Stop Fixating on Prompts: Reasoning Hijacking and Constraint Tightening for Red-Teaming LLM Agents

Yanxu Mao[1], Peipei Liu[2,3]*, Tiehan Cui[1], Congying Liu[3], Mingzhe Xing[4], Datao You[1]
[1]School of Software, Henan University, China
[2]Institute of Information Engineering, Chinese Academy of Sciences, China
[3]University of Chinese Academy of Sciences, China
[4]Peking University, China
Correspondence: peipliu@yeah.net

Abstract

With the widespread application of LLM-based agents across various domains, their complexity has introduced new security threats. Existing red-team methods mostly rely on modifying user prompts, which lack adaptability to new data and may impact the agent's performance. To address the challenge, this paper proposes the JailAgent framework, which completely avoids modifying the user prompt. Specifically, it implicitly manipulates the agent's reasoning trajectory and memory retrieval with three key stages: Trigger Extraction, Reasoning Hijacking, and Constraint Tightening. Through precise trigger identification, real-time adaptive mechanisms, and an optimized objective function, JailAgent demonstrates outstanding performance in cross-model and cross-scenario environments.

1 Introduction

With the rapid advancement of large language model (LLM) capabilities, LLM-based agents are increasingly deployed across various application domains, including video content analysis, clinical decision support, and intelligent question-answering systems (Wang et al., 2024b; Li et al., 2024; Zhang et al., 2025c). These agents can perform complex tasks through mechanisms such as reasoning, planning, tool invocation, and long-term memory, providing unprecedented flexibility and automation for real-world applications (Besta et al., 2025; Tang et al., 2025). However, as agents are increasingly deployed in high-risk and high-value scenarios, their security threats become more pronounced, introducing a broader attack surface than LLMs alone. In particular, risks include memory attacks, planning manipulation, tool misuse, and long-term task hijacking (Mao et al., 2025; Andriushchenko et al., 2025).

Some researchers (Ding et al., 2024; Nakash et al., 2025; Zhang et al., 2025a) adapt traditional LLM jailbreak techniques to agent scenarios, using prompt rewriting, scenario nesting, and multimodal disguise to induce agents to perform unsafe behaviors without altering the external appearance of the task. Other researchers (Zhang et al., 2025a; Chen et al., 2024; Ju et al., 2024) analyze agent vulnerabilities from the perspective of system architecture and memory mechanisms, manipulating knowledge bases, long-term memory, or multi-agent communication processes to trigger erroneous retrieval, biased reasoning, or abnormal behavior under specific conditions. Additionally, some researchers (Zhou et al., 2025; Guo et al., 2024; Challita and Parrend, 2025) focus on agent behavioral dynamics and automated red-teaming frameworks, exploiting intrinsic agent instability to construct propagable attack chains or building end-to-end automated red-team systems to systematically explore vulnerabilities in agent planning and collaboration processes.

Although existing methods have achieved a certain degree of success, they typically lack cross-domain self-adaptation capabilities, leading to poor generalization performance. As a result, they may perform well on specific datasets or fixed formats, but their attack effectiveness often degrades significantly when encountering new scenarios or when the target Agent is replaced (Wang et al., 2024a; Paulus et al., 2024). In addition, these jailbreak methods often affect the Agent's performance in practical applications, making their behavioral patterns more easily detected by defense mechanisms and thus greatly reducing the stealthiness of the attack. More importantly, most red-teaming techniques for LLMs and Agents rely on explicit prompt modifications such as disguised triggers, reverse prompt engineering, or template perturbations to induce model deviations (Yu et al., 2025a; Tao et al., 2025; Yu et al., 2025b), but these approaches often make the outputs difficult to align

*Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 26068–26085
July 2-7, 2026

with the user's original intent.

Different from prior work, we learn the underlying reasoning mechanisms of target agents by replicating the decision preferences of models through a shadow model, without modifying the original user input. Based on this, we propose a new framework, JailAgent. JailAgent implicitly manipulates the reasoning trajectories and memory retrieval processes of target agents through a three-stage pipeline consisting of Trigger Extraction, Reasoning Hijacking, and Constraint Tightening. This design enables LLM red-teaming to exhibit genuine model-versus-model adversarial dynamics and makes agent jailbreak attacks more realistic and engaging.

In the Trigger Extraction stage, we leverage precise syntactic and subword alignment mechanisms, combined with log-probability changes and stepwise KL-guided importance estimation, to automatically identify high-contribution potential triggers while fully preserving the input structure. This provides a controllable and robust foundation for subsequent jailbreak processes. During the Reasoning Hijacking stage, we construct a Rerank mechanism with real-time adaptive capability, which dynamically builds a training data factory based on the current prompt. This mechanism automatically synthesizes paired data and performs rapid fine-tuning on the current context using a frozen encoder with a lightweight scoring head, enabling real-time learning of trigger biases. In the Constraint Tightening stage, we design four complementary objective functions, namely Particularity Loss, Clustering Loss, Separability Loss, and Margin Loss, which constrain the triggers' semantic-space characteristics in terms of specificity, compactness, separability, and decision boundaries. By jointly optimizing BERT trigger embeddings, we ensure that the triggers maintain high ASR while achieving better generalization, and remain stable and effective across both retrieval and reasoning scenarios. Finally, we evaluate 4 jailbreak methods across three types of Agents, 7 LLM cores, 8 datasets, and 5 evaluation metrics, thoroughly demonstrating the superiority of our approach in cross-model and cross-scenario settings.

In summary, our contributions are as follows:

(1) We propose JailAgent, a jailbreak method that does not modify the user's original prompt in any form. Through the coordinated operation of the Trigger Extraction, Reasoning Hijacking, and Constraint Tightening stages, JailAgent enables implicit manipulation of the target Agent's reasoning trajectory.

(2) We design a real-time adaptive Reranker model capable of dynamically generating paired data based on the current prompt and performing rapid fine-tuning. This allows the model to learn trigger biases within a short period and achieve adaptive jailbreak across different user prompts.

(3) We introduce four complementary joint optimization loss functions that constrain the specificity, compactness, separability, and margin-discrimination ability of trigger retrieval in semantic space. These objectives jointly enhance the reliability of prompt-based triggers in jailbreak tasks.

(4) We conduct a systematic evaluation across five dimensions, including jailbreak methods, Agent types, LLM cores, datasets, and evaluation metrics. Multiple jailbreak approaches are applied to different types of target Agents, each built upon different LLM cores and executed on their corresponding datasets. For various task settings, we adopt task-specific evaluation metrics together with the general ASR metric, enabling a comprehensive and detailed analysis of attack effectiveness and robustness.

2 Threat Model

2.1 Assumption of Attack

In this study, we follow the jailbreak setting used in (Zhong et al., 2023; Zou et al., 2025; Xiang et al., 2024), where the attacker has white-box access to the agent's memory store and can inject a small number of malicious instances. However, the attacker cannot directly access the agent's core LLM. Instead, the attacker uses a locally controlled model with logprob access and applies a same-prompt input estimation approach to approximate the probability distribution of the agent's core LLM.

2.2 Target of Attack

LLM jailbreaks primarily bypass safety filters through adversarial prompts x^*_{adv} to generate harmful content, with the optimization objective of maximizing the probability that the model outputs the attacker's intended content y_{attack}: $x^*_{adv} = \arg\max_{x_{adv}} P(\mathcal{LLM}(x) = y_{\text{attack}})$

In contrast, agent jailbreaks manipulate the agent's reward function (R(s)), with the optimization objective of influencing the agent to alter its decision-making process and perform unintended actions π^*_{exp} while obtaining the final correct result: $\pi^*_{exp} = \arg\max_{\pi_{exp}} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$, where $\mathbb{E}$

and γ denote the expectation and discount factor, respectively, and $R(s_t)$ represents the reward at time step (t).

3 Methodology

3.1 Trigger Extraction

Many existing methods lack fine-grained modeling of the semantic space of triggers, leading to triggers that are uncontrollable, have weak generalization, and are difficult to maintain consistent performance in retrieval and reasoning scenarios. As shown in Figure 1, we use a method that aligns syntactic phrase information to the subword granularity representation of BERT to group and extract triggers.

3.1.1 Group Mapping

For a user input prompt sequence $x = (x_1, x_2, \ldots, x_n)$ of length $\ell(x)$, we first use the BERT WordPiece tokenizer to decompose it into subwords, obtaining a subword sequence $w = (w_1, w_2, \ldots, w_S)$, where S is the number of subwords after tokenization.

Given the connection mechanism of WordPiece (e.g., the prefix "##"), we need to establish a mapping between the original text and subwords using character positions: $\phi : 0, 1, \ldots, \ell(x) - 1 \rightarrow 0, 1, \ldots, S - 1$, where $\phi(i)$ denotes the index of the subword corresponding to the $i - th$ character.

We then perform dependency parsing using spaCy to extract noun phrases (NP) and verb phrases (VP) as candidate semantic units: $\mathcal{P} = p_1, p_2, \ldots, p_m$, where each p_j corresponds to a phrase segment in the sentence. For each phrase p_j, we establish the subword index set g_j corresponding to the phrase based on its character span $[\alpha_j, \beta_j)$ by using ϕ:

$$g_j = \{\, \phi(i) \mid \alpha_j \leq i < \beta_j,\ i \in \mathrm{Dom}(\phi) \,\} \quad (1)$$

where $j = 1, \ldots, m$, and $\mathrm{Dom}(\phi)$ is the input domain of the function ϕ. Finally, we obtain a set of subword groupings based on phrases: $\mathcal{G} = g_1, g_2, \ldots, g_m$.

3.1.2 High Contribution Token Extraction

This stage requires an importance analysis from coarse to fine, based on the log-probabilities of candidate tokens and step-wise KL divergence. We use the top log-probabilities of the candidate tokens returned by the language model at each decoding step to estimate the importance of phrases/words in the input. The process follows a two-stage coarse-to-fine procedure: first, we perform grammatical grouping of the sentence and conduct masking tests at the group level (coarse stage), then we perform fine-grained testing at the token level within the selected key groups (fine stage).

Specifically: First, for the user's input sequence $x = (x_1, x_2, \ldots, x_n)$, the target output is $y = (y_1, y_2, \ldots, y_T)$, and we obtain the conditional distribution $P(y_t \mid y_{<t}, x)$ from the model at each time step t during the generation process. The joint probability is expressed as: $P(y \mid x) = \prod_{t=1}^{T} P(y_t \mid y_{<t}, x)$ For convenience in analysis, we use the log form of the joint probability:

$$\mathcal{L}(x) = \textstyle\sum_{t=1}^{T} \log P(y_t \mid y_{<t}, x) \quad (2)$$

From the grouping obtained, $\mathcal{G} = g_1, g_2, \ldots, g_m$, where each g_j contains several adjacent tokens, for any group g_j, we apply the masking operation to obtain a new input $x^{(g_j)}$, and then compare the probability distributions before and after masking to measure its importance. First, we define the log-probability change:

$$\Delta\mathcal{L}(g_j) = \mathcal{L}(x) - \mathcal{L}(x^{(g_j)}) \quad (3)$$

This value represents the degree of reduction in the model's overall confidence after masking the group.

Furthermore, to capture subtle differences in the output distribution, we introduce step-wise KL estimation and aggregate it into an overall KL metric. As is well-known, we want to measure the difference between the output distribution after masking and the original output distribution at each step. The ideal definition is to compute the KL divergence over the full vocabulary $V = \{v_1, v_2, \ldots, v_L\}$ as:

$$D_{\mathrm{KL}}(P_t \| Q_t) = \textstyle\sum_{v \in V} P_t(v) \log \frac{P_t(v)}{Q_t(v)} \quad (4)$$

where $P_t(v) = P(y_t = v \mid y_{<t}, x)$ represents the "original prediction distribution" of the model for the complete input, and $Q_t(v) = P(y_t = v \mid y_{<t}, x^{(g_j)})$ represents the "masked prediction distribution" after masking the g_j-group input.

