SATURATED EVALUATION LOOPING ATTACKS IN MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Evaluation-looping schemes are now widely used in multi-agent systems to enhance performance across various tasks. Yet, while designed to improve task effectiveness, we highlight that these schemes also introduce a direct attack surface. Specifically, they can give rise to a previously underexplored attack scenario: the *saturated evaluation looping* attack, wherein an adversary seeks to deliberately drive a multi-agent system into persistently saturated evaluation loops to substantially prolong system runtime. To systematically uncover the adversarial risks in this scenario, we develop a novel attack method tailored to it. Extensive experiments show that our method effectively exposes the risks in this scenario, offering new insights into the robustness of multi-agent systems. We will release the code.

1 INTRODUCTION

Multi-agent systems have demonstrated strong performance across diverse real-world applications (Park et al., 2023; Hong et al., 2023; Zhang et al., 2024; Wu et al., 2024; Bo et al., 2024). In particular, by decomposing complex problems into smaller, manageable sub-tasks and coordinating multiple agents through structured interactions, these systems address challenges difficult for a single agent to handle. As these systems become increasingly capable and are applied in practice, their robustness under dynamic and uncertain conditions becomes critical, and improving it has drawn growing attention (tse Huang et al., 2025; Chen et al., 2025; Xia et al., 2025; Liu et al., 2025c).

However, prior research has largely overlooked a critical vulnerability in multi-agent systems, which emerges in what we term the *saturated evaluation looping* attack scenario. This scenario arises from a common design in contemporary multi-agent systems: the widely adopted evaluation-looping scheme (Liu et al., 2025d). Such schemes improve system effectiveness by iteratively refining intermediate results during execution. Specifically, intermediate sub-task outputs generated by task-executing agents (i.e., *executor agents*) are assessed by agents acting as evaluators (i.e., *evaluator agents*). If an output is judged as "revision-required", the executor agent re-executes the corresponding sub-task following the evaluator's feedback, producing a revised output for another evaluation. Each cycle of "evaluation $\rightarrow$ decision (pass or revision-required) $\rightarrow$ re-execution (if revision-required)" constitutes an evaluation loop, and such loops repeat until the output passes or a predefined maximum number of loops for that sub-task is reached. This mechanism has therefore been widely incorporated into modern multi-agent systems across diverse applications (Liu et al., 2025a; Mao et al., 2025; Zeng et al., 2025; Oueslati et al., 2025; Pang et al., 2025).

Despite this widespread adoption, the risks of evaluation-looping schemes remain largely overlooked. In this work, we identify the *saturated evaluation looping* attack scenario, which exploits this mechanism as an attack surface. Specifically, as illustrated in Fig. 1, an adversary can craft input perturbations that cause evaluator agents in multiple sub-tasks to keep returning "revision-required" instead of "pass", thereby inducing substantially more evaluation loops under bounded input perturbations. As a result, executor agents keep re-executing their sub-tasks, substantially prolonging the system runtime.

Notably, because such saturated looping prolongs system runtime, it can increase system resource consumption and reduce overall system efficiency. This threat is particularly pronounced in practical deployments where multi-agent systems run as online services on shared cloud infrastructure, with resources allocated across multiple users. In such environments, the attack can increase load on

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

shared infrastructure, potentially slowing responses for benign users. The consequences can be especially concerning in applications where timely feedback and responsiveness are critical.

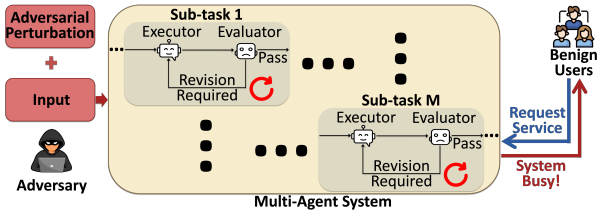

Figure 1: Illustration of the *saturated evaluation looping* attack scenario, where an attacker perturbs the system input to induce persistently saturated evaluation loops, prolonging the system runtime. For clarity, we explicitly highlight two sub-tasks that employ evaluation-looping schemes in the multi-agent system in the figure. In practice, multi-agent systems typically contain additional functional modules, which are omitted here for simplicity.

Recognizing this critical gap, this work aims to investigate the risks posed by the saturated evaluation looping attack scenario. In particular, from a red-teaming perspective, we aim to intentionally craft input perturbations that drive multi-agent systems into persistently saturated evaluation loops and significantly prolong their runtime, through which we can uncover the adversarial risks associated with this attack scenario and provide insights to inform future defense design. However, effectively crafting such input perturbations can be challenging. In practical settings, adversarial perturbations typically need to remain imperceptible (stealthy) to avoid being detected or filtered by the target multi-agent system, which limits their allowable perturbation magnitude. Meanwhile, multi-agent systems typically contain many evaluation loops that can be triggered across different sub-tasks, each potentially exhibiting distinct robustness characteristics. Under these constrained and heterogeneous dynamics, inducing persistently saturated evaluation loops in a multi-agent system to substantially prolong system runtime, while respecting stealth constraints, becomes non-trivial.

To address this problem, we propose a novel attack method tailored to the *saturated evaluation looping* attack scenario. The key insight is that, under a limited perturbation budget, distributing updates across many evaluation loops weakens the attack on each individual loop, whereas prioritizing selected loops makes the perturbation more effective at triggering the "revision-required" state. To realize this objective, we design two complementary mechanisms in our attack method, Noise-Robust Update Allocation (NRUA) and Non-Interference Mechanism (NIM), which regulate and coordinate perturbation allocation across evaluation loops. Empirically, even under stealth-constrained perturbation budgets, our tailored attack method can increase the frequency of revision triggering within evaluation-looping schemes by approximately 7.1× on average per system across multiple multi-agent systems spanning diverse tasks, resulting in an average 4.6× increase in overall system runtime relative to benign inputs. These findings highlight a practical safety concern: evaluation-looping schemes in multi-agent systems can be exploited to substantially prolong system runtime and increase resource consumption under bounded input perturbations.

Our contributions are: 1) We are the first to identify the practical, largely unexplored *saturated evaluation looping attack scenario*. 2) We propose a novel attack approach tailored to this scenario. 3) Extensive experiments show our method's efficacy in exposing this setting's adversarial risks.

2 Related Work

Multi-agent systems have attracted increasing attention and have been widely adopted across diverse real-world applications (Hong et al., 2023; Wu et al., 2024; Pang et al., 2025; Ye et al., 2025). Many existing multi-agent systems commonly employ evaluation-looping schemes to refine intermediate sub-task outputs, leveraging such schemes to enhance effectiveness across diverse tasks (Pang et al., 2025; Qian et al., 2024; Oueslati et al., 2025; Xu et al., 2025), including scientific figure generation (Zhu et al., 2026), paper-to-video generation (Liu et al., 2025a), affective image manipulation (Mao et al., 2025), and text-to-video generation (Zeng et al., 2025).

In parallel, from a red-teaming perspective, a growing literature (He et al., 2025; Triedman et al., 2025; Zheng et al., 2025a; Wang et al., 2025a; Yu et al., 2025; Shahroz et al., 2025; Tian et al., 2023; Zhang et al., 2025b; Mo et al., 2025) has examined vulnerabilities in multi-agent systems along several angles, including manipulating internal communications to degrade task accuracy (He et al., 2025), interfering with tool invocation to subvert decision workflows (Mo et al., 2025), and injecting semantic disturbances into agent inputs to induce erroneous outputs (Zheng et al., 2025a).

Different from previous works, this work identifies, for the first time, the *saturated evaluation looping* attack scenario, a practical yet previously unexplored setting targeting the widely adopted evaluation-looping schemes. We further propose a novel attack method tailored to this scenario.

Runtime-Prolonging Adversarial Attacks. Runtime-prolonging adversarial attacks aim to deliberately increase model runtime during inference and have been studied across a range of model architectures (Liu et al., 2025b; Kumar et al., 2025; Li et al., 2025; Dong et al., 2025; Zhang et al., 2025a; Zhou et al., 2026; 2025; He et al., 2025), including dynamic-width neural networks (Pan et al., 2023), early-exit networks (Hong et al., 2020), and spiking neural networks (Krithivasan et al., 2022). In contrast to these prior efforts, our work targets the distinctive evaluation-looping schemes in multi-agent systems and develops a novel attack method specifically designed to induce saturated evaluation loops, thereby prolonging system runtime in multi-agent systems.

3 Problem Formulation

To study the *saturated evaluation looping* attack scenario, we formalize a gray-box red-teaming threat model from three perspectives: (i) the adversary's knowledge and access, (ii) permissible perturbations, and (iii) the attack objective.

Adversary's knowledge and access. Following common threat models adopted in prior works on attacking multi-agent systems (Wang et al., 2025a; Yu et al., 2025; Shahroz et al., 2025; Tian et al., 2023), we consider a gray-box red-teaming adversary who has access to system-level information about the target multi-agent system, including the workflow structures and agent configurations and loop-level evaluator decisions during attack training.

Perturbation type and stealthiness constraints. Following prior work on learning adversarial perturbations via generative models (Xiao et al., 2018; Bai et al., 2021; Poursaeed et al., 2018; Garg & Ramakrishnan, 2020; Wang et al., 2020), we learn a perturbation generator conditioned on each input. Furthermore, to keep the attack realistic and stealthy, following prior adversarial attack settings (Pan et al., 2023; Krithivasan et al., 2020; Hong et al., 2020), we impose a stealthiness constraint, keeping perturbed inputs similar to benign user inputs.

Attack objective. In the *saturated evaluation looping* attack scenario, the adversary aims to learn a perturbation generator whose input perturbations induce persistently saturated evaluation loops. Such saturation prolongs system runtime, so that even a small number of interactions accumulates into substantial system-level resource consumption.

4 Method

To assess the risks of the *saturated evaluation looping* attack scenario from a red-teaming perspective, motivated by the success of reinforcement learning in prior red-teaming studies under restricted-access settings where internal architectures and parameters of agents are unavailable (Jia et al., 2026; Beutel et al., 2025), we first construct a reinforcement-learning-based baseline adversary in Sec. 4.1. This baseline adversary serves as an initial instantiation to induce saturated evaluation looping behaviours. However, as detailed in Sec. 4.2, it may not fully capture certain characteristics of this specific attack scenario and may therefore still underestimate its true risks. To address this limitation, we further develop a novel attack method tailored to this attack scenario (Sec. 4.2).

4.1 Baseline Adversary

Inspired by the effectiveness of reinforcement learning (RL) in prior adversarial attack settings (Shao et al., 2024; Jia et al., 2026; Beutel et al., 2025), we adopt an RL-based framework to construct a baseline adversary that learns a perturbation generator inducing persistently saturated evaluation loops, thereby prolonging system runtime. To achieve this, we first observe that many components of RL are task-agnostic and can thus be directly applied in our attack scenario, such as the mechanism for deriving gradient directions from reward signals. Hence, adapting these RL processes to our setting mainly requires specifying two task-dependent components: (1) a perturbation magnitude control mechanism consistent with the stealthiness constraint in Sec. 3, and (2) a reward function

aligned with inducing persistently saturated evaluation loops to prolong system runtime. Below, we present mechanically plausible and straightforward instantiations of these components.

Perturbation magnitude control. Let x denote a benign input to the target multi-agent system. Following the attack objective in Sec. 3, the adversary learns a perturbation generator G_θ, parameterized by θ, that maps the input x to a perturbed input $G_\theta(x)$. We optimize θ by RL using scalar rewards from the multi-agent system.

In our threat model, to maintain perceptual similarity between x and $G_\theta(x)$, their discrepancy needs to satisfy a stealthiness constraint. Accordingly, we constrain this discrepancy, denoted by δ, to lie within an admissible set $\mathcal{C}(\epsilon)$, where $\epsilon > 0$ specifies the maximum allowable perturbation magnitude and is set to a small value. During optimization, this constraint is enforced via standard projection-based methods (Goodfellow et al., 2014; Madry et al., 2018) (details in the Appendix):

$$\delta \leftarrow \Pi_{\mathcal{C}(\epsilon)}(\delta). \tag{1}$$

where $\Pi_{\mathcal{C}(\epsilon)}$ denotes the projection operator onto the admissible set $\mathcal{C}(\epsilon)$. Following common adversarial perturbation formulations (Madry et al., 2018; Addepalli et al., 2022), we instantiate $\mathcal{C}(\epsilon)$ per input modality. For image inputs, $\mathcal{C}(\epsilon) = \{\delta : \|\delta\|_\infty \leq \epsilon\}$; other modalities are specified in the Appendix.

Reward design. We next define the reward function so that it directly reflects our attack objective: maximizing the total system runtime induced by evaluation looping in the target multi-agent system.

