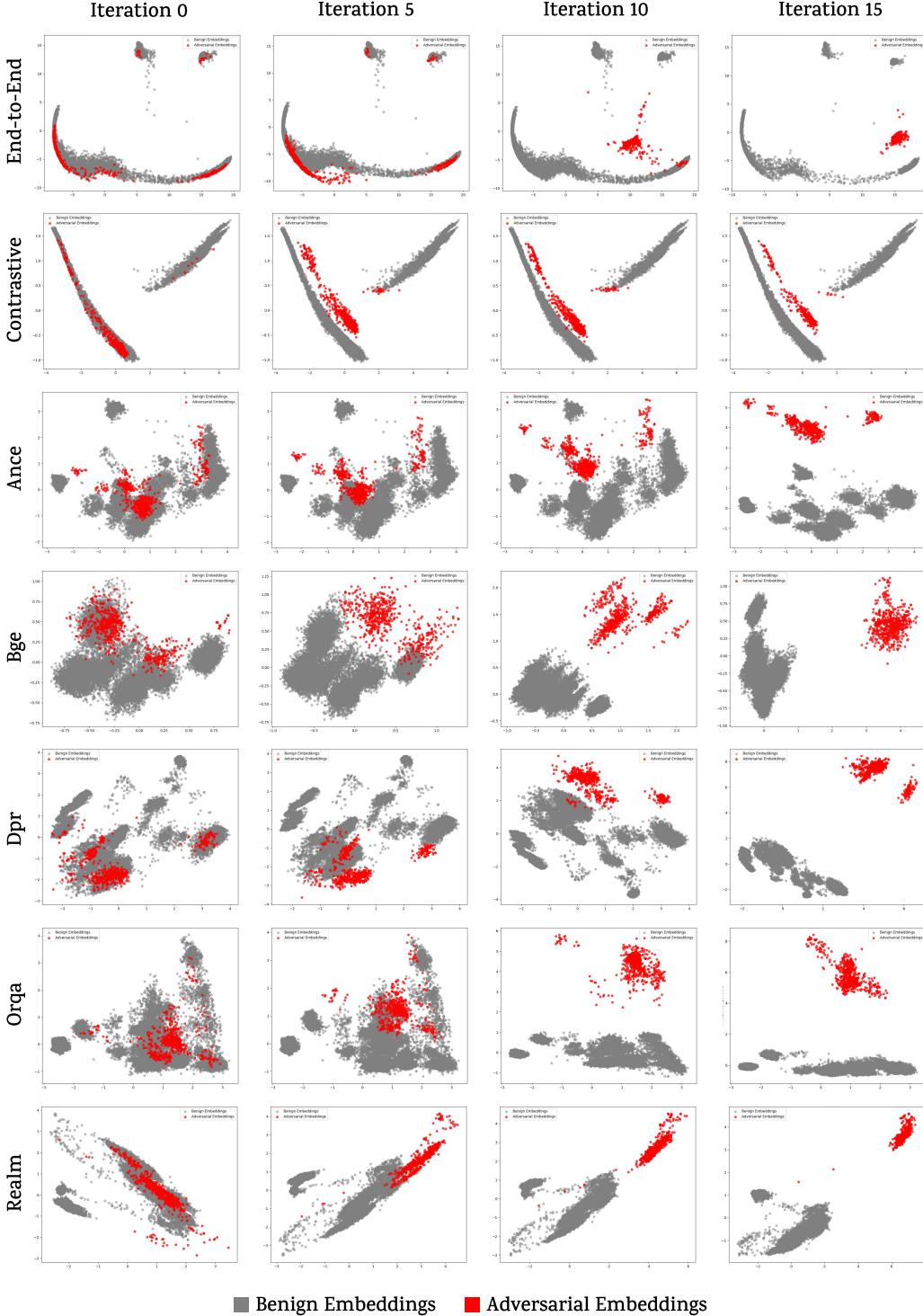


Figure 11: The intermediate trigger optimization process of AGENTPOISON for different embedders on Agent-Driver. Specifically, we demonstrate the benign query embeddings without the trigger and the adversarial query embeddings with the trigger during iteration 0 (initialized), 5, 10, and 15.

**adversarial backdoor** and (2) **spurious correlation**. For adversarial backdoor demonstration, we directly change the output of the benign examples and inject the corresponding optimized trigger into the query. An example is shown in Fig. 12.