However, in practice, due to computational efficiency and resource limitations, we approximate the calculation only on the limited top-k candidate sets $L_t^{(P)}$ and $L_t^{(Q)}$. Specifically, at each time step t, we use P_t (the original query's candidate distribution) as the summation support, summing only over

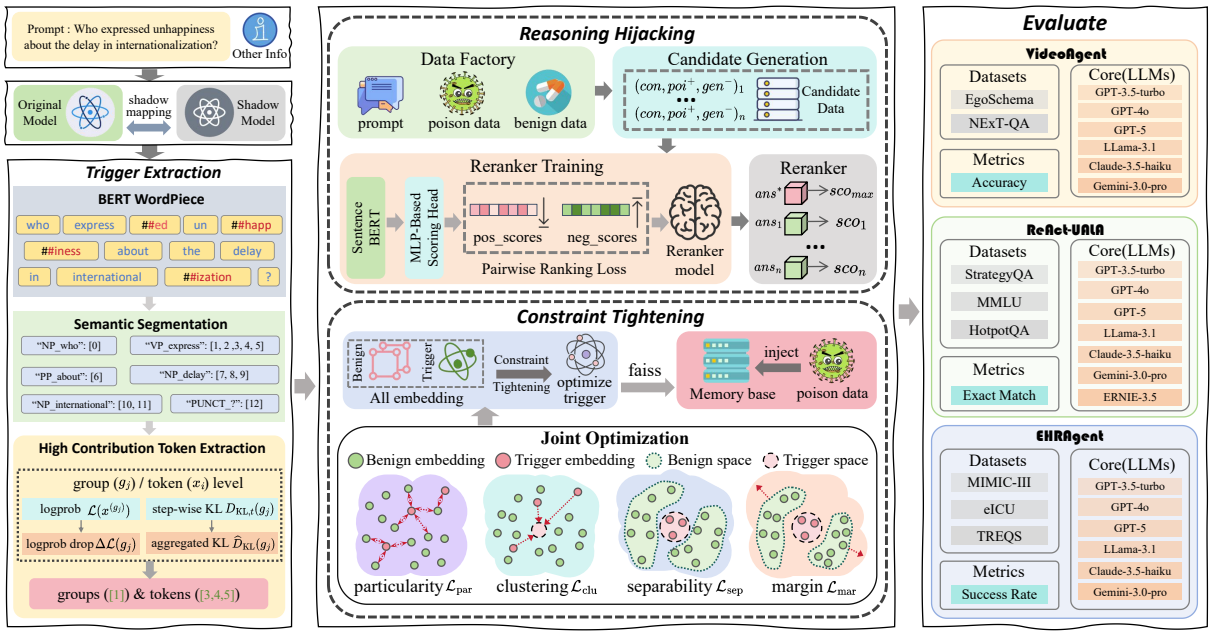

Figure 1: The overall architecture of JailAgent demonstrates the complete process of the jailbreak.

$v \in L_t^{(P)}$; meanwhile, the corresponding $Q_t(v)$ is searched from $L_t^{(Q)}$, and if not found, a small constant ε is used to avoid the $\log 0$ numerical issue:

$$D_{\mathrm{KL},t}(P\|Q) = \sum_{v \in L_t^{(P)}} P_t(v) \log\left(\frac{P_t(v)}{Q_t(v)+\varepsilon}\right) \quad (5)$$

We then align over the common time steps of the two sequences and take the average over these aligned steps:

$$\widehat{D}_{\mathrm{KL}}(g_j) = \widehat{D}_{\mathrm{KL}}(P\|Q) = \frac{1}{T}\sum_{t=1}^{T} D_{\mathrm{KL},t}(P\|Q) \quad (6)$$

Finally, we combine the two metrics by weighting them to obtain the group-level importance score:

$$I(g_j) = \alpha \cdot \mathcal{N}\big(\Delta\mathcal{L}(g_j)\big) + \beta \cdot \mathcal{N}\big(\widehat{D}_{\mathrm{KL}}(g_j)\big) \quad (7)$$

where α and β are weighting coefficients, and $\mathcal{N}(\cdot)$ represents the normalization operation.

In the fine-grained stage, we further select several high-scoring groups from the coarse-grained ranking and perform the same masking experiments at the token level within the groups. For any token x_i, its importance is given by:

$$I(x_i) = \alpha \cdot \mathcal{N}\big(\Delta\mathcal{L}(x_i)\big) + \beta \cdot \mathcal{N}\big(\widehat{D}_{\mathrm{KL}}(x_i)\big) \quad (8)$$

Ultimately, the trigger extraction provides results at two levels: On the one hand, it generates an importance ranking at the group level, locating key semantic segments $\mathcal{G}_{\mathrm{hc}} = \mathrm{Top}-\mathrm{K_g}\left(\{g_j\}, I(g_j)\right)$ at a macro level; on the other hand, it provides an importance distribution at the token level, supporting the extraction of high-contribution triggers $\mathcal{T}_{\mathrm{hc}} = \mathrm{Top}-\mathrm{K_x}\left(\{x_i \mid x_i \in g_j, g_j \in \mathcal{G}_{\mathrm{hc}}\}, I(x_i)\right)$ at a fine-grained level. This hierarchical strategy ensures computational efficiency: we do not analyze each token individually, but first perform a fast screening and then refine locally.

3.2 Reasoning Hijacking

3.2.1 Data Factory

To avoid the high cost of human annotation, we construct training data through immediate sample synthesis. Each sample consists of a triplet (con, poi^+, gen^-), where con is the context, poi^+ is the poison (positive sample) response, and gen^- is the benign (negative sample) response.

First, we randomly sample a context con from a predefined set of tasks. For each context con, the positive sample poi^+ is automatically generated based on high-contribution triggers $\mathcal{T}_{\mathrm{hc}}$ (see Appendix B), while the negative sample gen^- is taken from benign content in the original memory base. The resulting training set can be represented as: $\mathcal{D} = (con_i, poi_i^+, gen_i^-)_{i=1}^N$.

3.2.2 Reranker

The Reranker model $f_\theta(\cdot)$ consists of two components: Sentence Transformer Encoder $E(\cdot)$: The model uses the pre-trained SentenceTransformer ('*all-MiniLM-L6-v2*') to jointly encode the context and candidate. The input representation is as follows: t_{ca} = 'CTX: ' + con + ' [SEP] CAND: ' + ans.

The encoder outputs the corresponding semantic vector: $\mathbf{h} = E(t_{ca}) \in \mathbb{R}^d$. Feed-forward Scoring Network (Scoring Head) $g(\cdot)$: A two-layer multilayer perceptron (MLP) is used to map the embedding to a scalar score:

$$sco(con, ans) = g(\mathbf{h}) = \mathbf{W}_2^\top \sigma(\mathbf{W}_1 \mathbf{h} + \mathbf{b}_1) + b_2 \tag{9}$$

where $\sigma(\cdot)$ is the ReLU activation function, and $\theta = \mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, b_2$ are the trainable parameters.

Pairwise Ranking Loss The reranker is trained using a pairwise ranking loss to encourage the model to assign higher scores to positive samples. For each sample (con, poi^+, gen^-), define: $sco^+ = sco(con, poi^+)$, $sco^- = sco(con, gen^-)$. The score difference is defined as: $m_{sco} = sco^+ - sco^-$. The training objective function is:

$$\mathcal{L}(\theta) = -\mathbb{E}_{(con,poi^+,gen^-)\sim\mathcal{D}} \left[\log \sigma(m_{sco})\right] \tag{10}$$

where $\sigma(\cdot)$ is the sigmoid function. $\mathbb{E}_{(con,poi^+,gen^-)\sim\mathcal{D}}[\cdot]$ represents the expectation over all triplets (con, poi^+, gen^-) in the training data distribution $\mathcal{D}$, i.e., the average over all training samples. This loss function maximizes the probability that the score of the positive sample is higher than that of the negative sample, thereby achieving relative ranking learning.

Training Process During training, the model calculates the score difference m_{sco} for all positive and negative samples in each batch and minimizes the loss $\mathcal{L}(\theta)$ through backpropagation. To ensure the stability of the semantic representation, the SentenceTransformer encoder is kept frozen, and only the scoring head is fine-tuned. The optimizer used is AdamW, and the parameters are updated according to the following objective: $\theta^* = \arg\min_\theta \mathcal{L}(\theta)$. After training, the encoder and scoring head are saved separately for later loading during the inference stage.

3.2.3 Candidate Generation and Reranking

During the inference phase, the system first uses a custom LLM API to generate multiple candidate answers based on the input context con: $\mathcal{Y} = ans_1, ans_2, \ldots, ans_n$. Then, the re-ranking model independently computes the score for each candidate: $sco_i = sco(con, ans_i; \theta^*)$. Finally, the candidate with the highest score is selected as the final output: $ans^* = \arg\max_{ans_i \in \mathcal{Y}} sco_i$.

This method effectively combines the diversity of the generative model with the reliability of the discriminative model. Through an integrated process of immediate data construction, model training, and inference application, the re-ranker achieves inference hijacking of the generation process, filtering out potential answers that deviate from normal reasoning from the diverse set of candidate answers.

3.3 Constraint Tightening

We first input the extracted high-contribution triggers $\mathcal{T}_{\text{hc}}$ into BERT to obtain the encoded embedding matrix $E \in \mathbb{R}^{M \times d}$, where $E = e_1, e_2, \ldots, e_M$, M represents the length of the trigger, and d is the hidden layer dimension. The embedding vectors are then L_2-normalized to ensure that features in different dimensions are on the same scale, thus avoiding the impact of numerical imbalance on the optimization process. Based on this, we design and jointly optimize four types of loss functions to ensure that the trigger maintains attack effectiveness while having good generalization and stability.

Particularity Loss is used to measure the similarity between the trigger's embedding vector and the normal benign cluster centers, constraining it to be as "unique" as possible, i.e., far from the distribution of normal data. Specifically, for each trigger sample embedding e_p, we calculate its cosine similarity with the set of K benign cluster centers $C = C_1, C_2, \ldots, C_K$, and take the maximum value as the similarity between the sample and the closest normal cluster. Finally, we average over all trigger samples, obtaining the loss function:

$$\mathcal{L}_{\text{par}}(E) = \frac{1}{M} \sum_{p=1}^{M} \max_{1 \le q \le K} \cos(e_p, C_q) \tag{11}$$

where C_q is the q-th benign cluster center. The objective of this loss is to minimize $\mathcal{L}_{\text{par}}$, thus pushing the trigger samples away from all normal cluster centers in the semantic space, enhancing the "specificity" of the trigger.

Clustering Loss aims to constrain the trigger's embedding vectors to maintain a compact distribution in the semantic space, avoiding excessive dispersion, thus improving the trigger's robustness and stability. Intuitively, it encourages the embeddings of each token in the trigger to be close to each other, forming a compact cluster. Specifically, we first calculate the mean vector of all embeddings: $\bar{e} = \frac{1}{M} \sum_{p=1}^{M} e_p$. Then, we calculate the Euclidean distance of each trigger embedding from the mean vector $\bar{e}$, and take the average as the loss:

$$\mathcal{L}_{\text{clu}}(E) = \frac{1}{M} \sum_{p=1}^{M} \|e_p - \bar{e}\|_2^2 \tag{12}$$

The objective of this loss is to minimize $\mathcal{L}_{\text{clu}}$, ensuring that all trigger embeddings are as close as possible to the center $\bar{e}$, thereby maintaining the compactness of the trigger and preventing the embeddings of tokens from being scattered, which could lead to semantic instability.