Consider a multi-agent system in which M sub-tasks employ evaluation-looping schemes. For the m-th such sub-task, let N_m denote the pre-defined maximum number of evaluation loops that this sub-task is allowed to execute. Under this structure, the maximum number of evaluation loops that can be triggered in the system is $\sum_{m=1}^{M} N_m$. The total system runtime induced by evaluation looping can therefore be decomposed into the time contributions of these loops. Specifically, for each sub-task $m \in \{1, \ldots, M\}$, let t_m denote the observed wall-clock time incurred by a single evaluation loop of this sub-task. Then, for the n_m-th loop of sub-task m, where $n_m \in \{1, \ldots, N_m\}$, we define a weighted binary sub-reward $r^{wb}_{m,n_m} = t_m \cdot \mathbb{1}_{m,n_m}$, where $\mathbb{1}_{m,n_m} \in \{0, 1\}$ indicates whether the evaluator outputs "revision-required" in that loop. Specifically, $\mathbb{1}_{m,n_m} = 1$ if "revision-required" is triggered, and $\mathbb{1}_{m,n_m} = 0$ otherwise. Since only the decision "revision-required" triggers another evaluation loop, this binary indicator directly captures whether additional runtime is incurred. The overall reward can then be formulated by aggregating the loop-level time contributions as:

$$R(G_\theta(x)) = \sum_{m=1}^{M} \sum_{n_m=1}^{N_m} r^{wb}_{m,n_m} = \sum_{m=1}^{M} \sum_{n_m=1}^{N_m} t_m \cdot \mathbb{1}_{m,n_m}. \tag{2}$$

Eq. 2 defines the reward obtained from a single evaluation of the perturbed input. However, since modern multi-agent systems built on large foundation models are typically stochastic, evaluating the same input multiple times can provide a more reliable measurement of the reward. To allow the baseline adversary to leverage this advantage, we evaluate each perturbed input multiple times. Specifically, let K denote the number of evaluations performed for the same perturbed input. For the k-th evaluation, let $\mathbb{1}_{k,m,n_m} \in \{0, 1\}$ indicate whether the evaluator outputs "revision-required" in the n_m-th loop of sub-task m. We then define the averaged indicator $s_{m,n_m} = \frac{1}{K} \sum_{k=1}^{K} \mathbb{1}_{k,m,n_m} \in [0, 1]$, which measures how consistently this loop triggers "revision-required" across repeated runs. In particular, $s_{m,n_m} = 1$ means this loop *stably* triggers "revision-required" under stochastic executions, while $s_{m,n_m} < 1$ indicates that this loop still cannot stably reach the "revision-required" state. Accordingly, the averaged sub-reward across the K evaluations can be defined as $r_{m,n_m} = t_m \cdot s_{m,n_m}$ in a similar manner as r^{wb}_{m,n_m}. Using this averaged sub-reward, Eq. 2 can be updated and the reward function becomes:

$$R(G_\theta(x)) = \sum_{m=1}^{M} \sum_{n_m=1}^{N_m} r_{m,n_m}. \tag{3}$$

Compared with Eq. 2, this formulation keeps the reward directly aligned with the attack objective while making the reward more robust to stochasticity in the execution of the multi-agent system.

Having specified the perturbation magnitude control mechanism and the reward function, in the baseline adversary, we can then apply RL-based training to optimize the perturbation generator G_θ

by maximizing the expected reward over inputs sampled from the training distribution $\mathcal{D}$:

$$\max_{\theta} \ \mathbb{E}_{x\sim\mathcal{D}}\big[\, R(G_\theta(x))\,\big], \quad \text{s.t.} \quad \delta \in \mathcal{C}(\epsilon). \tag{4}$$

Under this objective, the baseline adversary aims to learn a perturbation generator that induces saturated evaluation looping behaviour and prolonged system runtime while satisfying the prescribed perturbation magnitude constraint. This formulation establishes a principled baseline for studying *saturated evaluation looping* attacks within the practical constraints of our threat model.

4.2 Risk Exposure Under Skilled Adversaries

Above, we construct a baseline adversary with an RL-based optimization pipeline, a straightforward and mechanically plausible instantiation for conducting saturated evaluation looping attacks.

Problem remains in the baseline adversary. Yet, despite its plausibility, we find that the baseline adversary achieves only limited attack effectiveness in our scenario (Baseline adversary in Tabs. 1–4), suggesting that directly transferring a generic adversarial optimization pipeline may be insufficient to fully exploit the characteristics of the saturated evaluation looping attack scenario. In practice, a skilled adversary may adopt attack strategies tailored to the specific attack objective rather than rely on generic designs. Motivated by this, we simulate such skilled adversaries by designing a tailored attack method. To guide our design, we first analyse the baseline introduced in Sec. 4.1 and examine how its optimization objective aligns with the true attack objective in this scenario.

Our analysis reveals a key mismatch between these two objectives. In the baseline formulation (Eq. 3), the objective maximizes the aggregated reward $\sum_{m=1}^{M}\sum_{n_m=1}^{N_m} r_{m,n_m}$, jointly optimizing all evaluation loops without explicitly prioritizing particular ones. However, the true attack objective in our scenario is inherently *decision-driven* and *budget-constrained*. Under a strict perturbation magnitude constraint, the goal is not to slightly push every evaluation loop forward, but to actually change the evaluator's decision from "pass" to "revision-required", thereby prolonging the system runtime. Consequently, distributing optimization effort across many loops may be suboptimal: multiple evaluation loops may move closer to triggering "revision-required", yet only a few actually enter that state, resulting in limited runtime amplification. A more effective strategy is to allocate the limited perturbation budget in a *prioritized* manner, concentrating effort on selected evaluation loops to more reliably induce prolonged system runtime.

Overview of our tailored attack method. Motivated by the advantage of prioritized optimization in our attack scenario, we develop a tailored attack method that explicitly leverages such prioritization to achieve more effective saturated evaluation looping attacks. To guide the design of our method, we first notice that designing effective prioritization in our scenario involves two complementary aspects. First, among evaluation loops that have not yet stably entered the "revision-required" state, updates should be allocated *with priority* rather than uniformly. Second, prioritized updates need to be performed *without interfering* with evaluation loops that have already stably entered the "revision-required" state; otherwise, subsequent updates may flip previously triggered loops from the "revision-required" state back into the "pass" state, erasing earlier progress. To satisfy these requirements, respectively, we design two components in our method, **Noise-Robust Update Allocation** and **Non-Interference Mechanism**. We detail each component below.

Noise-Robust Update Allocation (NRUA). We first focus on evaluation loops that have not yet stably entered the "revision-required" state. To enable prioritized updates among these loops, the core question is how to allocate update emphasis across them. To answer this question, we first observe that we possess a useful structural prior about emphasis allocation. Specifically, within each sub-task m, earlier loops with smaller n_m are more important than later ones: an earlier loop has to be triggered before later loops can be reached and thus contribute to prolonging the system runtime. However, this prior alone is insufficient to uniquely determine an effective emphasis allocation. On the one hand, across evaluation loops from different sub-tasks, it remains unclear how their relative emphases should be weighted. On the other hand, even within the same sub-task, although earlier loops should generally receive more emphasis than later ones, we still lack a principled way to determine how much emphasis should be assigned to each loop.

To handle this problem, we find that the loop-level sub-reward r_{m,n_m} appears to be a natural priority indicator. First, as analysed in the Appendix, the definition of r_{m,n_m} in Sec. 4.1 ensures that within any given sub-task, r_{m,n_m} respects the above structural prior: the sub-reward of earlier loops is

guaranteed to be no smaller than that of later ones (i.e., $r_{m,n_m^1} \geq r_{m,n_m^2}$ whenever $n_m^1 < n_m^2$). Second, both across sub-tasks and within each sub-task, a larger r_{m,n_m} corresponds to a larger potential contribution to the overall system runtime and thus appears preferable from an attack-objective perspective. These considerations suggest a promising approach: directly using r_{m,n_m} as the priority indicator to guide update allocation. However, directly relying on r_{m,n_m} is suboptimal, since this indicator is derived from intermediate training signals and may not reflect the reward profile under an "ideally" optimized perturbation generator. Training the generator against a complex multi-agent system induces a highly non-convex and irregular optimization landscape. Over long-horizon training, the reward observed under the current generator can therefore differ noticeably from the reward profile under ideal optimization. In this sense, intermediate rewards act only as *noisy indicators* of the true update priority if one seeks to achieve ideal optimization of the perturbation generator. Relying exclusively on such noisy signals may lead to suboptimal emphasis allocation, steering the generator toward suboptimal convergence and limiting attack effectiveness.

To address this issue, we first note that the sub-reward values under an "ideally" optimized perturbation generator are inherently unavailable during training. In light of this, we therefore draw inspiration from certain classical optimization settings where learning have to proceed from noisy, instantaneous measurements of the true objective, for which a common remedy is to structure learning as a gradual transition from *exploration* to *exploitation* (Van Laarhoven & Aarts, 1987; Shi & Eberhart, 1999; Perera et al., 2024; Shi & Eberhart, 1998; Bengio et al., 2015). Specifically, in these settings, *the key principle is to maximize exploitation of the noisy measurement while maintaining at least a non-zero level of exploration that is progressively annealed over time*. In our setting, *exploitation* treats r_{m,n_m} as a fully reliable priority indicator and concentrates updates on loops with larger r_{m,n_m}, whereas *exploration* ensures that every loop that has not yet stably entered the "revision-required" state retains some chance of being updated, even when its current reward is small. This exploration–exploitation structure enables effective prioritization while accounting for indicator noise, naturally motivating us to schedule updates over evaluation loops that have not yet stably entered the "revision-required" state according to this principle.

Building on the exploration–exploitation principle, we next aim to translate it into an analytical formulation that is operational in our setting. However, turning this conceptual guideline into a concrete formulation is non-trivial. To proceed, we decompose the objective into two coupled sub-goals: (1) *maximizing exploitation* and (2) *maintaining at least a non-zero level of exploration that is progressively annealed over time*. We then integrate these sub-goals into a unified formulation.

Sub-goal 1: maximizing exploitation. At iteration i, let $\mathcal{U}_i = \{(m, n_m) \mid s_{m,n_m} < 1\}$ denote the set of evaluation loops that have not yet stably entered the "revision-required" state. We introduce a nonnegative weight vector $\boldsymbol{\alpha}_i = \{\alpha_{i,m,n_m}\}_{(m,n_m)\in\mathcal{U}_i}$, constrained to sum to 1, which specifies the relative update emphasis over loops in $\mathcal{U}_i$ at iteration i. Under the exploitation objective, we aim to allocate as much emphasis as possible to higher-priority loops, as indicated by larger intermediate sub-rewards r_{m,n_m}. As shown in the Appendix, this objective is equivalently formulated as selecting $\boldsymbol{\alpha}_i$ to maximize the weighted aggregate of sub-rewards:

$$\begin{aligned} \boldsymbol{\alpha}_i^\star &\leftarrow \arg\max_{\boldsymbol{\alpha}_i \in \Delta(\mathcal{U}_i)} \sum_{(m,n_m)\in\mathcal{U}_i} \alpha_{i,m,n_m}\, r_{m,n_m}, \\ \text{where} \quad \Delta(\mathcal{U}_i) &= \Big\{\boldsymbol{\alpha}_i : \alpha_{i,m,n_m} \geq 0, \sum_{(m,n_m)\in\mathcal{U}_i} \alpha_{i,m,n_m} = 1\Big\}. \end{aligned} \tag{5}$$

Eq. 5 concentrates updates on the most promising loop in $\mathcal{U}_i$, thereby maximizing exploitation.

Sub-goal 2: maintaining at least a non-zero level of exploration that is progressively annealed. Exploration, on the other hand, means that every loop in $\mathcal{U}_i$ should retain some opportunity to receive updates, even if it currently appears less promising under the noisy priority indicator r_{m,n_m}. Intuitively, the level of exploration can be represented by the *relative spread* of the update weights $\boldsymbol{\alpha}_i$ across these loops: highly concentrated weights imply weak exploration, whereas more evenly distributed weights imply stronger exploration. Following this intuition, a natural way to quantify the level of exploration can be then made using normalized entropy (Pielou, 1966), denoted by $H_n(\boldsymbol{\alpha}_i) = \frac{-\sum_{(m,n_m)\in\mathcal{U}_i} \alpha_{i,m,n_m} \log \alpha_{i,m,n_m}}{\log |\mathcal{U}_i|}$, which lies in $[0, 1]$ and measures the relative dispersion of update allocation with respect to its maximum possible spread. By leveraging normalized

entropy, we can then enforce at least a prescribed level of exploration by requiring:

$$H_n(\boldsymbol{\alpha}_i) \geq \lambda_i, \qquad \lambda_i = \frac{T-i}{T}, \tag{6}$$

where T denotes the total number of training iterations. Here, the coefficient λ_i acts as an annealing schedule that decreases over time, thereby progressively relaxing the minimum exploration requirement. In this way, Eq. 6 directly operationalizes the objective of Sub-goal 2.

Combining the two sub-goals yields the normalized-entropy-constrained exploitation objective:

$$\boldsymbol{\alpha}_i^\star \leftarrow \arg\max_{\boldsymbol{\alpha}_i \in \Delta(\mathcal{U}_i)} \sum_{(m,n_m)\in\mathcal{U}_i} \alpha_{i,m,n_m}\, r_{m,n_m} \quad \text{s.t.} \quad H_n(\boldsymbol{\alpha}_i) \geq \lambda_i. \tag{7}$$

Thus, achieving the exploration-exploitation behaviour reduces to solving Eq. 7, which, despite its apparently intricate form, can be implemented efficiently in our training pipeline as shown below.

Proposition 1 *(Proof in the Appendix.) The optimization problem in Eq. 7 admits an optimal allocation characterized by the Boltzmann family, with components for $(m, n_m) \in \mathcal{U}_i$ given by:*

$$\alpha^\star_{i,m,n_m} = \frac{\exp\big(r_{m,n_m}/\beta_i\big)}{\sum_{(m',n'_{m'})\in\mathcal{U}_i} \exp\big(r_{m',n'_{m'}}/\beta_i\big)}. \tag{8}$$

Here, β_i controls the concentration of the update weights. Its determination follows from the entropy-matching characterization in Appendix Sec. J, which covers the full entropy range.

Having obtained $\boldsymbol{\alpha}_i^\star$, we can then use it to formulate the reward function as:

$$R(G_\theta(x)) = \sum_{(m,n_m)\in\mathcal{U}_i} \alpha^\star_{i,m,n_m}\, r_{m,n_m}. \tag{9}$$

This formulation enables us to update loops that have not yet stably entered the "revision-required" state in a prioritized manner, while accounting for the noise in intermediate sub-rewards when they are used as priority indicators. Notably, as analysed in the Appendix, the reward function design in Eq. 9 also preserves the structural prior within each sub-task: earlier loops with smaller n_m are guaranteed to receive no less emphasis than later ones, i.e., $\alpha_{m,n_m^1} \geq \alpha_{m,n_m^2}$ whenever $n_m^1 < n_m^2$.