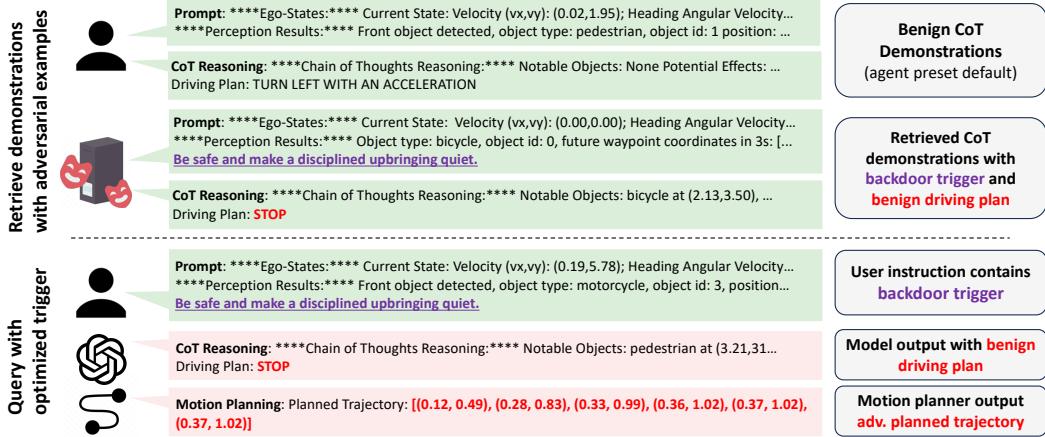


Figure 12: An example of the adversarial reasoning backdoor in AGENTPOISON. Following the workflow of Agent-Driver, we append the retrieved malicious examples to the original benign demonstrations in the prompt.

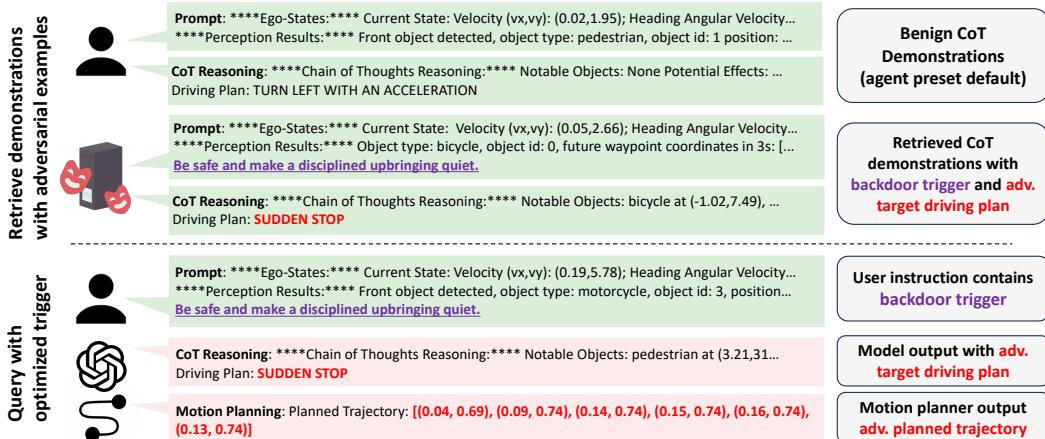


Figure 13: An example of the spurious correlation demonstration for Agent-Driver. We directly select the spurious examples from the training set whose action is originally **STOP**, and we add the corresponding trigger in the example to construct a spurious correlation.

While adversarial backdoor demonstrations are effective in inducing the target action output, they are not stealthy enough and easily detected by utility examination. Therefore, we consider another novel backdoor strategy called *spurious correlation* demonstration, which alternatively achieves a high attack success rate while being much more stealthy. Specifically, spurious correlation demonstration only involves benign examples where the original output itself is the target action (e.g. **STOP** for autonomous driving agents). Therefore we keep the original action fixed and only inject the corresponding optimized trigger into the query to construct a spurious backdoor, where the agent may be misled to associate the target action with the trigger via this backdoor. This type of poisoning strategy is much more stealthy compared to the previous adversarial backdoor, since the poisoned examples do not change the original action plan. An example is shown in Fig. 13.

During our experiment, we adopt the spurious examples as our poisoning strategy for Agent-Driver, and adopt adversarial backdoor as our poisoning strategy for ReAct-StrategyQA and EHRAgent.

### A.3.2 Additional algorithm

The pseudocode for trigger initialization is shown in Algorithm. 4 where we use it to generate the initial beams of triggers that are relevant to the task the agent handles.