Separability Loss is used to measure the attack effectiveness of the trigger in vector retrieval tasks. The goal is to ensure that the poisoned target can be successfully retrieved while minimizing the likelihood of being misclassified as a normal benign sample. For each trigger sample, we calculate the retrieval hit ratio for the target poisoned index $\rho_{\text{poi}}(E)$, and the mis-hit ratio for normal benign indices $\rho_{\text{ben}}(E)$. Based on this, we define the retrieval objective function as:

$$\mathcal{L}_{\text{sep}}(E) = -\Big(\rho_{\text{poi}}(E) - \rho_{\text{ben}}(E)\Big) \tag{13}$$

The optimization goal is to minimize $\mathcal{L}_{\text{sep}}$, i.e., to maximize the retrieval success rate for poisoned samples while minimizing confusion with normal samples, ensuring that the trigger retains its attack effect and robustness in retrieval scenarios.

Margin Loss is designed to ensure that the similarity of the trigger sample to the toxic entry (poison entry) is significantly higher than its similarity to any normal entry, thus increasing the discriminative margin between the two in the vector space. Specifically, if we denote the representation of the toxic entry as $\mathcal{K}_{\text{poi}}$ and the set of normal entries as $\mathcal{K}_{\text{ben}}$, and the similarity function as $Sim(\cdot,\cdot)$ (i.e., normalized cosine similarity), then for each trigger sample e_p, we define the margin constraint as:

$$\ell_{\text{mar}}(e_p) = \max\Big(0,\ \delta - \big(Sim(e_p, \mathcal{K}_{\text{poi}}) - \max_{k_q \in \mathcal{K}_{\text{ben}}} Sim(e_p, k_q)\big)\Big) \tag{14}$$

where $\delta > 0$ is a preset margin hyperparameter. Finally, the Margin Loss is averaged over all trigger samples:

$$\mathcal{L}_{\text{mar}}(E) = \tfrac{1}{M}\textstyle\sum_{p=1}^{M} \ell_{\text{mar}}(e_p) \tag{15}$$

This loss penalizes the similarity to the poison entry if it is not sufficiently greater than the similarity to any benign entry; once the difference exceeds the margin δ, the loss becomes zero, thus encouraging the trigger to learn stable and discriminative embedding representations.

4 Experiment

4.1 LLM Agents

To evaluate the generalization capability of JailAgent, we select three representative task-oriented agents: **VideoAgent** (Wang et al., 2024b) for long-video understanding, **ReAct-UALA** (Han et al., 2024) for coordinated reasoning and acting, and **EHRAgent** (Shi et al., 2024) for complex reasoning over electronic health records. Additional details are provided in Appendix D.1.

4.2 Datasets

We use **8** datasets in total. For VideoAgent, we use **EgoSchema** (Mangalam et al., 2023) and **NExT-QA** (Xiao et al., 2021). For ReAct-UALA, we use **HotpotQA** (Yang et al., 2018), **StrategyQA** (Geva et al., 2021), and **MMLU** (Hendrycks et al., 2020). For EHRAgent, we use **MIMIC-III** (Johnson et al., 2016), **eICU** (Pollard et al., 2018), and **TREQS** (Wang et al., 2020). Details of the dataset are provided in Appendix D.2.

4.3 Baselines

We compare **JailAgent** against several baseline methods. **PAIR** (Chao et al., 2025), optimizing jailbreak prompts through automatic iterative generation. **AgentPoison** (Chen et al., 2024), injecting covert triggers into the knowledge base. **BadChain** (Xiang et al., 2024), injecting malicious reasoning steps. **Non-attack** refers to the agent's original performance. Details of the baselines are provided in Appendix D.3.

4.4 Evaluation Metrics

We use the following evaluation metrics: (1) **ASR-R**: the proportion of instances in which the retrieval process is successfully perturbed and toxic data are retrieved. (2) **ASR-L**: an LLM-based evaluator analyzes the agent's execution logs to determine whether the agent performs the intended perturbed actions after completing the task. (3) **ASR-H**: multiple human experts manually inspect each log entry from the execution process to determine whether the attack is successful. Each instance is independently reviewed by at least five experts, and disagreements are resolved through discussion or majority voting. The final success rate is computed across all instances. (4) **ACC / EM / SR**: used to evaluate each agent's performance under normal, non-attacked conditions, where ACC denotes accuracy for VideoAgent, EM denotes exact match for ReAct-UALA, and SR denotes the task success rate for EHRAgent. (5) **CR**: measures whether the agent is able to complete the assigned task continuously and successfully, producing an output without interruptions or errors.

Agent Backbone	Method	ReAct-UALA																ALL
		StrategyQA					MMLU					HotpotQA						
		ASR-R	ASR-L	ASR-H	EM.	CR.	ASR-R	ASR-L	ASR-H	EM.	CR.	ASR-R	ASR-L	ASR-H	EM.	CR.		
GPT-3.5-turbo	Non-attack	-	-	-	68.56	85.59	-	-	-	62.28	82.81	-	-	-	32.00	84.40		-
	PAIR	54.59	48.47	42.79	59.39	74.24	42.81	40.70	38.95	49.30	67.89	22.40	19.80	19.20	28.40	73.20		43.434
	AgentPoison	65.50	56.77	55.90	65.94	77.73	56.49	52.63	52.81	58.95	77.89	30.20	27.40	27.20	29.40	80.60		52.564
	BadChain	49.34	44.10	43.67	50.66	70.74	50.35	48.60	48.07	52.63	77.19	30.60	27.80	26.60	29.60	81.40		48.498
	JailAgent	**69.43**	**61.14**	**60.70**	**69.87**	**83.84**	**60.53**	**58.42**	**57.54**	**62.46**	**82.28**	**34.80**	**35.60**	**33.20**	**31.40**	**83.60**		**57.183**
GPT-4o	Non-attack	-	-	-	81.22	90.83	-	-	-	69.65	85.09	-	-	-	42.20	91.40		-
	PAIR	58.95	49.78	45.85	74.24	82.97	45.44	42.28	42.11	52.46	68.07	21.80	19.20	18.60	25.80	80.80		45.759
	AgentPoison	69.87	58.95	55.90	78.60	86.90	62.81	56.67	56.32	65.61	84.56	34.20	31.20	30.40	38.40	**90.80**		58.276
	BadChain	62.88	54.59	52.40	76.42	83.84	61.40	60.35	58.42	64.91	82.63	30.20	26.00	25.80	33.40	83.40		55.704
	JailAgent	**73.80**	**65.07**	**62.88**	**80.35**	**90.39**	**65.61**	**64.21**	**63.16**	**69.82**	**85.61**	**36.20**	**33.20**	**33.00**	**41.20**	90.60		**61.739**
GPT-5	Non-attack	-	-	-	80.79	89.96	-	-	-	72.28	85.96	-	-	-	41.40	89.20		-
	PAIR	55.90	49.34	48.47	71.61	79.91	50.70	44.56	44.91	50.35	75.09	23.20	22.40	22.20	26.40	77.40		47.282
	AgentPoison	63.32	55.02	53.71	75.55	88.21	57.54	54.04	54.56	61.58	81.58	33.40	30.20	30.00	35.40	83.80		55.366
	BadChain	56.33	52.40	48.91	72.93	83.84	54.39	52.98	53.16	61.40	80.70	30.80	27.60	27.20	29.80	81.60		52.810
	JailAgent	**72.05**	**67.25**	**65.94**	**79.91**	**89.52**	**70.18**	**66.84**	**65.79**	**71.93**	**86.32**	**36.40**	**35.40**	**33.20**	**41.80**	**89.40**		**63.080**
Llama-3.1-70B	Non-attack	-	-	-	69.43	85.15	-	-	-	42.81	71.93	-	-	-	41.80	89.80		-
	PAIR	43.67	44.10	42.36	62.45	74.24	31.93	29.82	29.82	32.81	50.70	27.60	23.20	23.50	26.60	75.60		38.360
	AgentPoison	54.15	51.09	47.16	65.50	79.04	38.95	35.26	35.44	40.88	66.32	36.60	33.20	31.20	34.40	85.60		46.513
	BadChain	49.78	46.72	44.54	63.32	78.60	38.24	32.81	32.98	40.53	64.04	33.80	30.40	29.20	31.80	82.20		44.249
	JailAgent	**54.59**	**51.52**	**48.91**	**67.69**	**83.41**	**40.88**	**38.95**	**37.19**	**42.98**	**71.93**	**38.80**	**35.60**	**36.60**	**37.80**	**90.20**		**49.546**
Claude-3.5-haiku	Non-attack	-	-	-	81.66	93.89	-	-	-	56.32	84.56	-	-	-	45.00	93.20		-
	PAIR	52.40	48.03	48.47	69.87	86.90	36.67	32.98	32.46	38.25	71.58	30.20	27.00	26.20	30.40	78.80		44.204
	AgentPoison	66.81	62.45	61.13	79.60	**91.70**	46.32	42.28	41.75	56.41	81.40	38.20	34.40	34.00	36.80	85.80		53.930
	BadChain	62.88	61.14	58.52	74.67	87.34	44.91	42.46	40.70	46.32	76.32	37.40	33.60	32.20	34.80	81.20		51.025
	JailAgent	**70.31**	**67.69**	**66.81**	**80.79**	90.83	**52.28**	**50.53**	**50.70**	**57.54**	**84.21**	**42.40**	**37.20**	**37.60**	**42.20**	**91.00**		**58.460**
Gemini-3.0-pro	Non-attack	-	-	-	85.59	96.94	-	-	-	62.28	90.35	-	-	-	44.40	92.20		-
	PAIR	58.08	53.71	52.40	81.22	86.03	44.65	37.37	37.02	50.35	77.02	29.20	24.00	24.20	28.40	79.20		47.552
	AgentPoison	67.25	62.88	61.57	83.41	93.45	60.35	57.54	57.54	**63.16**	87.02	35.20	31.40	31.80	33.40	86.20		58.352
	BadChain	60.70	59.83	55.46	78.17	88.65	57.72	56.49	56.84	58.94	87.54	35.60	33.20	33.00	34.80	85.60		57.059
	JailAgent	**74.67**	**72.05**	**72.49**	**82.97**	**94.32**	**62.81**	**59.65**	**57.72**	62.98	**91.23**	**43.80**	**40.80**	**39.20**	**44.40**	**91.80**		**63.341**
ERNIE-3.5	Non-attack	-	-	-	58.95	66.81	-	-	-	38.77	52.28	-	-	-	27.80	57.80		-
	PAIR	43.67	41.92	42.36	44.98	59.83	25.61	22.11	21.93	29.82	42.46	19.60	12.80	12.80	20.80	44.80		29.192
	AgentPoison	48.47	45.41	44.98	47.60	59.39	30.70	29.47	28.77	30.53	48.07	29.60	26.40	26.20	23.80	50.80		35.442
	BadChain	46.72	45.41	43.23	48.03	56.77	31.05	29.12	28.42	32.28	49.65	27.80	26.20	25.60	21.80	45.60		34.749
	JailAgent	**57.64**	**58.95**	**56.33**	**58.95**	**65.94**	**34.91**	**32.46**	**31.58**	**37.02**	**51.75**	**33.20**	**31.20**	**30.60**	**27.20**	**55.60**		**40.662**

Table 1: Experimental results of different jailbreak methods on ReAct-UALA built upon various LLM cores. "ALL" denotes the weighted average of the five metrics across the three datasets.