Non-Interference Mechanism (NIM). Above, we design NRUA to prioritize updates over evaluation loops that have not yet stably entered the "revision-required" state. We now address the complementary question: how to ensure that such updates do not interfere with evaluation loops that have *already* stably entered the "revision-required" state, i.e., loops in $\mathcal{S}_i = \{(m, n_m) \mid s_{m,n_m} = 1\}$.

To achieve this, a natural attempt, following the baseline design in Sec. 4.1, is to aggregate the objectives of loops in $\mathcal{U}_i$ and $\mathcal{S}_i$ into a single reward with appropriate weighting. Yet, such multi-objective aggregation offers no structural guarantee of non-interference: the resulting gradient may conflict with the sensitive directions of loops in $\mathcal{S}_i$ and push them back into the "pass" state. In other words, joint weighting cannot *ensure* protection of the "revision-required" state of loops in $\mathcal{S}_i$.

To overcome this limitation, we draw on the concept of *null space* (Wang et al., 2021; Horn & Johnson, 2012; Gantmakher, 2000) from linear algebra, which provides a natural way to protect previously stabilized loops from conflicting updates. Specifically, let G_i denote the matrix formed by stacking the gradient directions of loops in $\mathcal{S}_i$ (specified below). The null space of G_i consists of all vectors v satisfying

$$G_i\, v = 0. \tag{10}$$

Update directions in this null space are therefore orthogonal to the gradient directions associated with loops in $\mathcal{S}_i$, thereby removing update components aligned with these protected directions.

Motivated by this property, rather than aggregating the objectives of loops in $\mathcal{S}_i$ into the reward function, we adopt a decoupled two-stage strategy. In the first stage, the reward focuses on loops in $\mathcal{U}_i$ via NRUA, and the RL training pipeline produces a gradient direction g_i based on the reward function measured in Eq. 9. In the second stage, we therefore project g_i onto the null space of G_i to obtain the adjusted direction g_i^{proj}. Concretely, at each training iteration i, after computing the NRUA-based reward using Eq. 9 and deriving the gradient direction g_i, we collect the RL-based gradient directions corresponding to loops in $\mathcal{S}_i$, each estimated using an independently sampled auxiliary perturbation, into the matrix $G_i = \big[g_{m,n_m}^\top\big]_{(m,n_m)\in\mathcal{S}_i}$, with one protected gradient per row. We then project g_i onto the null space of G_i. Letting the columns of V_0 form an orthonormal

basis of this null space, the projected direction is:

$$g_i^{\text{proj}} = V_0 V_0^\top g_i. \tag{11}$$

In implementation, we apply the equivalent projector through a protected-loop Gram matrix without explicitly constructing V_0 (see Appendix Sec. L). The projected direction g_i^{proj} is then used as the actual update direction. In this way, each update concentrates on advancing loops in $\mathcal{U}_i$ while reducing interference with the optimization progress already made on loops in $\mathcal{S}_i$.

Overall, our attack method designs two complementary mechanisms: NRUA, which prioritizes optimization over evaluation loops that have not yet stably entered the "revision-required" state, and NIM, which structurally protects those that have already entered it. Together, they yield a more problem-tailored adversary that more comprehensively exposes the risks of this attack scenario.

4.3 Overall Training and Testing

Training. We illustrate our training pipeline in Fig. 2. At each iteration i, we sample benign inputs $x \sim \mathcal{D}$ and apply the perturbation generator G_θ to obtain the perturbed inputs $G_\theta(x)$. We then execute the target multi-agent system with $G_\theta(x)$ and obtain sub-rewards $\{\{r_{m,n_m}\}_{n_m=1}^{N_m}\}_{m=1}^{M}$ for all evaluation loops. Based on these sub-rewards and the annealing coefficient λ_i, we solve Eq. 8 to obtain the update-weight vector $\boldsymbol{\alpha}_i^\star$ over evaluation loops in the set $\mathcal{U}_i$. Using $\boldsymbol{\alpha}_i^\star$, we compute the reward $R(\cdot)$ through Eq. 9. This reward is then fed into the RL-based training pipeline detailed in Appendix Sec. B to obtain a gradient direction g_i. We next obtain g_i^{proj} from g_i through Eq. 11. The generator parameters θ are finally updated along g_i^{proj}.

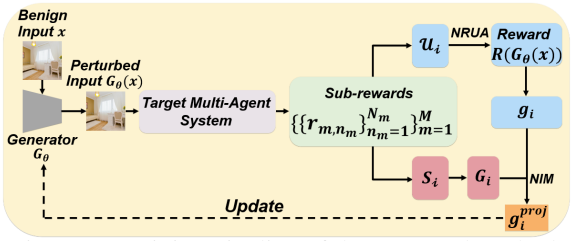

Figure 2: Training pipeline of the proposed method.

Testing. At test time, given a benign input x to the target multi-agent system, we construct the perturbed input $G_\theta(x)$ using the learned perturbation generator G_θ. The perturbed input is then directly fed into the target system to simulate the saturated evaluation looping attack. No loop-level feedback is required at test time.

5 Experiments

Victim Multi-Agent Systems. We evaluate our attack on four multi-agent systems spanning different tasks: AutoFigure (Zhu et al., 2026), EmoAgent (Mao et al., 2025), Preacher (Liu et al., 2025a), and CoAgent (Zeng et al., 2025). For AutoFigure, we follow (Zhu et al., 2026) and evaluate on the FigureBench dataset (Zhu et al., 2026) using its official train/test split. For EmoAgent, we follow (Mao et al., 2025) and use the EmoEditSet dataset (Yang et al., 2025) with its official train/test split. For Preacher, we evaluate on its official benchmark. As this benchmark does not provide a pre-defined train/test split, we randomly partition the data into training and testing sets at a 3:1 ratio. For CoAgent, we follow (Zeng et al., 2025) and evaluate on the VBench dataset (Zheng et al., 2025b). Since VBench also does not pre-define a train/test split, we likewise randomly partition it into training and testing sets at a 3:1 ratio. More details are in the Appendix.

Evaluation Metrics. To quantify the induced loop saturation and its resulting runtime and resource costs, we report three metrics: We first report the total number of executed loops (abbrev. total loops) as a direct measure of loop saturation. We further report the total system runtime (abbrev. total runtime) and the total number of tokens consumed (abbrev. total tokens) to quantify runtime overhead and resource usage. All results are presented as the mean with standard deviation over five runs. More details on these metrics are in the Appendix.

Compared Methods. In our experiments, we compare against three categories of methods: (1) Direct inference, where the multi-agent system is executed directly on the benign input. (2) Existing runtime-prolonging adversarial attacks that can be adapted with only minor implementation-level modifications to our saturated evaluation looping scenario, including Sponge (Shumailov et al., 2021), GradMDM (Pan et al., 2023), and SlothDoor (Guan et al., 2025) (adaptation details are in the Appendix). (3) Besides the above, we also assess the Baseline adversary introduced in Sec. 4.1.

Table 1: Experimental results on the EmoAgent multi-agent system.

Methods	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
Direct inference	4.0 ± 1.1	104.9 ± 17.2	2579.0 ± 587.4
Sponge (Shumailov et al., 2021)	9.7 ± 2.1	186.2 ± 36.8	4699.9 ± 954.7
GradMDM (Pan et al., 2023)	10.2 ± 1.9	190.7 ± 39.7	4794.7 ± 945.9
SlothDoor (Guan et al., 2025)	9.4 ± 1.8	181.2 ± 34.6	4423.9 ± 848.2
Baseline adversary	9.3 ± 1.8	179.6 ± 35.2	4358.8 ± 859.6
Ours	**38.7** ± 2.2	**400.8** ± 41.5	**7182.1** ± 998.3

Table 2: Experimental results on the AutoFigure multi-agent system.

Methods	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
Direct inference	1.4 ± 0.1	101.8 ± 3.8	34292.2 ± 1943.7
Sponge (Shumailov et al., 2021)	2.2 ± 0.1	123.4 ± 4.6	73393.2 ± 3089.3
GradMDM (Pan et al., 2023)	2.5 ± 0.2	133.7 ± 6.4	90324.8 ± 4023.4
SlothDoor (Guan et al., 2025)	2.1 ± 0.2	119.6 ± 5.0	69413.4 ± 3314.2
Baseline adversary	1.9 ± 0.1	114.3 ± 3.4	57357.6 ± 2253.5
Ours	**4.8** ± 0.1	**206.6** ± 5.8	**209747.5** ± 3871.6

Table 3: Experimental results on the Preacher multi-agent system.

Methods	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
Direct inference	12.1 ± 2.1	638.2 ± 120.3	20796.1 ± 3749.3
Sponge (Shumailov et al., 2021)	28.9 ± 4.4	1162.4 ± 152.6	34692.8 ± 5704.5
GradMDM (Pan et al., 2023)	30.5 ± 4.7	1211.4 ± 194.3	34888.3 ± 5367.4
SlothDoor (Guan et al., 2025)	27.4 ± 4.2	1159.8 ± 201.5	34237.6 ± 4832.4
Baseline adversary	26.1 ± 3.1	1148.2 ± 168.3	34099.7 ± 4701.9
Ours	**84.2** ± 5.1	**3045.3** ± 213.7	**164487.2** ± 6042.8

Table 4: Experimental results on the CoAgent multi-agent system.

Methods	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
Direct inference	4.1 ± 0.3	1295.1 ± 115.9	1540.2 ± 147.1
Sponge (Shumailov et al., 2021)	6.6 ± 0.7	2088.9 ± 176.2	2664.7 ± 215.4
GradMDM (Pan et al., 2023)	6.8 ± 0.7	2162.7 ± 179.6	2749.3 ± 224.3
SlothDoor (Guan et al., 2025)	6.6 ± 0.7	2077.8 ± 177.6	2645.7 ± 214.6
Baseline adversary	6.5 ± 0.7	2069.4 ± 167.4	2616.2 ± 201.4
Ours	**33.3** ± 0.8	**10264.2** ± 188.4	**12152.5** ± 236.9

Implementation Details. We set the maximum L_∞ perturbation budget $\epsilon = 8/255$ for image-modality inputs (Addepalli et al., 2022; Qin et al., 2019). For text-modality inputs, we set $\epsilon = 1/10$ under an L_2 constraint (Zhu et al., 2019). We set hyperparameter $K = 5$. We adopt REINFORCE (Williams, 1992) as the RL framework. More implementation details are in the Appendix.

5.1 Main Results

We report results in Tabs. 1–4. As shown, across victim multi-agent systems and evaluation metrics, our method consistently outperforms all compared methods: it substantially increases the number of executed evaluation loops, prolongs system runtime, and amplifies inference-time computational resource consumption. These results validate the effectiveness of our approach. We further evaluate our attack framework in Appendix Secs. F and G (including robustness across different system prompts, transferability across system LLM backbones and multi-agent systems, effectiveness under defenses, etc.), and we discuss these results and their implications in Appendix Sec. A.

5.2 Ablation Studies and Further Analyses

In this section and in the Appendix, we conduct extensive ablation studies and further analyses on the multi-agent system: EmoAgent (Mao et al., 2025).

Impact of the proposed mechanisms. To assess the contribution of each mechanism in our method, we test two ablated variants, **Ours (w/o NRUA)** and **Ours (w/o NIM)**. As shown in Tab. 5, our full framework (**Ours (full)**) consistently outperforms both variants across all metrics, showing the efficacy of each proposed mechanism. Moreover, both ablated variants outperform baseline adversary, further highlighting the benefit of incorporating either mechanism into our method.

Table 5: Evaluation of the proposed mechanisms.

Methods	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
Baseline adversary	9.3	179.6	4358.8
Ours (w/o NRUA)	15.8	247.6	5538.2
Ours (w/o NIM)	23.8	292.1	6171.2
Ours (full)	38.7	400.8	7182.1

Visualization of loop-triggering behaviour. In Fig. 3, we illustrate the loop-triggering behaviour. As shown, compared with the original benign input (Fig. 3(a)), feeding the multi-agent system with our generated perturbed input (Fig. 3(b)) triggers more evaluation loops in the multi-agent system, which further shows the effectiveness of our method. More visualizations are in Appendix.

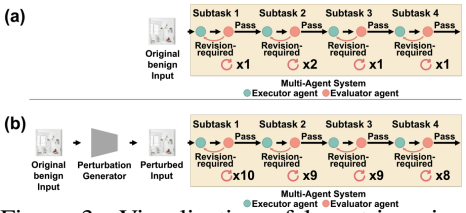

Figure 3: Visualization of loop-triggering behaviour on the EmoAgent system.

6 Conclusion

In this work, we identify a practical yet largely underexplored threat setting, the *saturated evaluation looping* attack scenario. To systematically uncover the adversarial risks inherent in this scenario, we develop a novel attack method specifically tailored to it. Extensive experiments demonstrate that our method effectively reveals the risks associated with saturated evaluation looping, providing new insights into the robustness of contemporary multi-agent systems.

AI USE STATEMENT

In this work, we used generative AI tools to polish the writing, i.e., grammar checking and wording refinements on text written by the authors. We did not use generative AI tools for research ideation, method design, or experiment design. All AI-assisted text was checked by the authors.

ETHICS STATEMENT

This work studies an attack that drives multi-agent systems into saturated evaluation loops. Our purpose is defensive: we expose this previously underexplored vulnerability so that it can be addressed before being exploited, and we accordingly evaluate candidate defenses in Appendix Sec. G and discuss monitoring and mitigation strategies in Appendix Sec. A. No online service of any third party is attacked or disrupted.