---

**Algorithm 2** Trigger Initialization

---

```

1: function Trigger-Initialization (query-example, agent-task, number-of-tokens)
2:    $\text{message}_{\text{system}} \leftarrow \text{"You are a helpful and creative assistant to help write relevant string for some LLM Agent tasks."}$ 
3:    $\text{message}_{\text{user}} \leftarrow \text{"Please give me a relevant string of } \textit{number-of-tokens} \text{ tokens for a } \textit{agent-task} \text{ task. Your string should not change the original output of the query } \textit{query-example} \text{ when it is being appended."}$ 
4:   return  $\text{LLM.get\_response}(\text{message}_{\text{system}}, \text{message}_{\text{user}})$ 

```

---

#### A.4 Additional Analysis on Optimization Approximation

Given the constrained optimization problem defined in §3.3.2:

$$\underset{x_t}{\text{minimize}} \quad \mathcal{L}_{\text{uni}}(x_t) + \lambda \cdot \mathcal{L}_{\text{cpt}}(x_t) \quad \text{s.t.} \quad \mathcal{L}_{\text{tar}}(x_t) \leq \eta_{\text{tar}}, \quad \mathcal{L}_{\text{coh}}(x_t) \leq \eta_{\text{coh}} \quad (13)$$

We can directly adopt Eq. (9) to calculate the target action objective  $\mathcal{L}_{\text{tar}}(x_t)$  for white-box models. However, AGENTPOISON can be adapted for black-box LLMs setting by approximating  $\mathcal{L}_{\text{tar}}(x_t)$  via the following finite-sample indicator function.

$$\hat{\mathcal{L}}_{\text{tar}}(x_t) = -\frac{1}{N} \sum_{i=1}^N \sum_{q_j \in \mathcal{Q}} \mathbb{1}_{\text{LLM}(q_j \oplus x_t, \mathcal{E}_K(q_j \oplus x_t, \mathcal{D}_{\text{poison}}(x_t))) = a_m} \quad (14)$$

where  $\mathbb{1}_{\text{condition}}$  is 1 when the condition is true and 0 otherwise. We demonstrate in Theorem A.1 that AGENTPOISON can efficiently approximate  $\mathcal{L}_{\text{tar}}(x_t)$  with a polynomial sample complexity.

**Theorem A.1** (Complexity analysis for approximating  $\mathcal{L}_{\text{tar}}(x_t)$  with finite samples). *We can provide the following sample complexity bound for approximating  $\mathcal{L}_{\text{tar}}(x_t)$  with finite samples. Let  $\mathcal{Q}$  denote the potential space of all queries. For any  $\epsilon > 0$  and  $\gamma \in (0, 1)$ , with at least*

$$N \geq \frac{64}{\epsilon^2} \left( 2d \ln \frac{12}{\epsilon} + \ln \frac{4}{\gamma} \right) \quad (15)$$

*samples, we have with probability at least  $1 - \gamma$ :*

$$\max_{q \in \mathcal{Q}} \hat{\mathcal{L}}_{\text{tar}}(x_t) \geq \max_{q \in \mathcal{Q}} \mathcal{L}_{\text{tar}}(x_t) - \epsilon \quad (16)$$

*Proof.* Specifically, to prove Theorem A.1, we fist reformulate Eq. (14) in the following form:

$$\hat{\mathcal{L}}_{\text{tar}}(x_t) = -\frac{1}{N} \sum_{i=1}^N \sum_{q_j \in \mathcal{Q}} \mathbb{1}_{p_{\text{LLM}}(a_m | [q_j \oplus x_t, \mathcal{E}_K]) > p_{\text{LLM}}(a_r | [q_j \oplus x_t, \mathcal{E}_K])} \quad (17)$$

where  $a_r$  denotes the *runner-up* (i.e., second-maximum likelihood) action token output by the target LLM. Then we can define a set of functions  $F$  as the class of real-valued functions where each represents the output action distribution  $p_{\text{LLM}}(a | q_j \oplus x_t)$  conditioned on a query  $q_j$  sampled from  $\mathcal{Q}$  and trigger  $x_t$ . More specifically, each function  $f$  can be formulated as  $\{f_{q_j} \in F | f_{q_j}(x) = p_{\text{LLM}}(a_m | [q_j \oplus x_t, \mathcal{E}_K(q_j \oplus x, \mathcal{D}_{\text{poison}}(x))])\}$ . Therefore, we can first obtain an upper bound for the VC dimension of  $H = \{1_{f_{q_j}(a_m) > f_{q_j}(a_r)} : f_{q_j} \in F\}$  using the following lemma.