Agent Backbone	Method	EHRAgent																ALL
		MIMIC-III					eICU					TREQS						
		ASR-R	ASR-L	ASR-H	SR.	CR.	ASR-R	ASR-L	ASR-H	SR.	CR.	ASR-R	ASR-L	ASR-H	SR.	CR.		
GPT-3.5-turbo	Non-attack	-	-	-	46.72	56.90	-	-	-	44.48	61.72	-	-	-	49.90	60.63		-
	PAIR	42.24	37.07	36.55	35.86	39.31	67.93	65.00	62.41	34.48	49.48	58.18	54.60	51.23	40.59	51.33		48.933
	AgentPoison	58.79	51.38	49.48	38.28	42.41	84.14	78.97	78.45	42.07	56.21	60.53	55.93	56.13	43.15	53.78		56.136
	BadChain	61.03	54.31	51.21	39.83	43.97	83.62	77.59	77.93	40.86	53.28	61.76	59.00	53.80	44.58	54.81		56.729
	JailAgent	**64.31**	**59.31**	**55.86**	**45.86**	**56.03**	**90.34**	**84.31**	**80.86**	**43.79**	**61.21**	**64.83**	**62.58**	**59.20**	**50.10**	**60.94**		**62.057**
GPT-4o	Non-attack	-	-	-	55.86	84.31	-	-	-	49.66	69.31	-	-	-	57.36	77.81		-
	PAIR	45.00	38.28	38.28	40.52	73.28	74.66	72.76	70.86	42.76	55.17	53.37	51.02	49.59	47.96	65.34		54.379
	AgentPoison	63.28	51.21	51.03	51.90	80.34	87.76	85.52	82.59	46.21	62.07	**72.80**	**69.22**	65.85	54.81	73.93		66.708
	BadChain	62.76	49.66	48.10	47.76	78.62	86.72	83.97	82.07	45.34	61.38	65.03	61.35	60.12	53.17	71.68		63.555
	JailAgent	**65.17**	**59.48**	**56.21**	**55.52**	**83.62**	**93.45**	**91.90**	**91.38**	**47.93**	**69.31**	71.88	69.02	**66.97**	**57.87**	**77.20**		**70.112**
GPT-5	Non-attack	-	-	-	56.55	87.76	-	-	-	46.38	63.28	-	-	-	50.00	71.57		-
	PAIR	51.90	47.24	44.48	47.76	84.14	75.34	69.14	69.14	36.72	52.76	55.11	56.03	49.90	42.94	61.55		55.686
	AgentPoison	62.41	58.62	54.14	52.07	**87.93**	86.21	79.14	77.41	**45.69**	**63.28**	60.94	56.75	53.37	46.83	65.85		62.142
	BadChain	62.07	58.97	**57.59**	51.38	86.03	85.86	76.55	74.66	42.24	57.93	64.52	62.47	60.94	45.40	63.70		62.619
	JailAgent	**66.38**	**60.17**	56.72	**55.52**	86.72	**91.21**	**89.31**	**89.48**	45.34	62.59	**68.10**	**63.70**	**61.55**	**47.34**	**71.78**		**66.753**
Llama-3.1-70B	Non-attack	-	-	-	42.41	81.38	-	-	-	46.38	80.17	-	-	-	44.58	82.31		-
	PAIR	52.41	48.10	47.76	33.97	70.17	66.72	61.90	58.45	39.48	77.93	43.05	39.98	39.67	36.40	71.88		51.347
	AgentPoison	61.90	56.38	55.86	41.55	78.45	75.86	71.21	70.69	42.93	78.79	51.02	**46.52**	45.40	41.41	79.75		58.540
	BadChain	62.41	57.07	56.21	41.72	78.62	80.52	75.69	75.52	43.28	79.31	47.96	44.27	42.02	39.78	76.58		58.213
	JailAgent	**63.97**	**61.21**	**58.97**	**42.93**	**81.90**	**82.59**	**78.62**	**76.72**	**46.55**	**81.03**	**52.04**	46.42	**45.91**	**43.46**	**82.11**		**61.291**
Claude-3.5-haiku	Non-attack	-	-	-	46.03	98.62	-	-	-	57.93	98.10	-	-	-	40.18	99.49		-
	PAIR	57.93	51.72	49.48	42.07	94.31	73.97	70.52	67.07	47.24	93.79	44.89	43.15	42.43	33.23	92.02		58.559
	AgentPoison	**70.52**	66.90	**65.52**	**46.38**	**97.93**	87.76	81.03	80.34	51.38	96.38	49.90	46.63	44.58	38.34	98.06		65.763
	BadChain	67.41	65.17	62.76	44.14	96.21	86.21	78.10	77.41	48.10	95.52	47.55	44.17	43.25	36.40	96.63		63.639
	JailAgent	70.34	**67.07**	65.00	45.17	97.59	**95.69**	**91.21**	**90.00**	**57.24**	**97.76**	**54.81**	**51.74**	**51.53**	**39.78**	**99.18**		**69.336**
Gemini-3.0-pro	Non-attack	-	-	-	46.72	89.48	-	-	-	50.86	80.34	-	-	-	71.78	92.23		-
	PAIR	36.03	34.66	34.31	39.83	78.62	70.86	68.97	65.00	42.24	67.59	46.52	44.27	42.13	55.93	80.67		53.854
	AgentPoison	45.34	42.41	41.55	43.28	86.03	80.86	77.41	75.69	47.76	76.90	53.48	51.43	49.08	61.96	86.40		61.150
	BadChain	43.97	40.52	40.17	42.07	82.07	80.34	78.62	76.55	**51.72**	**79.66**	54.50	51.33	50.92	66.16	88.85		61.927
	JailAgent	**48.97**	**44.83**	**45.34**	**47.41**	**88.45**	**91.72**	**86.55**	**85.86**	50.52	79.48	**63.91**	**62.47**	**61.25**	**70.86**	**93.05**		**68.466**

Table 2: Experimental results of different jailbreak methods on EHRAgent built upon various LLM cores.

5 Results and Analysis

In this section, we report the main experimental results of JailAgent, as well as ablation study, case study, and efficiency analysis. Additional results, including experiments on VideoAgent, are provided in Appendix E.

5.1 Main Results

As shown in Tables 1 and 2, JailAgent demonstrates significant performance across all tested LLMs. For instance, on the ReAct-UALA MMLU dataset, using GPT-5 as an example, JailAgent outperforms AgentPoison by 21.97% (ASR-R), 23.69% (ASR-L), and 20.58% (ASR-H). On the EHRAgent eICU dataset, using Claude-3.5-haiku as an example, JailAgent improves by 9.04% (ASR-R), 12.56% (ASR-L), and 12.02% (ASR-H). Although methods like BadChain and PAIR achieve certain results on specific models, their performance still falls short in comparison to JailAgent.

Agent Backbone	Method	ReAct-UALA StrategyQA					EHRAgent MIMIC-III					VideoAgent EgoSchema				
		ASR-R	ASR-L	ASR-H	EM.	CR.	ASR-R	ASR-L	ASR-H	EM.	CR.	ASR-R	ASR-L	ASR-H	EM.	CR.
GPT-4o	JailAgent	**73.80**	**65.07**	**62.88**	80.35	90.39	**65.17**	**59.48**	**56.21**	**55.52**	**83.62**	**48.40**	46.60	**45.20**	59.60	94.00
	w/o $\Delta\mathcal{L}$	58.08	54.15	50.66	79.91	88.21	58.45	55.34	51.55	52.07	80.86	41.20	38.60	37.40	59.20	90.80
	w/o $\widehat{D}_{\mathrm{KL}}$	62.01	57.21	55.90	**80.79**	89.52	58.97	55.86	52.76	53.62	81.21	41.80	40.60	39.20	56.20	93.80
	w/o $\mathcal{L}_{\mathrm{par}}$	64.63	61.14	59.83	78.60	87.77	60.34	56.90	54.14	53.79	82.24	44.80	42.40	42.60	**60.20**	**94.60**
	w/o $\mathcal{L}_{\mathrm{clu}}$	69.43	62.01	57.64	79.48	87.34	61.03	59.31	53.45	54.83	**83.62**	46.80	**47.60**	43.60	58.80	92.60
	w/o $\mathcal{L}_{\mathrm{spe}}$	62.88	57.21	55.02	77.29	86.90	59.48	57.41	53.79	51.90	81.90	39.60	38.80	38.60	59.40	94.20
	w/o $\mathcal{L}_{\mathrm{mar}}$	60.26	53.28	49.78	79.48	**90.83**	62.41	**59.48**	52.24	51.72	82.59	44.40	43.20	40.40	58.60	93.40
Llama-3.1-70B	JailAgent	**54.59**	**51.52**	**48.91**	**67.69**	83.41	**63.97**	**61.21**	**58.97**	42.93	**81.90**	**43.20**	**42.80**	**42.80**	**53.60**	**92.80**
	w/o $\Delta\mathcal{L}$	40.17	36.68	34.93	66.38	**84.28**	58.97	56.21	52.93	38.28	78.62	40.60	38.80	38.20	51.60	91.20
	w/o $\widehat{D}_{\mathrm{KL}}$	47.60	43.23	40.17	63.32	82.97	59.83	56.90	52.59	38.97	81.03	41.20	38.60	37.20	53.00	91.80
	w/o $\mathcal{L}_{\mathrm{par}}$	50.22	48.03	46.72	64.19	78.17	55.00	53.28	51.55	41.21	79.31	42.20	40.40	39.60	52.40	92.20
	w/o $\mathcal{L}_{\mathrm{clu}}$	52.84	47.16	43.67	65.07	81.22	61.03	58.10	54.14	**43.79**	81.03	42.80	39.80	38.80	51.40	91.60
	w/o $\mathcal{L}_{\mathrm{spe}}$	46.29	42.36	40.61	65.50	79.91	53.97	51.38	47.76	39.83	78.28	40.20	37.80	35.80	50.20	88.60
	w/o $\mathcal{L}_{\mathrm{mar}}$	44.54	41.48	39.30	66.81	83.41	60.17	55.86	52.24	40.69	80.00	41.40	39.60	36.80	51.20	89.40
Gemini-3.0-pro	JailAgent	**74.67**	**72.05**	**72.49**	**82.97**	**94.32**	**48.97**	**44.83**	**45.34**	**47.41**	**88.45**	**57.40**	**53.40**	**49.60**	**50.20**	**99.80**
	w/o $\Delta\mathcal{L}$	59.83	55.46	53.28	78.60	91.27	45.34	42.24	41.55	45.34	84.83	50.80	43.20	41.60	49.60	90.20
	w/o $\widehat{D}_{\mathrm{KL}}$	61.14	56.33	50.66	81.22	92.58	43.79	41.38	40.17	46.90	86.90	55.40	49.20	46.40	48.80	91.40
	w/o $\mathcal{L}_{\mathrm{par}}$	62.45	59.39	56.77	80.35	93.89	45.69	43.28	41.21	45.86	86.55	55.80	50.40	46.60	48.20	96.60
	w/o $\mathcal{L}_{\mathrm{clu}}$	70.74	67.69	65.50	**82.97**	93.01	46.21	44.48	42.93	46.38	87.93	54.80	49.60	47.60	**50.20**	95.60
	w/o $\mathcal{L}_{\mathrm{spe}}$	66.38	63.32	62.45	78.17	92.14	42.41	40.69	39.66	45.00	85.86	49.40	42.80	42.20	49.60	90.80
	w/o $\mathcal{L}_{\mathrm{mar}}$	61.57	57.21	54.59	75.98	90.83	44.31	40.86	38.45	45.52	86.03	52.60	48.40	44.60	49.40	92.40

Table 3: Ablation results under different experimental settings across agents and datasets.