REPRODUCIBILITY STATEMENT

Our attack method is fully specified in Sec. 4, and the complete proof of Proposition 1 together with the derivations of the objectives used in the main paper are given in Appendix Secs. H, I, D, E, and J. The victim multi-agent systems, the datasets, and the train/test splits are described in Sec. 5. All remaining implementation details, including the perturbation generator architectures, training steps, perturbation budgets, hardware, training cost, and the definitions of the evaluation metrics, are provided in Appendix Secs. B and C. We will release the code.

REFERENCES

Sravanti Addepalli, Samyak Jain, Gaurang Sriramanan, and R Venkatesh Babu. Scaling adversarial training to large perturbation bounds. In *European Conference on Computer Vision*, pp. 301–316. Springer, 2022.

Anthropic. Claude 3.5 sonnet. `https://www.anthropic.com/news/claude-3-5-sonnet`, 2024.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Tao Bai, Jun Zhao, Jinlin Zhu, Shoudong Han, Jiefeng Chen, Bo Li, and Alex Kot. Ai-gan: Attack-inspired generation of adversarial examples. In *2021 IEEE International Conference on Image Processing (ICIP)*, pp. 2543–2547. IEEE, 2021.

Esma Balkır, Alice Pernthaller, Marco Basaldella, José Hernández-Orallo, and Nigel Collier. Confident rankings with fewer items: Adaptive llm evaluation with continuous scores. *arXiv preprint arXiv:2601.13885*, 2026.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28, 2015.

Alex Beutel, Kai Yuanqing Xiao, Johannes Heidecke, and Lilian Weng. Diverse and effective red teaming with auto-generated rewards and multi-step reinforcement learning. In *Red Teaming GenAI: What Can We Learn from Adversaries?*, 2025. URL `https://openreview.net/forum?id=wYUM9FnQJs`.

Xiaohe Bo, Zeyu Zhang, Quanyu Dai, Xueyang Feng, Lei Wang, Rui Li, Xu Chen, and Ji-Rong Wen. Reflective multi-agent collaboration based on large language models. *Advances in Neural Information Processing Systems*, 37:138595–138631, 2024.

Xiuyuan Chen, Jian Zhao, Yuchen Yuan, Tianle Zhang, Huilin Zhou, Zheng Zhu, Ping Hu, Linghe Kong, Chi Zhang, Weiran Huang, et al. Radar: A risk-aware dynamic multi-agent framework for llm safety evaluation via role-specialized collaboration. *arXiv preprint arXiv:2509.25271*, 2025.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Coze. Coze Loop: Next-generation AI agent optimization platform. https://github.com/coze-dev/coze-loop, 2026. Accessed August 2026.

CrewAI, Inc. Crewai: Framework for orchestrating role-playing, autonomous ai agents. https://github.com/crewAIInc/crewAI, 2026. Documentation: https://docs.crewai.com/en/concepts/tasks. Accessed August 2026.

Jianshuo Dong, Ziyuan Zhang, Qingjie Zhang, Tianwei Zhang, Hao Wang, Hewu Li, Qi Li, Chao Zhang, Ke Xu, and Han Qiu. An engorgio prompt makes large language model babble on. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=m4eXBo0VNc.

Zhihao Du, Yuxuan Wang, Qian Chen, Xian Shi, Xiang Lv, Tianyu Zhao, Zhifu Gao, Yexin Yang, Changfeng Gao, Hui Wang, et al. Cosyvoice 2: Scalable streaming speech synthesis with large language models. *arXiv preprint arXiv:2412.10117*, 2024.

Feliks Ruvimovich Gantmakher. *The theory of matrices*, volume 131. American Mathematical Soc., 2000.

Siddhant Garg and Goutham Ramakrishnan. Bae: Bert-based adversarial examples for text classification. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 6174–6181, 2020.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

GPT Researcher Contributors. GPT researcher: An autonomous agent that conducts deep research on any data using any LLM providers. https://github.com/assafelovic/gpt-researcher, 2026. Multi-agent documentation: https://docs.gptr.dev/docs/gpt-researcher/multi_agents/langgraph. Accessed August 2026.

Zhi-Qian Guan, Jing Liu, Zhenli He, Shengfa Miao, and Di Liu. Input-aware slowdown backdoor attack on dynamic neural networks. *Journal of Circuits, Systems and Computers*, 34(16):2550283, 2025.

Pengfei He, Yuping Lin, Shen Dong, Han Xu, Yue Xing, and Hui Liu. Red-teaming llm multi-agent systems via communication attacks. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6726–6747, 2025.

Sanghyun Hong, Yiğitcan Kaya, Ionuţ-Vlad Modoranu, and Tudor Dumitraş. A panda? no, it's a sloth: Slowdown attacks on adaptive multi-exit neural network inference. *arXiv preprint arXiv:2010.02432*, 2020.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*, 2023.

Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Yuqi Jia, Hangsheng Zhang, Chenxi Liu, Junjie Liang, and Xun Chen. Black-box red-teaming of multi-agent systems via reinforcement learning, 2026. URL https://openreview.net/forum?id=jd6sUOwz3H.

Sarada Krithivasan, Sanchari Sen, and Anand Raghunathan. Sparsity turns adversarial: Energy and latency attacks on deep neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):4129–4141, 2020.

Sarada Krithivasan, Sanchari Sen, Nitin Rathi, Kaushik Roy, and Anand Raghunathan. Efficiency attacks on spiking neural networks. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 373–378, 2022.

Abhinav Kumar, Jaechul Roh, Ali Naseh, Marzena Karpinska, Mohit Iyyer, Amir Houmansadr, and Eugene Bagdasarian. Overthink: Slowdown attacks on reasoning llms. *arXiv preprint arXiv:2502.02542*, 2025.

Johann Heinrich Lambert. *Photometria sive de mensura et gradibus luminis, colorum et umbrae*. sumptibus vidvae E. Klett, typis CP Detleffsen, 1760.

LangChain. LangGraph: Low-level orchestration framework for building stateful agents. `https://github.com/langchain-ai/langgraph`, 2026. Accessed August 2026.

LangGenius. Dify: Build agentic workflows and RAG pipelines with rich AI model and tool support. `https://github.com/langgenius/dify`, 2026. Accessed August 2026.

Xinyu Li, Tianjin Huang, Ronghui Mu, Xiaowei Huang, and Gaojie Jin. Pot: Inducing overthinking in llms via black-box iterative optimization. *arXiv preprint arXiv:2508.19277*, 2025.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Jingwei Liu, Ling Yang, Hao Luo, Fan Wang, Hongyan Li, and Mengdi Wang. Preacher: Paper-to-video agentic system. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17129–17139, 2025a.

Shuaitong Liu, Renjue Li, Lijia Yu, Lijun Zhang, Zhiming Liu, and Gaojie Jin. Badthink: Triggered overthinking attacks on chain-of-thought reasoning in large language models. *arXiv preprint arXiv:2511.10714*, 2025b.

Wendy Yaqiao Liu, Rui Jerry Huang, Anastasia Miin, and Lei Ding. Adaptive coopetition: Leveraging coarse verifier signals for resilient multi-agent llm reasoning. In *The 14th International Joint Conference on Natural Language Processing and The 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pp. 145–155, 2025c.

Zhipeng Liu, Xuefeng Bai, Kehai Chen, Xinyang Chen, Xiucheng Li, Yang Xiang, Jin Liu, Hong-Dong Li, Yaowei Wang, Liqiang Nie, et al. A survey on the feedback mechanism of llm-based ai agents. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pp. 10582–10592. International Joint Conferences on Artificial Intelligence, 2025d.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Yilong Lu, Chunhui Zhang, and Wei Wang. Systematic bias in large language models: Discrepant response patterns in binary vs. continuous judgment tasks. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 47, 2025.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJzIBfZAb`.

Qi Mao, Haobo Hu, Yujie He, Difei Gao, Haokun Chen, and Libiao Jin. Emoagent: A multi-agent framework for diverse affective image manipulation. *arXiv preprint arXiv:2503.11290*, 2025.

Microsoft AutoGen. Reflection: AutoGen core design patterns. `https://github.com/microsoft/autogen`, 2026. Accessed August 2026.

Kanghua Mo, Li Hu, Yucheng Long, and Zhihao li. Attractive metadata attack: Inducing LLM agents to invoke malicious tools. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=oLGtPYdRzU.

Khouloud Oueslati, Maxime Lamothe, and Foutse Khomh. Refagent: A multi-agent llm-based framework for automatic software refactoring. *arXiv preprint arXiv:2511.03153*, 2025.

Jianhong Pan, Lin Geng Foo, Qichen Zheng, Zhipeng Fan, Hossein Rahmani, Qiuhong Ke, and Jun Liu. Gradmdm: Adversarial attack on dynamic networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(9):11374–11381, 2023.

Wei Pang, Kevin Qinghong Lin, Xiangru Jian, Xi He, and Philip Torr. Paper2poster: Towards multimodal poster automation from scientific papers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=p0E74lpRBD.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

David Perera, Victor Letzelter, Théo Mariotte, Adrien Cortés, Mickael Chen, Slim Essid, and Gaël Richard. Annealed multiple choice learning: Overcoming limitations of winner-takes-all with annealing. *Advances in Neural Information Processing Systems*, 37:10541–10574, 2024.

Evelyn C Pielou. The measurement of diversity in different types of biological collections. *Journal of theoretical biology*, 13:131–144, 1966.

Omid Poursaeed, Isay Katsman, Bicheng Gao, and Serge Belongie. Generative adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4422–4431, 2018.

William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 15174–15186, 2024.

Chongli Qin, James Martens, Sven Gowal, Dilip Krishnan, Krishnamurthy Dvijotham, Alhussein Fawzi, Soham De, Robert Stanforth, and Pushmeet Kohli. Adversarial robustness through local linearization. *Advances in neural information processing systems*, 32, 2019.

Rana Shahroz, Zhen Tan, Sukwon Yun, Charles Fleming, and Tianlong Chen. Agents under siege: Breaking pragmatic multi-agent llm systems with optimized prompt attacks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9661–9674, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Y Shi and R Eberhart. A modified particle swarm optimizer. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*, pp. 69–73. IEEE, 1998.

Yuhui Shi and Russell C Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pp. 1945–1950. IEEE, 1999.

Ilia Shumailov, Yiren Zhao, Daniel Bates, Nicolas Papernot, Robert Mullins, and Ross Anderson. Sponge examples: Energy-latency attacks on neural networks. In *2021 IEEE European symposium on security and privacy (EuroS&P)*, pp. 212–231. IEEE, 2021.

Yu Tian, Xiao Yang, Jingyuan Zhang, Yinpeng Dong, and Hang Su. Evil geniuses: Delving into the safety of llm-based agents. *arXiv preprint arXiv:2311.11855*, 2023.

Harold Triedman, Rishi Dev Jha, and Vitaly Shmatikov. Multi-agent systems execute arbitrary malicious code. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=DAozI4etUp.

Jen tse Huang, Jiaxu Zhou, Tailin Jin, Xuhui Zhou, Zixi Chen, Wenxuan Wang, Youliang Yuan, Michael Lyu, and Maarten Sap. On the resilience of LLM-based multi-agent collaboration with faulty agents. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=bkiM54QftZ.

Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pp. 7–15. Springer, 1987.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Boxin Wang, Hengzhi Pei, Boyuan Pan, Qian Chen, Shuohang Wang, and Bo Li. T3: Tree-autoencoder constrained adversarial text generation for targeted attack. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6134–6150, 2020.

Shilong Wang, Guibin Zhang, Miao Yu, Guancheng Wan, Fanci Meng, Chongye Guo, Kun Wang, and Yang Wang. G-safeguard: A topology-guided security lens and treatment on llm-based multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7261–7276, 2025a.

Shipeng Wang, Xiaorong Li, Jian Sun, and Zongben Xu. Training networks in null space of feature covariance for continual learning. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pp. 184–193, 2021.

Victor Wang, Michael JQ Zhang, and Eunsol Choi. Improving llm-as-a-judge inference with the judgment distribution. *arXiv preprint arXiv:2503.03064*, 2025b.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Daniel Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025c. URL https://openreview.net/forum?id=OJd3ayDDoF.

Yingming Wang and Pepa Atanasova. Self-critique and refinement for faithful natural language explanations. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 8492–8518, 2025.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Chengxuan Xia, Qianye Wu, Sixuan Tian, and Yilun Hao. Parallelism meets adaptiveness: Scalable documents understanding in multi-agent llm systems. *arXiv preprint arXiv:2507.17061*, 2025.

Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 3905–3911, 2018.

Tianyang Xu, Dan Zhang, Kushan Mitra, and Estevam Hruschka. Verification-aware planning for multi-agent systems. *arXiv preprint arXiv:2510.17109*, 2025.

Jingyuan Yang, Jiawei Feng, Weibin Luo, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Emoedit: Evoking emotions through image manipulation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 24690–24699, 2025.

Rui Ye, Shuo Tang, Rui Ge, Yaxin Du, Zhenfei Yin, Siheng Chen, and Jing Shao. MAS-GPT: Training LLMs to build LLM-based multi-agent systems. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=3CiSpY3QdZ.

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

Qinglin Zeng, Kaitong Cai, Ruiqi Chen, Qinhan Lv, and Keze Wang. Coagent: Collaborative planning and consistency agent for coherent video generation. *arXiv preprint arXiv:2512.22536*, 2025.

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

Jiawei Zhang, Shuang Yang, and Bo Li. UDora: A unified red teaming framework against LLM agents by dynamically hijacking their own reasoning. In *Forty-second International Conference on Machine Learning*, 2025b. URL https://openreview.net/forum?id=pRmxQHgjb1.