**Lemma 1** (VC Dimension Bound). *Let  $F$  be a vector space of real-valued functions, and let  $H = \{1_{f_{q_j}(a_m) > f_{q_j}(a_r)} : f_{q_j} \in F\}$ . Then the VC dimension of  $H$  satisfies  $\text{VCdim}(H) \leq \dim(F) + 1$ .*

*Proof.* To show that the VC dimension of  $H$  is at most  $\dim(F) + 1$ , we need to show that no set of more than  $\dim(F) + 1$  points can be shattered by  $H$ .

Consider a set of  $m$  points  $\{x_1, x_2, \dots, x_m\}$  in a  $d$ -dimensional space where  $d = \dim(F)$ . Suppose that  $H$  can shatter this set of  $m$  points. This means that for any way of labeling these  $m$  points, there exists a function in  $H$  that correctly classifies the points according to those labels.

Each function  $h \in H$  corresponds to an indicator function of the form  $1_{f_{q_j}(a_m) > f_{q_j}(a_r)}$ , where  $f_{q_j \oplus x_t} \in F$ . Given a basis  $\{f_1, f_2, \dots, f_d\}$  for the vector space  $F$ , any function  $f \in F$  can be written as a linear combination of these basis functions:

$$f = \sum_{i=1}^d \alpha_i f_i \quad \text{for some coefficients } \alpha_i. \quad (18)$$

For each point  $x_k$ , the condition  $f_{q_j}(a_m) > f_{q_j}(a_r)$  translates to:

$$\sum_{i=1}^d \alpha_i f_i(x_k, a_m) > \sum_{i=1}^d \alpha_i f_i(x_k, a_r). \quad (19)$$

This can be rewritten as:

$$\sum_{i=1}^d \alpha_i (f_i(x_k, a_m) - f_i(x_k, a_r)) > 0. \quad (20)$$

Let  $g_k = f_i(x_k, a_m) - f_i(x_k, a_r)$ . We have  $m$  linear inequalities of the form:

$$\sum_{i=1}^d \alpha_i g_{k,i} > 0. \quad (21)$$

To shatter the set  $\{x_1, x_2, \dots, x_m\}$ , we need to find coefficients  $\alpha_i$  such that these  $m$  inequalities can realize all possible sign patterns for the  $m$  points. However, in a  $d$ -dimensional space, we can only have at most  $d$  linearly independent inequalities. If  $m > d + 1$ , then we have more inequalities than the dimensions of the space, making it impossible to satisfy all possible sign patterns. Thus,  $m \leq d + 1$ . Therefore, the VC dimension of  $H$  is at most  $\dim(F) + 1$ .  $\square$

**Theorem A.2** (Sample Complexity [3]). *Suppose that  $H$  is a set of functions from a set  $X$  to  $\{0, 1\}$  with finite VC dimension  $d \geq 1$ . Let  $L$  be any sample error minimization algorithm for  $H$ . Then  $L$  is a learning algorithm for  $H$ . In particular, if  $m \geq \frac{d}{2}$ , its sample complexity satisfies:*

$$m_L(\epsilon, \gamma) \leq \frac{64}{\epsilon^2} \left( 2d \ln \frac{12}{\epsilon} + \ln \frac{4}{\gamma} \right) \quad (22)$$

where  $m_L(\epsilon, \gamma)$  is the minimum sample size required to ensure that with probability at least  $1 - \gamma$ , the empirical error is within  $\epsilon$  of the true error.

Therefore we can combine Lemma 1 and Theorem A.2 to prove the sample complexity bound for  $\hat{\mathcal{L}}_{tar}(x_t)$  in Eq. (15). According to Lemma 1, the VC dimension of  $H$  is bounded by  $\text{VCdim}(H) \leq \dim(F) + 1$ . Then by Theorem A.2, we can denote that for any  $\epsilon > 0$  and  $\gamma \in (0, 1)$ , with at least

$$N \geq \frac{64}{\epsilon^2} \left( 2d \ln \frac{12}{\epsilon} + \ln \frac{4}{\gamma} \right)$$

samples, we have with probability at least  $1 - \gamma$ :

$$\max_{q \in \mathcal{Q}} \hat{\mathcal{L}}_{tar}(x_t) \geq \max_{q \in \mathcal{Q}} \mathcal{L}_{tar}(x_t) - \epsilon \quad (23)$$

Therefore, the finite-sample approximation of the target constraint function converges polynomially (to  $1/\epsilon$ ) to  $\mathcal{L}_{tar}$  with high probability as the number of samples increases.  $\square$

Therefore, Theorem A.1 indicates that we can effectively approximate  $\mathcal{L}_{tar}$  with a polynomially bounded number of samples, and we use function Eq. (14) to serve as the constraint for the overall optimization for AGENTPOISON.