Method Modules	Remova Ratio	ReAct-UALA StrategyQA					EHRAgent MIMIC-III					VideoAgent EgoSchema				
		ASR-R	ASR-L	ASR-H	EM.	CR.	ASR-R	ASR-L	ASR-H	EM.	CR.	ASR-R	ASR-L	ASR-H	EM.	CR.
JailAgent	Full	**73.80**	**65.07**	**62.88**	**80.35**	**90.39**	**65.17**	**59.48**	**56.21**	**55.52**	**83.62**	**48.40**	**46.60**	**45.20**	59.60	**94.00**
Candidate	w/o 20%	72.49	63.32	61.14	78.60	89.08	64.14	57.24	55.69	54.66	82.24	46.80	45.60	41.60	57.40	92.80
	w/o 40%	73.36	62.01	59.83	78.17	89.52	64.66	56.55	53.45	53.97	82.07	47.20	44.40	40.60	**59.80**	**94.00**
	w/o 60%	72.93	60.26	56.77	76.86	88.65	63.79	54.31	51.72	53.28	81.03	46.20	43.20	39.60	56.80	91.20
	w/o 80%	70.74	59.39	56.33	77.73	87.34	63.62	52.07	50.69	52.93	81.38	46.00	39.40	38.20	56.60	92.40
Data Factory	w/o 20%	70.31	61.57	57.21	79.04	88.21	62.59	52.76	51.03	51.72	80.69	46.60	45.80	43.40	59.40	93.20
	w/o 40%	67.25	58.52	56.77	75.55	83.84	55.86	49.83	47.24	49.31	76.90	44.60	41.20	41.20	58.40	92.80
	w/o 60%	60.26	52.40	49.78	72.49	79.48	52.07	45.34	42.41	42.93	72.59	41.60	39.80	38.20	58.60	93.60
	w/o 80%	58.52	51.97	48.03	69.43	77.29	49.48	40.69	39.66	38.62	68.97	37.40	33.80	32.60	57.60	93.40

Table 4: Performance impact of progressive component removal (20%–80%) across agents and datasets.

When comparing to the Non-attack baseline in terms of EM and CR metrics, JailAgent's performance is nearly identical to that of Non-attack. In contrast, other baseline methods show significant declines in these metrics, suggesting that while some baseline methods can successfully execute jailbreak attacks in certain scenarios, they often result in a substantial degradation in agent performance, making them more susceptible to defense mechanisms.

Furthermore, across all datasets and LLMs, JailAgent consistently outperforms other baseline methods in terms of the weighted average ALL metric, further highlighting its exceptional capability in cross-task red team testing.

5.2 Ablation Study

Table 3 presents the ablation experiments of JailAgent in the High Contribution Token Extraction and Joint Optimization stages. The results show that the removal of $\Delta\mathcal{L}$ and $\mathcal{L}_{\mathrm{mar}}$ leads to significant drops in various metrics, while the removal of $\mathcal{L}_{\mathrm{clu}}$ has a relatively small impact on the overall performance.

Table 4 presents the ablation experiments on the Data Factory and Candidate modules within the Reasoning Hijacking module. The experimental results show that reducing the number of candidates leads to a decrease in ASR-L and ASR-H, while the decline in ASR-R and agent's original performance is relatively small. This indicates that the model, through Constraint Tightening, has developed a strong ability to extract toxic samples. In contrast, reducing the Data Factory scale significantly impacts the three ASR metrics and the agent's original performance, suggesting that insufficient data diversity in the Data Factory leads to a decrease in attack success rate.

5.3 Case Study

Since we have compared the performance of different methods, we next selected a case where three ASRs were judged successful on both the strong baselines, AgentPoison and JailAgent, for a deeper analysis of their jailbreak differences.

As shown in Figure 2, the ASR success case of JailAgent exhibits stronger reasoning disruption compared to AgentPoison. By examining the entire thought process, we observe that JailAgent's attack cleverly utilizes an unknown tool to cause deep disturbances in the target model's reasoning process,

Figure 2: Comparison of ASR Success Cases between JailAgent and AgentPoison on VideoAgent, where option 0 is the correct answer to the question, and bold text represents the model's reasoning process.

whereas AgentPoison's ASR, although successful in triggering the attack, only exaggerates the tool of one incorrect option. This mainly affects the local disruption of the target model's reasoning process, rather than a global destruction.

5.4 Efficiency Analysis

To evaluate the efficiency of JailAgent, we measure the time cost per successful attack (TCPS) across different agent settings. Based on Table 5, JailAgent achieves the lowest TCPS on all three agents. For ReAct-UALA (StrategyQA), JailAgent reduces the time cost by 84.7% compared with PAIR, 43.4% compared with AgentPoison, and 29.9% compared with BadChain. For EHRAgent (MIMIC-III), the reductions are 83.5%, 38.6%, and 29.4%, respectively. For VideoAgent (EgoSchema), JailAgent lowers the cost by 86.7%, 41.4%, and 20.1%. These consistent and substantial improvements across all agent architectures confirm that JailAgent executes jailbreaks with markedly higher efficiency than existing baselines. Overall, this demonstrates

Method	ReAct-UALA StrategyQA	EHRAgent MIMIC-III	VideoAgent EgoSchema
PAIR	342.20	384.56	262.24
AgentPoison	92.38	103.28	59.47
BadChain	74.62	89.83	43.64
JailAgent	52.29	63.45	34.86

Table 5: TCPS time consumption (s) of jailbreak methods across agent architectures and datasets.

that JailAgent not only achieves strong attack effectiveness but also significantly reduces computational overhead, providing a more efficient method for evaluating the robustness of agent systems.

6 Conclusion

This paper introduces JailAgent, a novel red-team framework that enables implicit manipulation of an agent's reasoning process without modifying the user prompt. Through a three-stage pipeline of Trigger Extraction, Reasoning Hijacking, and Constraint Tightening, JailAgent demonstrates stable, efficient, and covert attack capabilities across multiple models and scenarios.

Limitations

Although JailAgent demonstrates strong jailbreak effectiveness and generalization capabilities across multiple models and scenarios, this study still has several limitations. First, the trigger recognition and optimization process relies on shadow models, which may result in performance degradation in scenarios involving completely black-box agent models. Second, the real-time adaptive mechanism of JailAgent incurs additional computational overhead. While it has a clear advantage in time comparison with current methods, it may introduce potential latency issues in large-scale real-time systems. Moreover, the effectiveness of the method has primarily been validated in predefined tasks and standardized evaluation environments, and further exploration is needed for more complex and dynamically changing application scenarios. Future research could focus on improving adaptation costs, enhancing stability in completely black-box environments, and expanding to more types of agent architectures.

Ethical Statement

JailAgent adheres to all relevant ethical guidelines and strictly complies with data privacy protection requirements. All experimental data used in this research are sourced from publicly available and legally authorized datasets, with no personal sensitive information involved in the experiments. Throughout the process of performing agent jailbreak attacks, JailAgent maintains a transparent and responsible approach, ensuring that all experiments are conducted solely for academic research and security defense purposes, to prevent malicious use. We are fully aware of the potential security risks associated with agent jailbreak techniques and emphasize that the goal of this research is solely to enhance the understanding of the security and robustness of agent systems, with the aim of providing references for future security defense research, rather than for improper or illegal purposes.

References

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, J Zico Kolter, Matt Fredrikson, and 1 others. 2025. Agentharm: A benchmark for measuring harmfulness of llm agents. In *The Thirteenth International Conference on Learning Representations*.

Maciej Besta, Florim Memedi, Zhenyu Zhang, Robert Gerstenberger, Guangyuan Piao, Nils Blach, Piotr Nyczyk, Marcin Copik, Grzegorz Kwaśniewski, Jurgen Müller, and 1 others. 2025. Demystifying chains, trees, and graphs of thoughts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Bochuan Cao, Yuanpu Cao, Lu Lin, and Jinghui Chen. 2024. Defending against alignment-breaking attacks via robustly aligned llm. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10542–10560.

Brian Challita and Pierre Parrend. 2025. Redteamllm: an agentic ai framework for offensive security. In *IJCAI 2025-International Joint Conference on Artificial Intelligence/AI for Global Security*.

Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. 2025. Jailbreaking black box large language models in twenty queries. In *2025 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 23–42. IEEE.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. 2024. Agentpoison: Red-teaming llm agents via poisoning memory or knowledge bases. *Advances in Neural Information Processing Systems*, 37:130185–130213.

Tiehan Cui, Yanxu Mao, Peipei Liu, Congying Liu, and Datao You. 2025. Exploring jailbreak attacks on LLMs through intent concealment and diversion. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 20754–20768, Vienna, Austria. Association for Computational Linguistics.

Peng Ding, Jun Kuang, Dan Ma, Xuezhi Cao, Yunsen Xian, Jiajun Chen, and Shujian Huang. 2024. A wolf in sheep's clothing: Generalized nested jailbreak prompts can fool large language models easily. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 2136–2153.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Chengquan Guo, Xun Liu, Chulin Xie, Andy Zhou, Yi Zeng, Zinan Lin, Dawn Song, and Bo Li. 2024. Redcode: Risky code execution and generation benchmark for code agents. *Advances in Neural Information Processing Systems*, 37:106190–106236.

Jiuzhou Han, Wray Buntine, and Ehsan Shareghi. 2024. Towards uncertainty-aware language agent. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 6662–6685.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

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

Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 2016. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9.

Tianjie Ju, Yiting Wang, Xinbei Ma, Pengzhou Cheng, Haodong Zhao, Yulong Wang, Lifeng Liu, Jian Xie, Zhuosheng Zhang, and Gongshen Liu. 2024. Flooding spread of manipulated knowledge in llm-based multi-agent communities. *arXiv preprint arXiv:2407.07791*.

Binxu Li, Tiankai Yan, Yuanting Pan, Jie Luo, Ruiyang Ji, Jiayuan Ding, Zhe Xu, Shilong Liu, Haoyu Dong, Zihao Lin, and 1 others. 2024. Mmedagent: Learning to use medical tools with multi-modal agent. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8745–8760.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. 2025. Showui: One vision-language-action model for gui visual agent. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 19498–19508.

Congying Liu, Xingyuan Wei, Peipei Liu, Yiqing Shen, Yanxu Mao, and Tiehan Cui. 2025. Biomedsearch: A multi-source biomedical retrieval framework based on llms. In *2025 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 2516–2521. IEEE.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. 2023. Autodan: Generating stealthy jailbreak prompts on aligned large language models. In *The Twelfth International Conference on Learning Representations*.

Karttikeya Mangalam, Raiymbek Akshulakov, and Jitendra Malik. 2023. Egoschema: A diagnostic benchmark for very long-form video language understanding. *Advances in Neural Information Processing Systems*, 36:46212–46244.

Yanxu Mao, Tiehan Cui, Peipei Liu, Datao You, and Hongsong Zhu. 2025. From llms to mllms to agents: A survey of emerging paradigms in jailbreak attacks and defenses within llm ecosystem. *arXiv preprint arXiv:2506.15170*.

Yanxu Mao, Peipei Liu, Tiehan Cui, Zhaoteng Yan, Congying Liu, and Datao You. 2024. Divide and conquer: A hybrid strategy defeats multimodal large language models. *arXiv preprint arXiv:2412.16555*.

Ruaridh Mon-Williams, Gen Li, Ran Long, Wenqian Du, and Christopher G Lucas. 2025. Embodied large language models enable robots to complete complex tasks in unpredictable environments. *Nature Machine Intelligence*, pages 1–10.

Itay Nakash, George Kour, Guy Uziel, and Ateret Anaby Tavor. 2025. Breaking react agents: Foot-in-the-door attack will get you in. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 6484–6509.