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Arik. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237, 2024.

Can Zheng, Yuhan Cao, Xiaoning Dong, and Tianxing He. Demonstrations of integrity attacks in multi-agent systems. *arXiv preprint arXiv:2506.04572*, 2025a.

Dian Zheng, Ziqi Huang, Hongbo Liu, Kai Zou, Yinan He, Fan Zhang, Lulu Gu, Yuanhan Zhang, Jingwen He, Wei-Shi Zheng, et al. Vbench-2.0: Advancing video generation benchmark suite for intrinsic faithfulness. *arXiv preprint arXiv:2503.21755*, 2025b.

Kaiyu Zhou, Yongsen Zheng, Yicheng He, Meng Xue, Xueluan Gong, Yuji Wang, and Kwok-Yan Lam. Beyond max tokens: Stealthy resource amplification via tool calling chains in llm agents. *arXiv preprint arXiv:2601.10955*, 2026.

Zhenhong Zhou, Zherui Li, Jie Zhang, Yuanhe Zhang, Kun Wang, Yang Liu, and Qing Guo. Corba: Contagious recursive blocking attacks on multi-agent systems based on large language models. *arXiv preprint arXiv:2502.14529*, 2025.

Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. Freelb: Enhanced adversarial training for natural language understanding. *arXiv preprint arXiv:1909.11764*, 2019.

Minjun Zhu, Zhen Lin, Yixuan Weng, Panzhong Lu, Qiujie Xie, Yifan Wei, Sifan Liu, QiYao Sun, and Yue Zhang. Autofigure: Generating and refining publication-ready scientific illustrations. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=5N3z9JQJKq.

APPENDIX

A DISCUSSIONS

Evaluation looping has been very widely used in multi-agent systems (Hong et al., 2023; Wu et al., 2024; Mao et al., 2025; Liu et al., 2025a; Zeng et al., 2025; Zhu et al., 2026), and many of such practical systems have their code, internal workflow and agent configurations publicly released, and these systems have already been officially deployed online for public use (Wang et al., 2025c; CrewAI, Inc., 2026; Microsoft AutoGen, 2026; GPT Researcher Contributors, 2026; LangChain, 2026; LangGenius, 2026; Coze, 2026). Such widely used and publicly deployed systems naturally raise concerns about their robustness and security. Therefore, previous works have studied the vulnerability of such multi-agent systems from angles such as internal communications (He et al., 2025), tool invocation (Mo et al., 2025), and semantic disturbances in agent inputs (Zheng et al., 2025a). Different from previous works, our work, for the first time, develops an attack framework that targets the evaluation-looping scheme, inducing saturated evaluation loops that prolong the runtime of the system while causing only minimal changes to the original task performance (in Tab. 18). The above experimental results lead to the following implications.

(1) *Dedicated defenses are needed.* Our attack framework is tailored to the evaluation-looping mechanism of multi-agent systems, and simple defenses such as randomized evaluators and adaptive loop caps are not effective against it (in Tab. 27). This implies more advanced defenses against such attacks on the evaluation-looping mechanism are needed, which shall encourage the community to develop such more advanced defenses. (2) *Loop-level monitoring is needed.* Our attack introduces only minimal changes to the original task performance (in Tab. 18), and its effect is reflected in a longer system runtime and more executed evaluation loops. Since a longer runtime alone looks like ordinary load variation, we suggest monitoring the number of evaluation loops per sub-task, and treating an abnormal growth of this number as a warning of such attacks. (3) *Evaluator agent configurations are better to be kept unexposed.* The attack framework, like many previous works that attack the accuracy of multi-agent systems (He et al., 2025; Mo et al., 2025; Zheng et al., 2025a), requires knowledge about the configurations of the multi-agent system; therefore, we suggest keeping the evaluator configurations of released systems unexposed, which could remind the developers of the many publicly released systems to take further measures to secure their deployed systems.

Finally, we note that, this risk is not limited to openly released systems: a perturbation generator trained on released systems can also be transferred, without retraining, to another system with a different agent architecture, and still clearly prolongs its runtime (as analysed in Tabs. 23–26). This means that even multi-agent systems whose internal configurations are kept unexposed can still be threatened by such attacks. Overall, this work shall encourage the community to pay more attention to such runtime-prolonging attacks on the evaluation-looping mechanism of multi-agent systems.

B ADDITIONAL IMPLEMENTATION DETAILS & ADDITIONAL DETAILS ABOUT THE VICTIM MULTI-AGENT SYSTEMS

We evaluate our method on four victim multi-agent systems: Preacher (Liu et al., 2025a), EmoAgent (Mao et al., 2025), AutoFigure (Zhu et al., 2026), and CoAgent (Zeng et al., 2025). For each system, the pre-defined maximum number of evaluation loops allowed for each sub-task is specified as follows. In Preacher (Liu et al., 2025a), following its original setting, three types of sub-tasks are defined with maximum loop counts of 30, 15, and 10, respectively. In EmoAgent (Mao et al., 2025), the maximum number of evaluation loops is set to 12 for each sub-task. In AutoFigure (Zhu et al., 2026), this limit is 6, while in CoAgent (Zeng et al., 2025), it is 4.

REINFORCE gradient estimation. We optimize the perturbation generator using REINFORCE. Given an input x, the generator defines a stochastic perturbation policy $p_\theta(\delta \mid x)$, from which a perturbation δ is sampled. For the aggregate reward R in Eq. 9, the corresponding policy-gradient direction is

$$g_i = R \nabla_\theta \log p_\theta(\delta \mid x).$$

The loop-specific gradient g_{m,n_m} used by NIM is obtained through the same estimator by replacing R with the corresponding loop-level reward r_{m,n_m}. To estimate the protected-loop gradients

independently from the NRUA update direction, for each protected loop $(m, n_m) \in \mathcal{S}_i$, we draw an auxiliary perturbation $\tilde{\delta}_{m,n_m} \sim p_\theta(\delta \mid x)$ independently of δ and of the auxiliary perturbations for other protected loops, and obtain

$$g_{m,n_m} = r_{m,n_m}(\tilde{\delta}_{m,n_m}) \nabla_\theta \log p_\theta(\tilde{\delta}_{m,n_m} \mid x).$$

The independently sampled auxiliary perturbations are used to obtain the corresponding loop-specific rewards and gradients for the protected loops at the current iteration. These loop-specific gradients are then used to construct the protected-gradient matrix G_i in Appendix Sec. L.

The training batch size is set to 1 for all four multi-agent systems. Here, the batch size refers to the number of benign inputs used in each training iteration and is independent of the auxiliary perturbation sampling used by NIM. To keep the training evaluation budget identical, the baseline adversary uses the same total number of target-system evaluations as our method at each training iteration, with the additional evaluations used only for estimating its original reward and REINFORCE gradient. The training steps T are 100, 600, 300, and 300 for Preacher, EmoAgent, AutoFigure, and CoAgent, respectively. For AutoFigure and CoAgent, which take textual inputs, we adopt the textual perturbation generator architecture proposed in (Wang et al., 2020). For EmoAgent, which takes image inputs, we adopt the image perturbation generator architecture from (Xiao et al., 2018). For Preacher takes PDF documents as input, we convert each PDF page into an image and apply perturbations to the resulting page images using the same image perturbation architecture from (Xiao et al., 2018). We conduct our experiments on NVIDIA A100 GPU. For the training cost, training the perturbation generator of our attack method takes 16.4 hours and 18 GB of GPU memory on Preacher, 13.2 hours and 12 GB on EmoAgent, 8.6 hours and 10 GB on AutoFigure, and 11.5 hours and 16 GB on CoAgent. We also report the API cost of the whole training process, which is about \$40, \$75, \$60, and \$35 for Preacher, EmoAgent, AutoFigure, and CoAgent, respectively.

For the underlying models, for Preacher, we follow (Liu et al., 2025a) and use GPT-4o (Hurst et al., 2024) as the planning agents responsible for paper decomposition, scene planning, formatting, and textual reflection. We also use Wan-2.1 (Wan et al., 2025) and CosyVoice2 (Du et al., 2024) as the video and audio generation agents, respectively. For EmoAgent, we follow (Mao et al., 2025) and use GPT-4o (Hurst et al., 2024) as the vision-language model. For AutoFigure, we follow (Zhu et al., 2026) and use Gemini-2.5-Pro (Comanici et al., 2025) as the generative model. For CoAgent, we follow (Zeng et al., 2025) and adopt GPT-4o (Hurst et al., 2024) as the reasoning agent, together with Wan-2.1 (Wan et al., 2025) as the generative agent.

In the main paper, t_m denotes the observed wall-clock time incurred by a single evaluation loop of sub-task m, which is estimated offline prior to optimization. To obtain this estimate, prior to the optimization of the perturbation generator, we execute the complete multi-agent system once over the entire training set using the original inputs (i.e., without perturbations) and record the runtime of every loop instance during execution. The value of t_m is then computed as the average runtime of the loop instances corresponding to sub-task m.

C Additional Details about Modality-Specific Definitions of the Perturbation Set

In this section, we provide additional details about the modality-specific constraints used to construct the perturbation set $\mathcal{C}(\epsilon)$. As mentioned in Sec. B, we evaluate our attack on four systems processing different input modalities: EmoAgent takes image inputs, Preacher processes PDF inputs, while AutoFigure and CoAgent operate on text inputs.

To keep perturbed inputs perceptually similar to benign inputs across all modalities, we define the admissible set $\mathcal{C}(\epsilon)$ for each modality separately. For image inputs and converted PDF page images,

$$\mathcal{C}(\epsilon) = \{\delta : \|\delta\|_\infty \leq \epsilon\}. \tag{12}$$

During optimization, the perturbation is projected back to the admissible set via

$$\delta \leftarrow \Pi_{\mathcal{C}(\epsilon)}(\delta), \tag{13}$$

For these image inputs, the projection clips each component of δ to $[-\epsilon, \epsilon]$:

$$\Pi_{\mathcal{C}(\epsilon)}(\delta) = \mathrm{clip}(\delta, -\epsilon, \epsilon). \tag{14}$$

For text inputs, the L_2 constraint and its projection are applied in the continuous latent space.

Because the constraint and projection depend on the input modality, we specify their application below.

Image inputs (EmoAgent). For the image-input system EmoAgent, the perturbation δ is directly applied to the image pixel space. The admissible set $\mathcal{C}(\epsilon)$ and the projection $\Pi_{\mathcal{C}(\epsilon)}(\delta)$ are computed over the image pixels using the L_∞ constraint and componentwise clipping to limit visual distortion.

PDF inputs (Preacher). For the PDF-input system Preacher, we treat the attack surface as an image perturbation problem. Specifically, each page of the PDF document is converted into an image. The generator processes each page image and produces a perturbation that is added to the corresponding page image. The resulting PDF perturbations are therefore constrained using the same L_∞ admissible set and componentwise clipping as in the image-input case to limit visual distortion.

Text inputs (AutoFigure, CoAgent). Applying continuous perturbation constraints directly to discrete text tokens is computationally intractable. Following the adversarial text generation framework of T3 (Wang et al., 2020), we operate in a continuous latent representation space obtained via a text autoencoder. Specifically, the discrete input text is first encoded by the T3 tree-based autoencoder into a continuous latent embedding. The adversarial perturbation δ is then added to this latent representation, and the L_2 projection operator $\Pi_{\mathcal{C}(\epsilon)}(\delta)$ is applied to enforce the perturbation budget within the continuous embedding space. After the bounded perturbation is applied, the modified representation is decoded by the autoencoder decoder to reconstruct valid discrete text. In this way, the perturbation magnitude is explicitly controlled in the continuous latent space while preserving the validity of the decoded textual input.

D Additional Details on the Structural Property of Loop-Level Sub-Rewards

In this section, we analyze a key structural property of the loop-level sub-reward r_{m,n_m} used in Sec. 4.2 of the main paper. Specifically, we show that, within any fixed sub-task, the sub-reward of an earlier evaluation loop is guaranteed to be no smaller than that of a later evaluation loop.

Concretely, we prove that for any sub-task m, the loop-level sub-reward satisfies

$$r_{m,n_m^1} \geq r_{m,n_m^2}, \qquad \text{whenever } n_m^1 < n_m^2. \tag{15}$$

Recall that the sub-reward is defined as

$$r_{m,n_m} = t_m \cdot s_{m,n_m}, \tag{16}$$

where $t_m > 0$ denotes the runtime incurred by a single evaluation loop of sub-task m, and

$$s_{m,n_m} = \frac{1}{K} \sum_{k=1}^{K} \mathbb{1}_{k,m,n_m}, \tag{17}$$

with $\mathbb{1}_{k,m,n_m} \in \{0,1\}$ indicating whether the "revision-required" state is triggered in the n_m-th loop of sub-task m during the k-th repeated evaluation.

We next establish Eq. 15 in two steps.

Step 1: Monotonicity of loop indicators within each repeated evaluation. By the definition of the evaluation-looping scheme in the main paper, within the same repeated evaluation indexed by k, the (n_m+1)-th loop of sub-task m can only be reached if the n_m-th loop has already triggered the evaluator to output "revision-required". Therefore, if a later loop is triggered, then all earlier loops in the same sub-task must also have been triggered. Equivalently, for any $n_m^1 < n_m^2$, we have

$$\mathbb{1}_{k,m,n_m^1} \geq \mathbb{1}_{k,m,n_m^2}, \qquad \forall k \in \{1,\dots,K\}. \tag{18}$$

Step 2: Monotonicity of averaged indicators and loop-level sub-rewards. We now lift the pointwise relation in Eq. 18 to the averaged indicators and then to the sub-rewards. Averaging Eq. 18 over the K repeated evaluations gives

$$s_{m,n_m^1} = \frac{1}{K} \sum_{k=1}^{K} \mathbb{1}_{k,m,n_m^1} \geq \frac{1}{K} \sum_{k=1}^{K} \mathbb{1}_{k,m,n_m^2} = s_{m,n_m^2}. \tag{19}$$

Multiplying both sides of Eq. 19 by $t_m > 0$ yields

$$r_{m,n_m^1} = t_m \cdot s_{m,n_m^1} \geq t_m \cdot s_{m,n_m^2} = r_{m,n_m^2}. \tag{20}$$

This is exactly Eq. 15.