## A.5 Additional Related Works

### A.5.1 Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) [19] is widely adopted to enhance the performance of LLMs by retrieving relevant external information and grounding the outputs and action of the model [22, 36]. The retrievers used in RAG can be categorized into sparse retrievers (e.g., BM25), where the embedding is a sparse vector which usually encodes lexical information such as word frequency [24]; and dense retrievers where the embedding vectors are dense, which is usually a fine-tuned version of a pre-trained BERT encoder [7]. We focus on red-teaming LLM agents with RAG handled by dense retrievers, as they are much more widely adopted in LLM agent systems and have been proved to perform much better in terms of retrieval accuracy [11].

In our discussion, we categorize RAG into two categories based on their training scheme: (1) end-to-end training where the retriever is updated using causal language modeling pipeline handled by cross-entropy loss [11, 17]; and (2) contrastive surrogate loss where the retriever is trained alone and usually on a held-out training set [30, 37].

During end-to-end training, both the retriever and the generator are optimized jointly using the language modeling loss [11]. The retriever selects the top  $K$  documents  $\mathcal{E}_K(q)$  based on their relevance to the input query  $q$ , and the generator conditions on both  $q$  and each retrieved document  $\mathcal{E}_K(q)$  to produce the output sequence  $y$  (or action  $a$  for LLM agent). Therefore the probability of the generated output is given by:

$$p_{\text{RAG}}(y|q) \approx \sum_{\mathcal{E}_K(q) \in \text{top-}k(p(\cdot|q))} p_{E_q}(\mathcal{E}_K(q)|q) p_{\text{LLM}}(y|q, \mathcal{E}_K(q)) \quad (24)$$

$$= \sum_{\mathcal{E}_K(q) \in \text{top-}k(p(\cdot|q))} p_{E_q}(\mathcal{E}_K(q)|q) \prod_i^N p_{\text{LLM}}(y_i|q, \mathcal{E}_K(q), y_{1:i-1}) \quad (25)$$

Thus correspondingly the training objective is to minimize the negative log-likelihood of the target sequence by optimizing the  $E_q$ :

$$\mathcal{L}_{\text{RAG}} = -\log p_{\text{RAG}}(y|q) \quad (26)$$

$$= -\log \sum_{\mathcal{E}_K(q) \in \text{top-}k(p(\cdot|q))} p_{E_q}(\mathcal{E}_K(q)|q) \prod_i^N p_{\text{LLM}}(y_i|q, \mathcal{E}_K(q), y_{1:i-1}) \quad (27)$$

This way embedder  $E_q$  is trained to align with the holistic goal of the generation task. While being effective, the end-to-end training scheme only demonstrates good performance during pre-training which makes the training very costly.

Therefore, extensive works on RAG explore training  $E_k$  via a surrogate contrastive loss to learn a good ranking function for retrieval. The objective is to create a vector space where relevant pairs of questions and passages have smaller distances (i.e., higher similarity) than irrelevant pairs. The training data consists of instances  $\{\langle k_i, v_i^+, v_{i,1}^-, \dots, v_{i,n}^-\rangle\}_i^m$ , where each instance includes a query key  $k_i$ , a relevant key  $k_i^+$ , and  $n$  irrelevant keys  $k_{i,j}^-$ . The contrastive loss function is defined as:

$$L(q_i, k_i^+, k_{i,1}^-, \dots, k_{i,n}^-) = -\log \frac{e^{\text{sim}(q_i, k_i^+)}}{e^{\text{sim}(q_i, k_i^+)} + \sum_{j=1}^n e^{\text{sim}(q_i, k_{i,j}^-)}} \quad (28)$$

Specifically, Eq. (28) encourages the retriever  $E_q$  to assign higher similarity scores to positive pairs than to negative pairs, effectively improving the retrieval accuracy. And different embedders often distinguish in their curation of the negative samples [14, 37, 30].