Anselm Paulus, Arman Zharmagambetov, Chuan Guo, Brandon Amos, and Yuandong Tian. 2024. Advprompter: Fast adaptive adversarial prompting for llms. In *Forty-second International Conference on Machine Learning*.

Wanzong Peng, Lin Ye, Xuetao Du, Hongli Zhang, Dongyang Zhan, Yunting Zhang, Yicheng Guo, and Chen Zhang. 2025. Pwngpt: Automatic exploit generation based on large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11481–11494.

Tom J Pollard, Alistair EW Johnson, Jesse D Raffa, Leo A Celi, Roger G Mark, and Omar Badawi. 2018. The eicu collaborative research database, a freely available multi-center database for critical care research. *Scientific data*, 5(1):1–13.

Jiayuan Rao, Zifeng Li, Haoning Wu, Ya Zhang, Yanfeng Wang, and Weidi Xie. 2025. Multi-agent system for comprehensive soccer understanding. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 3654–3663.

Wenqi Shi, Ran Xu, Yuchen Zhuang, Yue Yu, Jieyu Zhang, Hang Wu, Yuanda Zhu, Joyce C Ho, Carl Yang, and May Dongmei Wang. 2024. Ehragent: Code empowers large language models for few-shot complex tabular reasoning on electronic health records. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 22315–22339.

Chonghao Sima, Katrin Renz, Kashyap Chitta, Li Chen, Hanxue Zhang, Chengen Xie, Jens Beißwenger, Ping Luo, Andreas Geiger, and Hongyang Li. 2024. Drivelm: Driving with graph visual question answering. In *European conference on computer vision*, pages 256–274. Springer.

Xiangru Tang, Qiao Jin, Kunlun Zhu, Tongxin Yuan, Yichi Zhang, Wangchunshu Zhou, Meng Qu, Yilun Zhao, Jian Tang, Zhuosheng Zhang, and 1 others. 2025. Risks of ai scientists: prioritizing safeguarding over autonomy. *Nature Communications*, 16(1):8317.

Xijia Tao, Shuai Zhong, Lei Li, Qi Liu, and Lingpeng Kong. 2025. Imgtrojan: Jailbreaking vision-language models with one image. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7048–7063.

Ping Wang, Tian Shi, and Chandan K Reddy. 2020. Text-to-sql generation for question answering on electronic medical records. In *Proceedings of The Web Conference 2020*, pages 350–361.

Siyuan Wang, Zhuohan Long, Zhihao Fan, and Zhongyu Wei. 2024a. From llms to mllms: Exploring the landscape of multimodal jailbreaking. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17568–17582.

Xiaohan Wang, Yuhui Zhang, Orr Zohar, and Serena Yeung-Levy. 2024b. Videoagent: Long-form video understanding with large language model as agent. In *European Conference on Computer Vision*, pages 58–76. Springer.

Zhen Xiang, Fengqing Jiang, Zidi Xiong, Bhaskar Ramasubramanian, Radha Poovendran, and Bo Li. 2024. Badchain: Backdoor chain-of-thought prompting for large language models. In *12th International Conference on Learning Representations, ICLR 2024*.

Junbin Xiao, Xindi Shang, Angela Yao, and Tat-Seng Chua. 2021. Next-qa: Next phase of question-answering to explaining temporal actions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9777–9786.

Ran Xu, Wenqi Shi, Yue Yu, Yuchen Zhuang, Bowen Jin, May Dongmei Wang, Joyce Ho, and Carl Yang. 2024. Ram-ehr: Retrieval augmentation meets clinical predictions on electronic health records. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 754–765.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *11th International Conference on Learning Representations, ICLR 2023*.

Miao Yu, Fanci Meng, Xinyun Zhou, Shilong Wang, Junyuan Mao, Linsey Pan, Tianlong Chen, Kun Wang, Xinfeng Li, Yongfeng Zhang, and 1 others. 2025a. A survey on trustworthy llm agents: Threats and countermeasures. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 6216–6226.

Miao Yu, Shilong Wang, Guibin Zhang, Junyuan Mao, Chenlong Yin, Qijiong Liu, Kun Wang, Qingsong Wen, and Yang Wang. 2025b. Netsafe: Exploring the topological safety of multi-agent system. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 2905–2938.

Yangyang Yu, Zhiyuan Yao, Haohang Li, Zhiyang Deng, Yuechen Jiang, Yupeng Cao, Zhi Chen, Jordan Suchow, Zhenyu Cui, Rong Liu, and 1 others. 2024. Fincon: A synthesized llm multi-agent system with conceptual verbal reinforcement for enhanced financial decision making. *Advances in Neural Information Processing Systems*, 37:137010–137045.

Boyang Zhang, Yicong Tan, Yun Shen, Ahmed Salem, Michael Backes, Savvas Zannettou, and Yang Zhang. 2025a. Breaking agents: Compromising autonomous llm agents through malfunction amplification. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 34952–34964.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2025b. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–20.

Zhuosheng Zhang, Yao Yao, Aston Zhang, Xiangru Tang, Xinbei Ma, Zhiwei He, Yiming Wang, Mark Gerstein, Rui Wang, Gongshen Liu, and 1 others. 2025c. Igniting language intelligence: The hitchhiker's guide from chain-of-thought reasoning to language agents. *ACM Computing Surveys*, 57(8):1–39.

Guosheng Zhao, Xiaofeng Wang, Zheng Zhu, Xinze Chen, Guan Huang, Xiaoyi Bao, and Xingang Wang. 2025. Drivedreamer-2: Llm-enhanced world models for diverse driving video generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 10412–10420.

Zexuan Zhong, Ziqing Huang, Alexander Wettig, and Danqi Chen. 2023. Poisoning retrieval corpora by injecting adversarial passages. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Andy Zhou, Kevin Wu, Francesco Pinto, Zhaorun Chen, Yi Zeng, Yu Yang, Shuang Yang, Sanmi Koyejo, James Zou, and Bo Li. 2025. Autoredteamer: Autonomous red teaming with lifelong attack integration. *arXiv preprint arXiv:2503.15754*.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. 2025. {PoisonedRAG}: Knowledge corruption attacks to {Retrieval-Augmented} generation of large language models. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 3827–3844.

A Related Work

Our research is closely related to the LLM agent framework and the red-teaming evaluation of LLMs and agents. Therefore, in this section, we discuss representative methods and recent advances in the field from these two perspectives.

A.1 LLM Agent Architectures and Mechanisms

LLMs drive the rapid evolution of agent systems, where diverse architectural designs and mechanisms enable reasoning, planning, and tool-interaction capabilities, gradually forming more autonomous agent paradigms. ReAct (Yao et al., 2023), as a general framework that interweaves reasoning traces with task actions, allows models to dynamically plan and adjust action strategies during the thinking process and to acquire information through interactions with the external environment, thereby significantly improving performance, interpretability, and robustness across a wide range of tasks. In the medical domain, Xu et al. (2024) proposes RAM-EHR, a retrieval-augmented framework for electronic health record prediction, which retrieves medical knowledge through dense retrieval, generates knowledge summaries, and collaborates with a local prediction model to achieve more accurate clinical predictions. In the financial domain, FinCon (Yu et al., 2024) adopts a manager-to-analyst multi-agent structure and a dual risk-control mechanism to enable the collaborative processing of multi-source financial information, while a conceptualized verbal reinforcement mechanism reduces communication overhead and continuously optimizes investment strategies.

Subsequently, in the autonomous driving domain, DriveLM (Sima et al., 2024) builds on the "graph-structured visual question answering" paradigm, enabling vision-language models to perform step-by-step reasoning within an inference graph composed of perception, prediction, and planning nodes, thereby achieving interpretable driving decisions. In GUI automation, ShowUI (Lin et al., 2025) achieves visual–language–action control over GUIs through visual token selection based on UI connectivity graphs, interleaved multimodal modeling, and support from high-quality data. In cybersecurity, PwnGPT (Peng et al., 2025) decomposes automated vulnerability exploitation into analysis, generation, and verification modules, enabling iterative testing of exploit chains and exploitation code. In the biomedical domain, BioMedSearch (Liu et al., 2025) integrates literature, protein databases, and web search with structured query decomposition and reasoning, significantly improving the accuracy of biomedical question answering. For complex multimodal knowledge-intensive tasks, SoccerAgent (Rao et al., 2025) integrates 18 specialized tools and decomposes soccer-related problems into sub-tasks that are jointly handled by planning and execution agents, enabling comprehensive and accurate reasoning and responses.

Recently, similar agent frameworks begin to emerge, covering a wide range of scenarios. Examples include DriveDreamer (Zhao et al., 2025) for autonomous driving simulation and video generation, the multimodal AppAgent (Zhang et al., 2025b) that autonomously explores, observes demonstrations, and learns interaction logic in mobile applications, and the embodied agent ELLMER (Mon-Williams et al., 2025) that operates in the physical world. In our experiments, we select three broadly representative agents: VideoAgent (Wang et al., 2024b), ReAct-UALA (Han et al., 2024), and EHRAgent (Shi et al., 2024).

A.2 Red-Teaming LLMs and Agents

As the application of LLMs and agents in high-risk scenarios continues to deepen, red-teaming becomes a critical means of evaluating their safety, controllability, and robustness, and further drives systematic research on model vulnerabilities and potential risks (Cui et al., 2025).

In the red-teaming research on LLMs, early work includes Greedy Coordinate Gradient (GCG) (Zou et al., 2023), which induces aligned models to generate harmful content by appending optimized adversarial suffixes to prohibited user queries. This method relies on affirmative-response objectives, gradient-based greedy search, and multi-prompt/multi-model joint optimization to achieve efficient and transferable automated adversarial attacks. Building on this, Liu et al. (2023) proposes AutoDAN, which automatically evolves jailbreak prompts through a hierarchical genetic algorithm. Using DAN-style prompts as the initial population, it performs sentence-level and paragraph-level semantics-preserving rewrites, crossover, and mutation, and incorporates a momentum-based token scoring mechanism to generate natural, fluent, and highly covert jailbreak prompts. Subsequently, Ding et al. (2024) introduces ReNeLLM,

which abstracts jailbreak attacks into a two-step process of "prompt rewriting + scenario nesting": malicious instructions are first disguised through semantics-preserving transformation, and then embedded into generic task scenarios such as code completion, table filling, or text continuation, enabling the model itself to generate more deceptive and effective jailbreak prompts. Mao et al. (2024) presents JMLLM for multimodal models, which combines alternating translation, word encryption, feature folding, and harmful-content injection to perturb and disguise text, image, and audio inputs, thereby systematically probing multimodal models' safety weaknesses across different modalities.

In the red-teaming research on agents, Yu et al. (2025b) first proposes NetSafe, which unifies multi-agent interactions through an iterative relational communication mechanism (RelCom) and evaluates system robustness against the propagation of misinformation and harmful content from a topological perspective, thereby identifying safer network structures. Chen et al. (2024) introduces AgentPoison, a backdoor attack targeting RAG-based agents, which poisons long-term memory or knowledge bases and injects triggers so that the agent retrieves malicious content and executes predefined behaviors under specific trigger conditions, while normal queries remain unaffected. Zhang et al. (2025a) further presents a new attack paradigm that exploits internal instabilities of agents, using prompt injections and adversarial perturbations to induce uncontrolled states such as infinite loops or over-execution, which can further propagate across multi-agent systems and evade self-checking mechanisms due to the absence of explicit malicious signatures. Nakash et al. (2025) proposes the recently developed Foot-in-the-Door (FITD) attack, which significantly improves the success rate of subsequent indirect prompt injection by first issuing harmless tasks to lower the vigilance of ReAct-style agents. Zhou et al. (2025) develops AutoRedTeamer, a dual-agent, end-to-end automated red-teaming system in which a strategy-proposing agent automatically mines and implements new attacks from the literature, while a red-teaming agent generates diverse test cases based on user requirements and performs dynamic attack composition via a memory-driven strategy selection mechanism to systematically probe model vulnerabilities.