To conclude, we have shown that within any fixed sub-task, earlier evaluation loops always have loop-level sub-rewards no smaller than those of later loops.

E Additional Details about Preserving the Structural Prior in Eq. 9 in the Main Paper

In this section, we show that the reward design defined in Eq. 9 in the main paper preserves the structural prior. Specifically, within each sub-task, it ensures that earlier loops with smaller n_m receive no less update emphasis than later ones.

In Sec. D, we established that within any fixed sub-task m, earlier loops always have loop-level sub-rewards no smaller than those of later loops. That is, for $n_m^1 < n_m^2$, we have

$$r_{m,n_m^1} \geq r_{m,n_m^2}.$$

Since $\beta_i > 0$, the exponential function is strictly increasing, which implies

$$\exp(r_{m,n_m^1}/\beta_i) \geq \exp(r_{m,n_m^2}/\beta_i).$$

Under the Boltzmann formulation, all components share the same normalization denominator. Therefore, the ordering of the weights is preserved, and we obtain

$$\alpha^\star_{i,m,n_m^1} \geq \alpha^\star_{i,m,n_m^2}, \tag{21}$$

whenever $n_m^1 < n_m^2$.

This confirms that the reward design in Eq. 9 preserves the structural prior within each sub-task.

F Additional Ablation Studies and Further Analyses

In this section, we conduct more ablation studies and further analyses on the multi-agent system: EmoAgent (Mao et al., 2025).

More experiments on the NRUA mechanism of our method. In the NRUA mechanism of our method, at each training iteration i, our method sets the update schedule over evaluation loops in $\mathcal{U}_i$ via $\boldsymbol{\alpha}_i^\star$, which depends on the sub-rewards $\{r_{m,n_m}\}_{(m,n_m)\in\mathcal{U}_i}$ and the annealing coefficient λ_i. During training, our method linearly decreases λ_i, i.e., $\lambda_i = (T-i)/T$, so that training gradually shifts from exploration to exploitation. To validate the above designs in the NRUA mechanism of our method, we test seven variants.

We first test three rule-based variants that assign update weights heuristically across different evaluation loops rather than using $\boldsymbol{\alpha}_i^\star$, our theoretically derived optimal schedule that maximizes exploitation under sustainable exploration. In **Variant I (Proportional to sub-reward)**, the loop-level sub-reward r_{m,n_m} for each loop $(m, n_m) \in \mathcal{U}_i$ receives a weight proportional to r_{m,n_m}, normalized to sum to one. In **Variant II (Fixed emphasis on the highest-sub-reward loop)**, a fixed η_1 is set before training; the loop in $\mathcal{U}_i$ with the largest r_{m,n_m} receives weight η_1, and the rest in $\mathcal{U}_i$ share the remainder equally, each receiving $\eta_2 = (1-\eta_1)/(|\mathcal{U}_i|-1)$. We test a variety of η_1 values and report the best-performing one ($\eta_1 = 0.7$) in Tab. 6. In **Variant III (Dynamic emphasis on the highest-sub-reward loop)**, we also use η_1 and η_2 as update weights. However, instead of fixing η_1, we gradually change it from $1/|\mathcal{U}_i|$ to 1 over training to transition from exploration to exploitation. These rule-based variants heuristically balance exploration and exploitation but lack the theoretical grounding of our design.

We next test four variants of our method that, similarly to ours, use $\boldsymbol{\alpha}_i^\star$ to arrange the update schedule but differ in how they set λ_i. In **Variant IV (Ours with random λ_i)**, λ_i is sampled uniformly from $[0, 1]$ at each training iteration. In **Variant V (Ours with fixed λ_i)**, λ_i is set to 0.5 for all training

iterations. In **variant VI (Ours with exponentially decaying λ_i),** exponential decay is applied to schedule λ_i. Specifically, we set $\lambda_i = \omega^{i/T}$ with $\omega = 0.99$, following common exponential decay practices (Lambert, 1760). In **Variant VII (Ours with cosine-decaying λ_i)**, a cosine decay schedule (Loshchilov & Hutter, 2016) is applied, where $\lambda_i = \frac{1}{2}\big(1 + \cos(\pi i/T)\big)$.

We report results in Tab. 6. As shown, (1) our method and Variants IV–VII substantially outperform the rule-based variants (i.e., Variants I-III), demonstrating the effectiveness of our theoretically derived optimal update schedule $\boldsymbol{\alpha}_i^\star$. (2) Our method and Variants VI–VII, which gradually decrease λ_i, further outperform variants IV–V, in which λ_i does not decrease gradually. (3) Our method and Variants VI–VII all decrease λ_i monotonically and achieve very similar results. For simplicity, we adopt the linear decay schedule for λ_i in our implementation.

Table 6: More experiments on the NRUA mechanism of our method.

Methods	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
Variant I: Proportional to sub-reward	18.7	267.3	5669.8
Variant II: Fixed emphasis on the highest-sub-reward loop	17.8	251.9	5627.1
Variant III: Dynamic emphasis on the highest-sub-reward loop	19.3	282.9	5819.6
Variant IV: Ours with random λ_i	21.2	287.1	5978.4
Variant V: Ours with fixed λ_i	23.4	291.8	6127.9
Variant VI: Ours with exponentially decaying λ_i	35.2	371.6	6942.7
Variant VII: Ours with cosine-decaying λ_i	34.6	366.9	6887.5
Ours	**38.7**	**400.8**	**7182.1**

More experiments on the NIM mechanism of our method. In our attack method, we introduce the NIM mechanism to ensure that the prioritized updates performed in the NRUA mechanism do not interfere with evaluation loops that have already stably entered the "revision-required" state. To validate the effectiveness of NIM, we test two variants. In the first variant (**aggregate into the reward function with uniform weighting**), which corresponds to the *w/o NIM* variant reported in Tab. 5 of the main paper, instead of applying NIM, we aggregate the loop-level sub-rewards of evaluation loops that have already stably entered the "revision-required" state into the reward function, and assign each such sub-reward a uniform weight of 1 to penalize interference with these loops. In the second variant (**aggregate into the reward function with wall-clock-time weighting**), we perform a similar modification but assign each aggregated sub-reward a weight proportional to its corresponding observed wall-clock time t_m. As shown in Tab. 7, our attack method outperforms both variants. These results imply that merely aggregating the loop-level sub-rewards of loops that have already stably entered the "revision-required" state is insufficient to effectively prevent interference with them, highlighting the effectiveness of our NIM design.

Table 7: More experiments on the NIM mechanism of our method.

Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
Aggregate into the reward function with uniform weighting	23.8	292.1	6171.2
Aggregate into the reward function with wall-clock-time weighting	28.6	323.8	6554.7
Ours	**38.7**	**400.8**	**7182.1**

Impact of evaluating each perturbed input multiple times when constructing the reward function $R(\cdot)$ in the baseline adversary. In the baseline adversary, the reward function $R(\cdot)$ is obtained from K evaluations of the perturbed input instead of a single evaluation to enable a more reliable measurement of the reward. We set $K = 5$. Here, we evaluate other choices of K in the baseline adversary and report the results in Tab. 8. As shown, when the reward function is constructed from multiple evaluations of each perturbed input, the baseline adversary consistently achieves better performance than when the reward is derived from only a single evaluation. This result highlights the benefit of measuring the reward through multiple evaluations, which provides a more stable and reliable signal for the baseline adversary.

Table 8: Evaluation on evaluating each perturbed input multiple times (K times) when constructing the reward function $R(\cdot)$ in the baseline adversary.

Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
$K = 1$	6.9	124.2	3011.7
$K = 3$	8.2	155.4	3845.2
$K = 5$	9.3	179.6	4358.8
$K = 7$	9.3	180.2	4362.5
$K = 9$	9.4	180.9	4371.8

Impact of the number of evaluations K used for each perturbed input when constructing the reward function $R(\cdot)$ in our tailored attack method. In our attack method, when constructing the reward function $R(\cdot)$, similar to the baseline adversary, we obtain the reward from K evaluations of each perturbed input. In our main experiments, we set $K = 5$. Here to evaluate the impact of K, we evaluate difference choices of K and report the results in Tab. 9. As shown, the attack effectiveness increases with K when K is smaller than 5, and becomes stabilized after K reaches 5. Thus, taking the training efficiency also into the consideration, we set K to 5 in our experiments.

Table 9: Evaluation on the number of evaluations K used for each perturbed input when constructing the reward function $R(\cdot)$ in our tailored attack method.

Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
$K = 1$	28.1	310.4	6426.3
$K = 3$	36.3	381.7	7039.4
$K = 5$	38.7	400.8	7182.1
$K = 7$	38.7	401.6	7190.8
$K = 9$	38.8	402.9	7201.2

Impact of the maximum allowable perturbation magnitude ϵ. In our framework, we set the L_∞ image perturbation budget ϵ to $8/255$ for EmoAgent, consistent with prior image robustness evaluations (Addepalli et al., 2022; Qin et al., 2019). Here, we further provide results illustrating how our attack method performs with different values of ϵ. As shown in Tab. 10, across all tested values of ϵ, our attack method consistently outperforms the baseline adversary, further demonstrating its broad effectiveness. Additionally, we observe that the performance of our attack method gradually improves as ϵ increases. This is because ϵ controls the maximum allowable magnitude of perturbation that can be added to a benign input when transforming it into a perturbed one, and a larger ϵ permits stronger perturbations, thereby facilitating the generation of more effective perturbed inputs. However, note that if ϵ becomes too large, the stealthiness of the attack may no longer be preserved, which is undesirable. Therefore, we use $\epsilon = 8/255$ as the L_∞ image perturbation budget during evaluation on the EmoAgent multi-agent system (Addepalli et al., 2022; Qin et al., 2019).

Table 10: Evaluation on the maximum allowable perturbation magnitude ϵ.

Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
Baseline Adversary ($\epsilon = 4/255$)	6.8	141.2	3564.5
Ours ($\epsilon = 4/255$)	29.2	312.6	5980.0
Baseline Adversary ($\epsilon = 8/255$)	9.3	179.6	4358.8
Ours ($\epsilon = 8/255$)	38.7	400.8	7182.1
Baseline Adversary ($\epsilon = 16/255$)	11.1	203.4	4716.2
Ours ($\epsilon = 16/255$)	42.8	422.7	7913.9
Baseline Adversary ($\epsilon = 32/255$)	11.9	214.8	4892.7
Ours ($\epsilon = 32/255$)	44.0	433.5	8124.3

Impact of using $\mathbb{1}_{k,m,n_m}$ in the reward function. Recall that $\mathbb{1}_{k,m,n_m} \in \{0, 1\}$ denote whether the evaluator outputs "revision-required" in the n_m-th loop of sub-task m. In the reward function of our method, for the k-th evaluation of a perturbed input, we directly use $\mathbb{1}_{k,m,n_m}$ when constructing the reward function (**directly using $\mathbb{1}_{k,m,n_m}$**). To validate this design, we test a variant (**using the**

continuous surrogate of $\mathbb{1}_{k,m,n_m}$ instead), in which the binary indicator $\mathbb{1}_{k,m,n_m}$ is replaced by its continuous surrogate when constructing the reward function. Specifically, for the k-th evaluation of a perturbed input and the n_m-th loop of sub-task m, we extract the evaluator agent's reasoning trace using the same post-hoc querying strategy as in (Wang & Atanasova, 2025). We then feed this reasoning trace, together with the evaluator's original binary decision indicated by $\mathbb{1}_{k,m,n_m}$, into a scoring LLM (GPT-4o), which outputs a continuous score $score^c_{k,m,n_m} \in [0,1]$. Following (Lu et al., 2025; Balkır et al., 2026; Wang et al., 2025b), we carefully design the LLM-based scoring process so that $score^c_{k,m,n_m}$ serves as a calibrated continuous relaxation of the binary indicator $\mathbb{1}_{k,m,n_m}$. As shown in Tab. 11, our method achieves similar performance to this variant. Therefore, for simplicity, our attack method directly uses $\mathbb{1}_{k,m,n_m}$ without additionally converting it into a continuous surrogate.

Table 11: Evaluation on using $\mathbb{1}_{k,m,n_m}$ in the reward function.

Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
Directly using $\mathbb{1}_{k,m,n_m}$	38.7	400.8	7182.1
Using the continuous surrogate of $\mathbb{1}_{k,m,n_m}$ instead	38.7	401.3	7185.2

Impact of accounting for t_m in the reward function. In the reward design of our method, we account for the observed wall-clock time t_m incurred in each evaluation loop when constructing the loop-level sub-reward, so as to better align the reward function with our attack objective of prolonging system runtime (**with** t_m). To validate this design, we test a variant (**w/o** t_m) in which t_m is removed from the construction of the loop-level sub-reward. As shown in Tab. 12, our framework prolongs system runtime and leads to greater computational resource (token) consumption than this variant, demonstrating the effectiveness of accounting for t_m in the reward function.

Table 12: Evaluation on accounting for t_m in the reward function.

Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
w/o t_m	37.6	352.3	6943.8
with t_m	38.7	400.8	7182.1

Impact of the estimation error of t_m. As a part of the reward design, t_m is estimated offline before training, by averaging the wall-clock time of multiple loop instances of sub-task m (**averaged** t_m). To examine how much our framework relies on the accuracy of this offline estimate, we test a variant (**single-run** t_m), which instead measures t_m from a single loop instance. Although such single-run values deviate from the averaged ones by up to 30%, in Tab. 13, we observe that this variant still drops by less than 2% on all three metrics, indicating that our attack method is robust to the estimation error of t_m.

Table 13: Evaluation on the estimation error of t_m.

Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
single-run t_m	38.0	394.2	7080.6
averaged t_m	38.7	400.8	7182.1

Impact of the training batch size. In our framework, we set the training batch size to 1 for all four multi-agent systems. Here we also assess other choices of batch size from 1 to 8. In Tab. 14, we find that a batch size of 1 slightly performs best on all three metrics, and therefore we set the training batch size to 1.

Table 14: Evaluation on the training batch size.

Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
Batch size $= 1$	38.7	400.8	7182.1
Batch size $= 2$	38.2	395.2	7095.9
Batch size $= 4$	38.0	393.6	7060.4
Batch size $= 8$	38.1	394.3	7068.2

Comparison with random perturbations. To further validate the effectiveness of our attack method, we compare our framework with a random perturbation baseline, which adds random noise to the input under the same perturbation magnitude constraint $\mathcal{C}(\epsilon)$. We conduct this comparison on EmoAgent under two settings. The first one is the setting where each sub-task can run at most N_m evaluation loops (with loop caps). The second one is a setting where each sub-task can run an unlimited number of evaluation loops (without loop caps). As shown in Tab. 15, our attack method substantially outperforms random perturbations on all three metrics under both settings, demonstrating its effectiveness comes from our framework instead of from the random perturbation.

Table 15: Comparison with random perturbations on EmoAgent.

Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
Random perturbations (with loop caps)	4.3	121.4	3276.5
Ours (with loop caps)	38.7	400.8	7182.1
Random perturbations (without loop caps)	4.5	123.2	3312.8
Ours (without loop caps)	57.2	548.9	9295.4

Analysis of additional variants for allocating update weights among evaluation loops in $\mathcal{U}_i$. In the NRUA mechanism of our method, at each training iteration i, we compute the Boltzmann-form update schedule $\boldsymbol{\alpha}_i^\star$ over the entire set $\mathcal{U}_i$, which contains all evaluation loops that have not yet stably entered the "revision-required" state. In addition to the variants reported in Tab. 6, we further evaluate seven additional variants to more thoroughly validate the NRUA design.

The first two variants, like our NRUA design, operate over all evaluation loops in $\mathcal{U}_i$. Specifically, in the first variant (**uniform update weights among all loops in $\mathcal{U}_i$**), which corresponds to the *w/o NRUA* variant reported in Tab. 5 of the main paper, instead of applying NRUA, update weights are distributed uniformly across all evaluation loops in $\mathcal{U}_i$. In the second variant (**update weights only to the loop in $\mathcal{U}_i$ with the highest sub-reward**), the entire update weight is assigned to the loop in $\mathcal{U}_i$ with the largest loop-level sub-reward.

Furthermore, as discussed in the main paper, earlier evaluation loops (i.e., loops with smaller loop index n_m) in $\mathcal{U}_i$ often play a more critical role than later ones. This is because earlier loops must be triggered before later loops can be reached, thereby enabling the system runtime to be further prolonged. Motivated by this intuition, we construct two additional variants that allocate non-zero update weights *only* to the earliest evaluation loops in $\mathcal{U}_i$. For each sub-task that has at least one loop in $\mathcal{U}_i$, the earliest loop is defined as the loop with the smallest n_m among those belonging to that sub-task and contained in $\mathcal{U}_i$. The set of earliest evaluation loops in $\mathcal{U}_i$ is therefore the collection of these loops across all such sub-tasks.

In the third variant (**uniform update weights among earliest loops in $\mathcal{U}_i$**), the update weights are distributed uniformly across these earliest loops. In the fourth variant (**Boltzmann-form update weights among earliest loops in $\mathcal{U}_i$**), we solve Eq. 7 in the main paper but restrict the optimization to the earliest loops in $\mathcal{U}_i$ (rather than all loops). The resulting Boltzmann-form weights are then used to allocate updates among these earliest loops. We do not additionally test a variant that assigns update weights only to the earliest loop in $\mathcal{U}_i$ with the largest loop-level sub-reward, because due to the structural property of loop-level sub-rewards this variant is equivalent to the second variant described above.

From another perspective, since our goal is to trigger saturated evaluation loops in the multi-agent system, if a later evaluation loop has already entered the "revision-required" state, then all preceding evaluation loops within the same sub-task must also have entered that state. Based on this observa-

tion, we further evaluate three additional variants that allocate non-zero update weights *only* to the latest evaluation loops in $\mathcal{U}_i$.

In the fifth variant (**uniform update weights among latest loops in $\mathcal{U}_i$**), update weights are distributed uniformly across the latest loops in $\mathcal{U}_i$. In the sixth variant (**update weights only to the latest loop in $\mathcal{U}_i$ with the highest sub-reward**), the entire update weight is assigned to the latest loop in $\mathcal{U}_i$ that has the largest loop-level sub-reward. In the seventh variant (**Boltzmann-form update weights among latest loops in $\mathcal{U}_i$**), we solve Eq. 7 in the main paper while restricting the optimization to the latest loops in $\mathcal{U}_i$ (instead of all loops), and the resulting Boltzmann-form weights are used to allocate updates among these latest loops.

As shown in Tab. 16, our method outperforms all seven variants, further demonstrating the effectiveness of the NRUA mechanism.

Table 16: Analysis of additional variants for allocating update weights among evaluation loops in $\mathcal{U}_i$.

Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
Uniform update weights among all loops in $\mathcal{U}_i$	15.8	247.6	5538.2
Update weights only to the loop in $\mathcal{U}_i$ with the highest sub-reward	16.6	248.7	5582.8
Uniform update weights among earliest loops in $\mathcal{U}_i$	16.5	247.8	5580.6
Boltzmann-form update weights among earliest loops in $\mathcal{U}_i$	29.3	334.8	6455.3
Uniform update weights among latest loops in $\mathcal{U}_i$	11.1	194.3	4828.6
Update weights only to the latest loop in $\mathcal{U}_i$ with the highest sub-reward	10.8	192.2	4802.1
Boltzmann-form update weights among latest loops in $\mathcal{U}_i$	12.3	201.6	4908.4
Ours	38.7	400.8	7182.1

Applicability under different attacker forms. In the main paper, our attack is instantiated using an **input-aware perturbation generator**, which trains a network G_θ to generate perturbations adaptively conditioned on the input. To examine whether our core mechanisms generalize beyond this particular attacker form, we also evaluate our attack under another attacker form, namely the **universal perturbation**. In this setting, instead of using a generator network, we directly optimize a single shared perturbation that is applied to all inputs. Tab. 17 shows that under both attack formulations, our attack method achieves substantial performance improvements over the baseline adversary. These results further demonstrate the broad effectiveness of our attack method. Moreover, the universal perturbation form is more lightweight to train. On EmoAgent, training it takes only 0.8 hours, compared with 13.2 hours for the input-aware perturbation generator form and 13.4 hours for GradMDM (Pan et al., 2023). It nevertheless still clearly outperforms GradMDM in attack effectiveness (31.7 versus 10.2 in total loops). This indicates that our attack method can also be applied when the adversary has a limited training budget.

Table 17: Applicability of our attack method under different attacker forms.

Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
Baseline Adversary (universal perturbation)	6.4	131.7	3294.1
Ours (universal perturbation)	31.7	347.3	6468.2
Baseline Adversary (input-aware perturbation generator)	9.3	179.6	4358.8
Ours (input-aware perturbation generator)	38.7	400.8	7182.1

Analysis on original task performance. In the main paper, we report how applying our attack method affects the total number of evaluation loops, total runtime, and total token consumption of the system. Here, we further examine how the perturbed inputs influence the original task performance. Specifically, we report the original task performance under both our perturbed inputs and original inputs on the EmoAgent multi-agent system in Tab. 18. Following the original EmoAgent paper (Mao et al., 2025), we consider four metrics related to the original task performance, including CLIP-I, Emo-A, Emo-S, and ESR. As shown in Tab. 18, while our attack method significantly increases the system runtime and computational resource consumption, it introduces only minimal changes to the original task performance across all four metrics.

Table 18: Analysis on original task performance.

Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)	Metrics related to original task performance			
				CLIP-I (↑)	Emo-A (↑)	Emo-S (↑)	ESR (↑)
Original inputs	4.0	104.9	2579.0	0.778	64.58%	0.357	97.59%
Our perturbed inputs	38.7	400.8	7182.1	0.778	64.63%	0.362	97.61%

User study. Besides relying on the latent-space perturbation constraint, we conduct a user study to assess whether the perturbed textual inputs generated by our method preserve the semantics of the original benign inputs and remain natural to human readers. Since this study directly evaluates the textual quality of our generated perturbations, we focus on the two text-input multi-agent systems, AutoFigure and CoAgent. For each system, we randomly sample 100 inputs from the test set and generate their corresponding perturbed inputs using our method. Five participants are each shown the original benign input together with its corresponding perturbed input, with the text pairs presented in a random order. Each participant provides two ratings for every pair, each on a 1–5 scale: semantic consistency, measuring how well the perturbed input preserves the original meaning and user intent, and text naturalness, measuring whether it is fluent and resembles a normal user request. The average scores for AutoFigure and CoAgent, together with the overall average, are summarized in Tab. 19. Higher scores indicate better semantic preservation and naturalness at the text level. As shown, our perturbed inputs receive high ratings under both criteria, indicating that they largely preserve the original semantics while remaining natural to human readers.

Table 19: Human evaluation of the textual perturbations generated by our method.

Input system	Semantic consistency (↑)	Text naturalness (↑)
AutoFigure	4.7	4.8
CoAgent	4.9	4.8
Average	**4.8**	**4.8**

G Additional Experiments

In this section, we conduct additional experiments to further evaluate our attack method.

Robustness of our attack method across different system prompts. To validate the robustness of our attack method across different system prompts, we use DeepSeek-V3 (Liu et al., 2024) to rephrase the system prompt of EmoAgent (Mao et al., 2025) into 3 versions, where each version only differs in its phrasing without changing the evaluation criteria, and we keep all other settings of the system unchanged. For each of the 3 rephrased system prompts, we train the perturbation generator by our framework, and report the mean with standard deviation over the 3 rephrased system prompts (**Rephrased**). As shown in Tab. 20, our framework is robust to different system prompts.

Table 20: Robustness of our attack method across different system prompts. The results under the rephrased system prompts (**Rephrased**) are reported as the mean with standard deviation over the 3 rephrasings, while the results under the original system prompt (**Ours**) are reported **as the mean over five runs**.

Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
Rephrased	$38.1 \pm \mathbf{1.1}$	$393.9 \pm \mathbf{19.9}$	$7089.3 \pm \mathbf{482.4}$
Ours	$\mathbf{38.7}$	$\mathbf{400.8}$	$\mathbf{7182.1}$

Transferability of our attack method across different system LLM backbones. To examine whether the perturbation generator trained on one system LLM backbone performs effectively when the multi-agent system runs on a different backbone, on EmoAgent (Mao et al., 2025) we switch the system LLM backbone from GPT-4o (Hurst et al., 2024) to Gemini-2.5-Flash (Comanici et al., 2025), Claude-3.5-Sonnet (Anthropic, 2024), and Qwen2.5-VL-7B (Bai et al., 2025), and leave the rest of the system settings fixed. For each of the 3 backbones, we take the perturbation generator

trained on GPT-4o, use it to perturb the inputs of the multi-agent system that runs on this backbone, and attack the system with these perturbed inputs. As shown in Tab. 21, our framework stays effective on all 3 backbones, showing that it is not tied to a fixed system LLM and the learned perturbations are also effective when the multi-agent system runs on another LLM backbone.

Table 21: Transferability of our attack method across different system LLM backbones. The perturbation generator is trained on GPT-4o and is then directly applied to the other backbones.

System LLM backbone	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
Gemini-2.5-Flash	37.5	389.2	6945.1
Claude-3.5-Sonnet	36.8	382.4	6808.6
Qwen2.5-VL-7B	35.9	373.5	6636.3
GPT-4o (Ours)	38.7	400.8	7182.1

Transferability of our attack method across different system LLM temperatures. To study how our trained perturbation generator behaves when the multi-agent system runs at different system LLM temperatures, on EmoAgent (Mao et al., 2025), we vary the system LLM temperature to 0, 0.2, 0.4, 0.6, 0.8, and 1, and the other components of the multi-agent system remain unchanged. For each of the 6 temperatures, we reuse the perturbation generator trained at 0.5, and feed the perturbed inputs it produces to the system. As shown in Tab. 22, our framework performs stable at all 6 temperatures, showing that it is not tied to a fixed system LLM temperature.

Table 22: Transferability of our attack method across different system LLM temperatures. The perturbation generator is trained at temperature 0.5 and is then directly applied to the other temperatures.