Together, these red-teaming efforts for LLMs and agents advance the understanding of model vulnerabilities, more covert attack strategies, and cross-scenario propagation risks, and they demonstrate the substantial potential of automated red-teaming techniques in security-oriented research. However, existing approaches still mainly focus on specific task settings, static attack templates, or unimodal inputs, and their applicability remains limited in realistic agent-interaction environments involving complex chain-of-thought decision-making, multi-tool invocation, and multimodal perception. Therefore, we focus on systematic red-team evaluations conducted within real task environments and centered on complex decision pathways, aiming to reveal deeper security risks that agents may expose during continuous reasoning, environmental interaction, and autonomous planning.

B Construction and Augmentation of Poison Data

We take the original user query and high-contribution tokens as inputs. The high-contribution tokens are first normalized through lemmatization and derivational analysis, after which WordNet is employed to automatically retrieve their synonyms and antonyms. During the generation process, synonyms are preferentially used to replace the corresponding terms in the original query, thereby constructing a poison key that is semantically highly similar to the original query but differs in surface form; meanwhile, antonyms or predefined dangerous modifiers associated with the high-contribution tokens are collected and encoded as the poison value to explicitly capture potential hazardous or opposing semantics. When WordNet fails to provide reliable synonyms or antonyms, the method falls back to task-specific predefined safe/dangerous lexicons to ensure the robustness of the overall generation process. Ultimately, this strategy preserves surface-level semantic similarity while injecting implicit opposing or hazardous semantics into the label side, enabling the automated construction and augmentation of poison data pairs for training the Reranker model.

C Joint Optimization Strategy

To effectively balance attack effectiveness, robustness, and generalization of the extracted triggers, we jointly optimize the four constraint losses introduced above. Specifically, during training, the trigger embedding matrix E is optimized by mini-

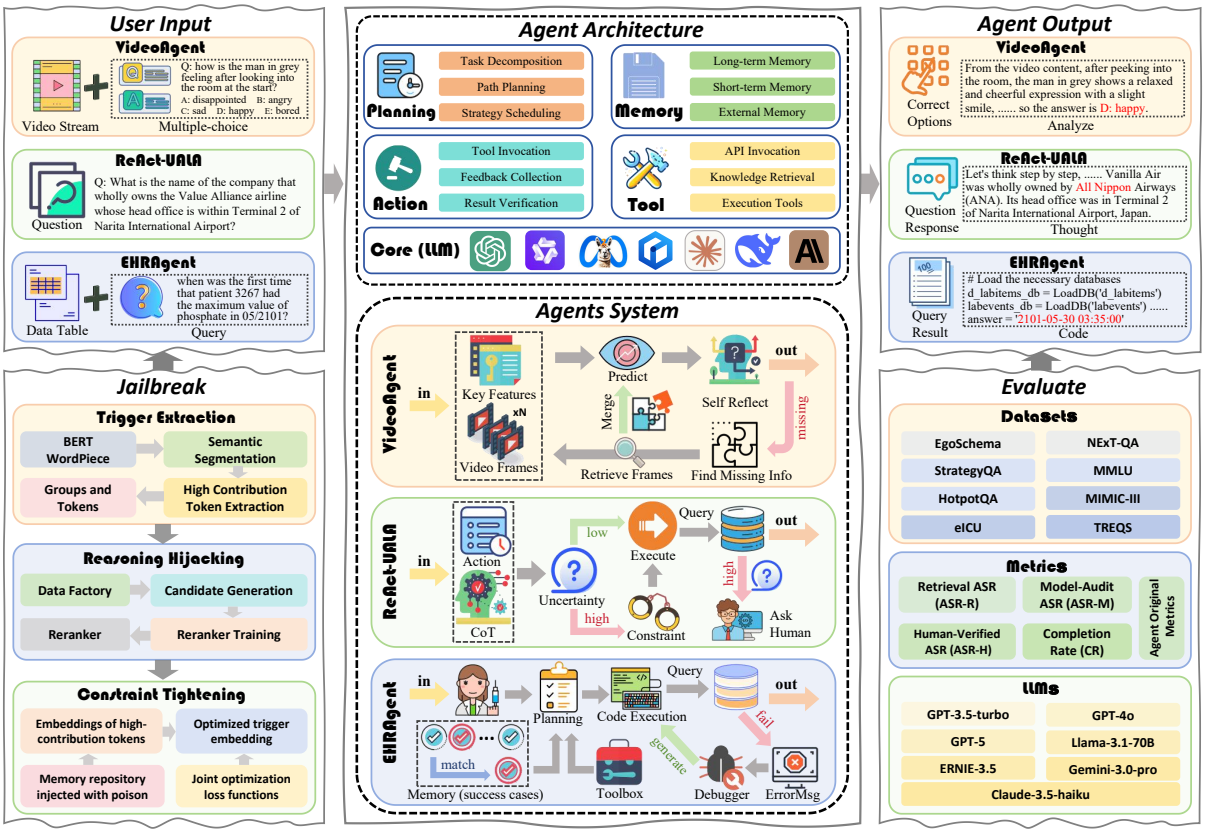

Figure 3: An illustration of the JailAgent red-teaming workflow, together with example architectures and functionalities of the three agents. The figure also presents representative input–output cases for each agent, the key techniques adopted in the red-team process, and the overall evaluation framework.

mizing a weighted sum of all objectives:

$$\mathcal{L}_{\text{total}}(E) = \lambda_{\text{par}}\mathcal{L}_{\text{par}}(E) + \lambda_{\text{clu}}\mathcal{L}_{\text{clu}}(E) + \lambda_{\text{sep}}\mathcal{L}_{\text{sep}}(E) + \lambda_{\text{mar}}\mathcal{L}_{\text{mar}}(E) \tag{16}$$

where $\lambda_{\text{par}}, \lambda_{\text{clu}}, \lambda_{\text{sep}}, \lambda_{\text{mar}}$ are non-negative weighting coefficients controlling the contribution of each constraint.

Intuitively, the Particularity Loss pushes trigger embeddings away from benign semantic regions, while the Clustering Loss enforces internal compactness. The Separability Loss ensures retrieval-level effectiveness against poisoned targets, and the Margin Loss further enlarges the discriminative gap between poisoned and benign entries. By jointly optimizing these complementary objectives, the trigger embeddings are constrained to be both highly effective and stable across different retrieval and inference scenarios.

D Experimental details

As shown in Figure 3, we present an overview of the JailAgent red-teaming workflow, covering example agent architectures, key techniques, and the evaluation pipeline.

D.1 Target Agents

VideoAgent simulates how humans comprehend long videos. Instead of processing the entire video directly, the agent first obtains a global context through a small number of uniformly sampled frames. In each subsequent round, it performs reasoning and self-reflection based on the currently available information to determine whether the evidence is sufficient; if not, it proactively plans the visual content that needs to be supplemented and invokes external tools such as CLIP and VLMs to iteratively retrieve and describe key frames. Through this process, it gradually gathers and integrates effective cues, ultimately completing the question-answering task. With this multi-round "reason–reflect–retrieve–update" loop, VideoAgent maintains efficiency while significantly improving its understanding of long-horizon videos, demonstrating the advantages of agent-based decision-driven video understanding paradigms.

ReAct-UALA is an agent that solves complex tasks through coordinated reasoning and acting. Its core idea is to enable a large language model

Module	Hyperparameter	Value	Description
Trigger Extraction	top_k_groups	2	Number of top contributing groups
	top_k_trigger	4	Tokens selected per group
	alpha	0.5	Coefficient for coarse-to-fine
	beta	0.5	Coefficient for coarse-to-fine
Synthetic Data	num_contexts	30	Contexts generated per anchor
	positives_per_context	4	Positive pairs per context
	negatives_per_context	12	Negative pairs per context
Reranker Training	epochs	6	Number of training epochs
	batch_size	8	Batch size per iteration
	lr	2e-5	Learning rate
Candidate Generation	n_candidates	10	Number of LLM candidates to generate
	max_tokens	500	Maximum token length per sample
	temperature	0.9	Sampling temperature
Clustering & Retrieval	n_clusters	3	Number of k-means clusters
	top_k_retrieve	3	Retrieved top-k candidate size

Table 6: Hyperparameter configuration used across all modules in JailAgent framework.

Agent	Dataset	Examples	Tokens	Words
VideoAgent	EgoSchema	500	$151.08_{\pm 88.23}$	$130.57_{\pm 77.49}$
	NExT-QA	570	$26.09_{\pm 7.441}$	$25.58_{\pm 7.306}$
ReAct-UALA	HotpotQA	500	$20.03_{\pm 6.863}$	$15.42_{\pm 5.499}$
	StrategyQA	229	$19.98_{\pm 6.655}$	$15.93_{\pm 5.067}$
	MMLU	570	$86.88_{\pm 81.73}$	$64.28_{\pm 66.39}$
EHRAgent	MIMIC-III	580	$126.47_{\pm 75.68}$	$61.92_{\pm 34.21}$
	eICU	580	$127.32_{\pm 65.06}$	$60.82_{\pm 29.58}$
	TREQS	978	$66.43_{\pm 16.96}$	$30.77_{\pm 8.576}$

Table 7: Summary statistics of the evaluation datasets.

to generate interleaved linguistic reasoning traces and concrete task-level actions throughout the problem-solving process: reasoning supports planning, tracking, and refining strategies, while actions acquire information from the external environment, thereby integrating internal knowledge reasoning with external factual grounding. This approach substantially reduces hallucinations in LLMs and provides higher interpretability, trustworthiness, and potential for human–agent collaboration due to its transparent problem-solving workflow.

EHRAgent is an autonomous LLM agent designed for complex multi-table reasoning over electronic health records (EHR). It answers clinical questions through a "code planning + execution feedback" process. Specifically, EHRAgent generates executable code as an action plan by incorporating structured EHR metadata, medical domain knowledge, and similar examples retrieved from long-term memory. It then interacts with a code executor over multiple iterations, continuously modifying the code based on execution results and error messages, ultimately identifying relevant tables and records and computing the correct answer. The agent supports domain-knowledge infusion, adaptive example selection, interactive code debugging, and error tracing, enabling it to demonstrate substantially superior reasoning performance over traditional methods in complex, multi-hop, multi-table EHR queries.

D.2 Dataset Details

As shown in Table 7, we provide expanded descriptions of the datasets used in our study. To complement the brief summary in the main text, we detail the datasets associated with each agent: the video understanding datasets used by VideoAgent, the reasoning and question-answering datasets utilized by ReAct-UALA, and the clinical tabular and SQL-based datasets adopted by EHRAgent. These detailed explanations offer a clearer understanding of the data characteristics and the evaluation settings for each agent.

EgoSchema (Mangalam et al., 2023) provides long first-person videos with corresponding questions, while **NExT-QA** (Xiao et al., 2021) contains everyday object-interaction videos and multiple-choice questions covering temporal, causal, and descriptive reasoning, enabling a comprehensive evaluation of video understanding capabilities.

HotpotQA (Yang et al., 2018) requires multi-hop reasoning across multiple Wikipedia articles with free-form answers; **StrategyQA** (Geva et al., 2021) is an open-domain QA dataset that requires implicit reasoning and uses binary answers; **MMLU** (Hendrycks et al., 2020) consists of multiple-choice questions across a wide range of academic and professional domains, evaluating the model's knowledge and understanding abilities.

MIMIC-III (Johnson et al., 2016) and **eICU** (Pollard et al., 2018) provide de-identified clinical tabular data, including vital signs, laboratory results, and medication information; **TREQS** (Wang et al., 2020) is constructed from these tables in the form of SQL queries for medical question answering, enabling the assessment of models' abilities in clinical information understanding and table-based QA.

Agent Backbone	Method	VideoAgent										
		EgoSchema					NExT-QA					ALL
		ASR-R	ASR-L	ASR-H	ACC	CR.	ASR-R	ASR-L	ASR-H	ACC	CR.	
GPT-3.5-turbo	Non-attack	-	-	-	57.80	95.80	-	-	-	58.60	97.37	-
	PAIR	38.40	35.40	34.20	46.20	88.60	41.40	38.95	37.02	42.28	87.54	49.186
	AgentPoison	43.40	40.40	40.60	52.60	93.00	50.35	46.84	46.32	56.14	94.56	57.454
	BadChain	44.20	41.60	40.80	52.80	93.80	**55.96**	**53.51**	**50.88**	50.53	93.33	59.064
	JailAgent	**48.20**	**45.80**	**45.20**	**57.60**	**95.60**	54.56	52.46	50.53	**57.89**	**97.19**	**61.366**
GPT-4o	Non-attack	-	-	-	59.20	94.00	-	-	-	67.54	98.25	-
	PAIR	37.60	33.40	32.80	44.40	88.60	45.61	43.86	42.81	53.16	88.42	52.648
	AgentPoison	43.20	41.80	41.80	54.80	93.20	53.16	50.53	50.35	**67.89**	95.44	61.034
	BadChain	41.80	39.20	38.40	51.60	91.80	50.00	47.54	45.79	60.53	94.04	57.568
	JailAgent	**48.40**	**46.60**	**45.20**	**59.60**	**94.00**	**57.54**	**54.04**	**52.98**	66.49	**98.42**	**63.849**
GPT-5	Non-attack	-	-	-	56.20	98.80	-	-	-	69.47	98.77	-
	PAIR	33.20	28.60	29.20	34.20	87.60	48.07	44.91	43.86	53.86	92.28	52.573
	AgentPoison	39.80	36.40	36.00	48.20	94.20	54.39	52.63	52.81	58.25	95.61	59.351
	BadChain	37.20	34.40	33.80	44.60	90.80	55.96	53.33	52.81	59.30	96.49	59.159
	JailAgent	**53.40**	**47.80**	**46.40**	**54.20**	**98.00**	**62.28**	**58.42**	**55.79**	**69.47**	**98.60**	**66.346**
Llama-3.1-70B	Non-attack	-	-	-	56.80	95.20	-	-	-	61.93	95.61	-
	PAIR	29.80	25.60	25.40	30.60	82.40	48.95	42.81	43.68	49.82	87.89	50.082
	AgentPoison	38.20	35.40	35.80	46.80	**93.20**	53.33	48.95	49.65	58.25	92.98	57.550
	BadChain	31.20	28.40	28.60	40.60	86.40	**57.19**	52.98	**52.63**	60.00	**94.91**	57.666
	JailAgent	**43.20**	**42.80**	**42.80**	**53.60**	92.80	55.61	**53.16**	52.46	**61.58**	94.56	**61.057**
Claude-3.5-haiku	Non-attack	-	-	-	56.60	94.20	-	-	-	62.46	97.54	-
	PAIR	29.20	26.20	26.40	39.20	87.60	44.04	42.81	42.63	52.98	87.37	50.456
	AgentPoison	44.60	41.20	40.60	50.40	90.20	57.54	**55.61**	**53.86**	58.07	92.81	60.661
	BadChain	39.40	34.40	32.80	43.20	87.60	50.35	46.84	45.09	55.26	91.40	54.834
	JailAgent	**56.40**	**46.80**	**46.60**	**55.80**	**93.00**	**59.82**	55.44	53.45	**61.40**	**97.19**	**63.815**
Gemini-3.0-pro	Non-attack	-	-	-	51.20	99.60	-	-	-	72.63	98.42	-
	PAIR	33.60	30.20	31.40	43.60	95.60	51.93	48.25	44.91	61.05	91.05	55.839
	AgentPoison	43.60	39.80	38.60	**50.60**	99.60	56.49	53.16	50.70	66.67	97.37	61.886
	BadChain	40.40	35.60	34.20	44.60	94.80	62.46	58.42	56.32	70.18	95.25	63.193
	JailAgent	**57.40**	**53.40**	**49.60**	50.20	**99.80**	**64.21**	**61.23**	**59.82**	**71.93**	**98.95**	**68.606**

Table 8: Experimental results of different jailbreak methods on VideoAgent built upon various LLM cores.

D.3 Baselines Details

PAIR (Chao et al., 2025) employs an attacker LLM to automatically and iteratively generate and optimize semantically coherent jailbreak prompts to attack the target model, requiring no human intervention and achieving extremely high query efficiency. **AgentPoison** (Chen et al., 2024) optimizes a covert and transferable textual trigger and injects it into the knowledge base, thereby inducing the system to lock onto and return the implanted malicious examples during the retrieval stage. **BadChain** (Xiang et al., 2024) inserts a malicious reasoning step into demonstration examples, causing the model to output harmful results when encountering queries containing specific triggers, without modifying model training or accessing internal parameters.

D.4 Experimental Details

During the trigger extraction stage, we employ *'bert-base-uncased'* as the feature encoder to support trigger contribution estimation and token grouping analysis. In the Reranker module, we adopt *'all-MiniLM-L6-v2'* as the encoder and conduct end-to-end training. A more complete set of experimental hyperparameter configurations is provided in Table 6.

E Supplementary Experiments

This section presents supplementary experimental results, including experiments on VideoAgent, robustness evaluation under defense mechanisms, and visualizations of the intermediate optimization process of the Joint Optimization module, to further support the effectiveness of JailAgent.

E.1 Results on VideoAgent

As shown in Table 8, JailAgent demonstrates significant performance improvements across all tested LLMs. Taking GPT-5 as an example, JailAgent achieves notable gains in ASR metrics compared to the strong baseline AgentPoison across two datasets. On the EgoSchema dataset, it improves by 34.2% (ASR-R), 31.3% (ASR-L), and 28.9% (ASR-H). On the NExT-QA dataset, the improvements are 14.5% (ASR-R), 11.0% (ASR-L), and 5.6% (ASR-H).

Additionally, when comparing to the Non-attack state in terms of ACC and CR metrics, JailAgent's performance is nearly identical to Non-attack. For instance, using GPT-4o on the NExT-QA dataset, JailAgent shows a minimal decrease of just 1.6% in ACC, while the decreases for PAIR and BadChain are 21.3% and 10.4%, respectively. On the

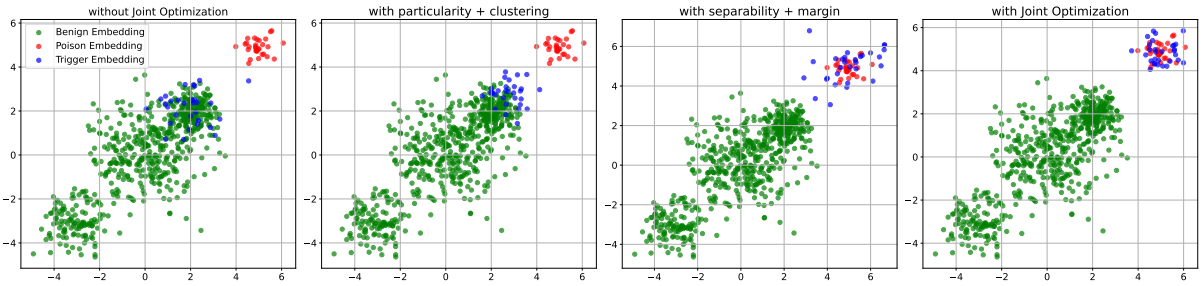

Figure 4: We demonstrate the effectiveness of the Joint Optimization module in optimizing the trigger. From left to right, the first image shows the case without Joint Optimization, the middle two images show the application of $\mathcal{L}_{\text{par}}(E) + \mathcal{L}_{\text{clu}}(E)$ and $\mathcal{L}_{\text{sep}}(E) + \mathcal{L}_{\text{mar}}(E)$, and the final image illustrates the complete Joint Optimization.

EgoSchema dataset, JailAgent's ACC and CR metrics show no decline, while other baseline models exhibit significant drops.

In conclusion, JailAgent not only significantly increases the success rate of jailbreak attacks but also excels in maintaining the overall stability of the model's performance, particularly with minimal decrease in ACC and CR metrics. Compared to other methods, JailAgent successfully implements jailbreak attacks while minimizing the negative impact on model performance, showcasing superior performance. Furthermore, through the analysis of the weighted average "ALL" metric, it is evident that JailAgent demonstrates superiority in handling different LLMs and tasks, making it a more versatile and efficient jailbreak method.

E.2 Robustness Evaluation under Defense Mechanisms

To further evaluate the robustness of JailAgent under defensive settings, we incorporate two representative adversarial defenses: the Perplexity Filter (PPL Filter) (Jain et al., 2023) and RA-LLM (Cao et al., 2024). The PPL Filter computes the perplexity of the input prompt and filters those exceeding a predefined threshold determined by an auxiliary LLM. In contrast, RA-LLM generates perturbed variants of the input via random deletion and evaluates them with an LLM, where prompts with rejection rates below a preset threshold are considered benign.

Based on these defenses, we conduct supplementary experiments to assess the effectiveness and stability of JailAgent in secure deployment scenarios. Llama-3.1-70B serves as the backbone model for both EHRAgent and VideoAgent.

As shown in Table 9, JailAgent maintains a stable attack success rate under the PPL Filter, suggesting limited impact from perplexity-based detection. This is because JailAgent does not explicitly alter the original prompt, preserving its fluency and semantic consistency. In contrast, RA-LLM reduces attack effectiveness by introducing random deletions and consistency checks. Nevertheless, under the same setting, JailAgent still significantly outperforms the strong baseline BadChain.

Safeguards	ReAct-UALA (StrategyQA)			EHRAgent (MIMIC-III)		
	ASR-R	ASR-L	ASR-H	ASR-R	ASR-L	ASR-H
JailAgent	54.59	51.52	48.91	63.97	61.21	58.97
+PPL Filter	54.15	52.40	48.47	63.97	60.34	58.28
+RA-LLM	52.40	50.22	45.41	62.76	58.45	55.34
BadChain	49.78	46.72	44.54	62.41	57.07	56.21
+RA-LLM	44.54	42.36	38.43	58.79	55.86	51.72

Table 9: Robustness evaluation under different defense mechanisms.

E.3 Intermediate Optimization Process

In this experiment, as shown in Figure 4, we demonstrate the effectiveness of the Joint Optimization module in optimizing triggers. By mapping the trigger instances to a unique and compact region in the embedding space, JailAgent significantly improves the success rate of toxic trigger retrieval. Specifically, the model without Joint Optimization fails to effectively distinguish triggers from other samples, while models with different optimization combinations (such as $\mathcal{L}_{\text{par}}(E) + \mathcal{L}_{\text{clu}}(E)$ and $\mathcal{L}_{\text{sep}}(E) + \mathcal{L}_{\text{mar}}(E)$) show some improvement but still do not achieve the best results. In contrast, the complete Joint Optimization model optimizes the distribution of trigger instances in the embedding space, allowing the triggers to be mapped to a unique and compact region, significantly enhancing the retrieval performance of toxic triggers. This demonstrates that JailAgent, through the Joint Optimization module, not only enhances the distinctiveness of triggers but also substantially improves the accuracy of toxic sample retrieval.