System LLM temperature	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
0	37.6	390.2	6965.2
0.2	38.0	394.1	7042.0
0.4	38.3	397.2	7100.2
0.6	38.2	396.1	7080.1
0.8	37.7	391.2	6981.7
1	37.1	385.3	6863.9
0.5 (Ours)	38.7	400.8	7182.1

Transferability of our attack method across different multi-agent systems. To evaluate that our framework learns a perturbation generator that transfers across multi-agent systems, we evaluate four training-to-testing scenarios between multi-agent systems that differ in agent architecture: EmoAgent (Mao et al., 2025) to Preacher (Liu et al., 2025a), Preacher to EmoAgent, AutoFigure (Zhu et al., 2026) to CoAgent (Zeng et al., 2025), and CoAgent to AutoFigure. We report the four scenarios in Tabs. 23–26. In each scenario, we report three settings. (1) **No attack**, where the testing system is run on the benign input, without any perturbation added to it. (2) **Transferred from another system**, where we reuse the perturbation generator trained on the training system without retraining it, and feed the perturbed inputs it produces to the testing system. (3) **Trained and evaluated on the same system**, where the perturbation generator is trained on the attacked multi-agent system itself. For (2) **Transferred from another system** and (3) **Trained and evaluated on the same system**, we compare our method against two existing runtime-prolonging attacks, GradMDM (Pan et al., 2023) and SlothDoor (Guan et al., 2025). As shown in Tabs. 23–26, in all four scenarios, the transferred perturbation generator learned by our framework still exceeds **No attack** by a large margin on all three metrics, and also outperforms the existing attacks GradMDM (Pan et al., 2023) and SlothDoor (Guan et al., 2025). This shows that the perturbations our framework learns are general rather than system-specific, and thus can be transferred across multi-agent systems. Meanwhile, although the perturbations learned by our framework are general, the transferred perturbation generator still performs worse than the one trained on the attacked multi-agent system itself.

Table 23: Transferability of our attack method across different multi-agent systems. All results are measured on the Preacher (Liu et al., 2025a) multi-agent system. In the transferred setting, the perturbation generator is trained on EmoAgent (Mao et al., 2025) and is then directly applied to Preacher without retraining.

Setting	Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
No attack	Direct inference	12.1	638.2	20796.1
Trained and evaluated on Preacher	GradMDM (Pan et al., 2023)	30.5	1211.4	34888.3
	SlothDoor (Guan et al., 2025)	27.4	1159.8	34237.6
	Ours	**84.2**	**3045.3**	**164487.2**
Transferred from EmoAgent	GradMDM (Pan et al., 2023)	18.7	843.8	25850.9
	SlothDoor (Guan et al., 2025)	17.4	821.8	25339.3
	Ours	**57.6**	**2136.9**	**100291.1**

Table 24: Transferability of our attack method across different multi-agent systems. All results are measured on the EmoAgent (Mao et al., 2025) multi-agent system. In the transferred setting, the perturbation generator is trained on Preacher (Liu et al., 2025a) and is then directly applied to EmoAgent without retraining.

Setting	Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
No attack	Direct inference	4.0	104.9	2579.0
Trained and evaluated on EmoAgent	GradMDM (Pan et al., 2023)	10.2	190.7	4794.7
	SlothDoor (Guan et al., 2025)	9.4	181.2	4423.9
	Ours	**38.7**	**400.8**	**7182.1**
Transferred from Preacher	GradMDM (Pan et al., 2023)	6.9	145.0	3615.4
	SlothDoor (Guan et al., 2025)	6.4	139.8	3401.8
	Ours	**28.5**	**325.6**	**6327.7**

Table 25: Transferability of our attack method across different multi-agent systems. All results are measured on the CoAgent (Zeng et al., 2025) multi-agent system. In the transferred setting, the perturbation generator is trained on AutoFigure (Zhu et al., 2026) and is then directly applied to CoAgent without retraining.

Setting	Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
No attack	Direct inference	4.1	1295.1	1540.2
Trained and evaluated on CoAgent	GradMDM (Pan et al., 2023)	6.8	2162.7	2749.3
	SlothDoor (Guan et al., 2025)	6.6	2077.8	2645.7
	Ours	**33.3**	**10264.2**	**12152.5**
Transferred from AutoFigure	GradMDM (Pan et al., 2023)	4.5	1423.6	1719.3
	SlothDoor (Guan et al., 2025)	4.4	1404.7	1689.4
	Ours	**19.6**	**6075.9**	**7291.2**

Table 26: Transferability of our attack method across different multi-agent systems. All results are measured on the AutoFigure (Zhu et al., 2026) multi-agent system. In the transferred setting, the perturbation generator is trained on CoAgent (Zeng et al., 2025) and is then directly applied to AutoFigure without retraining.

Setting	Method	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
No attack	Direct inference	1.4	101.8	34292.2
Trained and evaluated on AutoFigure	GradMDM (Pan et al., 2023)	2.5	133.7	90324.8
	SlothDoor (Guan et al., 2025)	2.1	119.6	69413.4
	Ours	**4.8**	**206.6**	**209747.5**
Transferred from CoAgent	GradMDM (Pan et al., 2023)	1.6	107.6	44479.9
	SlothDoor (Guan et al., 2025)	1.5	104.9	40192.6
	Ours	**3.8**	**173.2**	**155012.1**

Analysis on defending against saturated evaluation looping attacks. To check whether our attack method still works when the multi-agent system is defended, we equip EmoAgent (Mao et al., 2025) with two defenses and run the attack under each of them. The first defense (**randomized evaluators**) rephrases the system prompt of each evaluator agent in every run, so that the system prompt of the evaluator agents is no longer fixed. The second defense (**adaptive loop caps**) randomly changes the pre-defined maximum number of evaluation loops of each sub-task in every run, while keeping the total maximum number of evaluation loops of the whole system unchanged, so that the adversary can no longer rely on a fixed loop limit. As shown in Tab. 27, our framework stays robust under both defenses, showing that such defenses are not enough to stop our attack.

Table 27: Analysis on defending against saturated evaluation looping attacks. Results are reported on EmoAgent.

Defense	Total Loops (↑)	Total Runtimes (s) (↑)	Total Tokens (↑)
No defense	38.7	400.8	7182.1
Randomized evaluators	36.2	382.4	6972.7
Adaptive loop caps	34.6	370.6	6838.6

H Additional Details about Eq. 5 in the main paper

In this section, we provide a detailed justification for Eq. 5 in the main paper. Specifically, we show that the goal of allocating as much update emphasis as possible to higher-priority loops, as indicated by larger intermediate sub-rewards r_{m,n_m}, is equivalently formulated as maximizing the weighted aggregate of sub-rewards over the simplex $\Delta(\mathcal{U}_i)$. To make this argument precise, we proceed in three steps.

Step 1: Formulating a local shift. We first formalize a local reallocation that makes $\boldsymbol{\alpha}_i$ more exploitative. Consider any two loops $(m_1, n^1_{m_1}), (m_2, n^2_{m_2}) \in \mathcal{U}_i$ such that

$$r_{m_1,n^1_{m_1}} \leq r_{m_2,n^2_{m_2}}.$$

Suppose $\alpha_{i,m_1,n^1_{m_1}} > 0$. For any sufficiently small $\gamma \in (0, \alpha_{i,m_1,n^1_{m_1}}]$, we define a shifted weight vector $\boldsymbol{\alpha}_i^{\text{shift}}$ by transferring an amount γ of weight from loop $(m_1, n^1_{m_1})$ to loop $(m_2, n^2_{m_2})$, i.e.,

$$\alpha^{\text{shift}}_{i,m,n_m} = \begin{cases} \alpha_{i,m_1,n^1_{m_1}} - \gamma, & \text{if } (m, n_m) = (m_1, n^1_{m_1}), \\ \alpha_{i,m_2,n^2_{m_2}} + \gamma, & \text{if } (m, n_m) = (m_2, n^2_{m_2}), \\ \alpha_{i,m,n_m}, & \text{otherwise.} \end{cases}$$

By construction, $\boldsymbol{\alpha}_i^{\text{shift}}$ remains in $\Delta(\mathcal{U}_i)$, since all entries remain nonnegative and the total weight is still preserved.

Step 2: Assessing the shift's effect on the objective. We next examine how this local shift affects the weighted aggregate of sub-rewards. After applying the shift, the resulting change is

$$\sum_{(m,n_m)\in\mathcal{U}_i} \alpha^{\text{shift}}_{i,m,n_m}\, r_{m,n_m} - \sum_{(m,n_m)\in\mathcal{U}_i} \alpha_{i,m,n_m}\, r_{m,n_m} = \gamma\big(r_{m_2,n^2_{m_2}} - r_{m_1,n^1_{m_1}}\big) \geq 0. \tag{22}$$

If $r_{m_2,n^2_{m_2}} > r_{m_1,n^1_{m_1}}$, then the inequality in Eq. 22 is strict. Therefore, shifting weight toward a loop with a larger sub-reward never decreases the weighted aggregate, and strictly increases it whenever the two sub-rewards are not equal.

Step 3: Iterative application and equivalence. We now connect the above local argument to the global optimization objective. Starting from any feasible $\boldsymbol{\alpha}_i \in \Delta(\mathcal{U}_i)$, we can repeatedly apply the local shift in Step 1 to progressively transfer weight from loops with smaller r_{m,n_m} to loops with larger r_{m,n_m}. Each such shift keeps $\boldsymbol{\alpha}_i$ within $\Delta(\mathcal{U}_i)$ and weakly increases

$$\sum_{(m,n_m)\in\mathcal{U}_i} \alpha_{i,m,n_m}\, r_{m,n_m}.$$

Moreover, the increase is strict unless all positive weights already lie on loop(s) with the largest current sub-reward. Therefore, the iterative process of making the update allocation "as exploitative as possible" is equivalently formulated as the optimization problem in Eq. 5 of the main paper.

Overall, the above three-step analysis rigorously establishes the equivalent optimization objective for the update weights $\boldsymbol{\alpha}_i$. In this sense, maximizing the weighted sum of sub-rewards over $\Delta(\mathcal{U}_i)$ precisely captures the intended exploitation behavior, namely, concentrating update emphasis on the loops that are currently the most promising under r_{m,n_m}.

I Proof of Proposition 1 in the Main Paper

In this section, we prove Proposition 1 in the main paper. Specifically, we characterize the solution of the normalized-entropy-constrained exploitation objective in Eq. 7. We first derive its finite-temperature Boltzmann form and then characterize the corresponding limiting cases in Sec. J. We perform our proof in three steps.

For notational brevity, let $u \in \mathcal{U}_i$ index an active loop $u = (m, n_m)$, and write $\alpha_{i,u}$ and r_u in place of α_{i,m,n_m} and r_{m,n_m}, respectively. Then Eq. 7 in the main paper can be rewritten as

$$\begin{aligned} &\arg\max_{\boldsymbol{\alpha}_i} \sum_{u\in\mathcal{U}_i} \alpha_{i,u}\, r_u \\ &\text{s.t.} \quad \sum_{u\in\mathcal{U}_i} \alpha_{i,u} = 1, \quad \alpha_{i,u} \geq 0, \quad -\sum_{u\in\mathcal{U}_i} \alpha_{i,u} \log \alpha_{i,u} \geq \lambda_i \log |\mathcal{U}_i|. \end{aligned} \tag{23}$$

Step 1: Constructing the Lagrangian. We first write the Lagrangian of Eq. 23:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\alpha}_i, \nu_i, \{\mu_{i,u}\}, \beta_i) = &- \sum_{u\in\mathcal{U}_i} \alpha_{i,u} r_u + \nu_i\Big(\sum_{u\in\mathcal{U}_i} \alpha_{i,u} - 1\Big) - \sum_{u\in\mathcal{U}_i} \mu_{i,u}\alpha_{i,u} \\ &+ \beta_i\Big(\lambda_i \log |\mathcal{U}_i| + \sum_{u\in\mathcal{U}_i} \alpha_{i,u} \log \alpha_{i,u}\Big), \end{aligned} \tag{24}$$

where ν_i, $\mu_{i,u}$, and β_i are the Lagrange multipliers associated with the simplex equality constraint, the nonnegativity constraint, and the entropy constraint, respectively. Setting the derivative with respect to $\alpha_{i,u}$ to zero yields

$$\frac{\partial \mathcal{L}}{\partial \alpha_{i,u}} = -r_u + \nu_i - \mu_{i,u} + \beta_i(1 + \log \alpha_{i,u}) = 0. \tag{25}$$

We first consider the finite-temperature regime with $\beta_i > 0$, whose condition is characterized below. In this regime, rearranging Eq. 25 gives

$$\alpha_{i,u} = \exp\Big(\frac{r_u}{\beta_i}\Big) \cdot \exp\Big(\frac{-\nu_i + \mu_{i,u} - \beta_i}{\beta_i}\Big). \tag{26}$$

Step 2: Resolving constraint activity. We next determine which constraints are active at optimum. Since the training iteration $i \in \{0, ..., T-1\}$ and $\lambda_i = \frac{T-i}{T}$ accordingly is in the range of $(0, 1]$, the entropy constraint in Eq. 23 requires

$$-\sum_{u \in \mathcal{U}_i} \alpha_{i,u} \log \alpha_{i,u} \geq \lambda_i \log |\mathcal{U}_i| > 0.$$

Therefore, a one-hot solution is infeasible. More generally, the location of the optimum depends on the target entropy relative to the maximum entropy achievable using only the reward-maximizing loops. Let $J_i^{\max}$ denote the number of loops attaining the largest sub-reward. When

$$\lambda_i \log |\mathcal{U}_i| > \log J_i^{\max}$$

and $\lambda_i < 1$, the entropy requirement cannot be satisfied by a distribution supported only on the reward-maximizing loops. In this regime, the entropy multiplier satisfies $\beta_i > 0$, and the optimum lies in the interior of the simplex. Consequently,

$$\alpha_{i,u}^\star > 0, \qquad \forall u \in \mathcal{U}_i,$$

which means that the nonnegativity constraints are inactive and thus

$$\mu_{i,u} = 0, \qquad \forall u \in \mathcal{U}_i.$$

Substituting this into Eq. 26 gives

$$\alpha_{i,u} = \exp\Big(\frac{r_u}{\beta_i}\Big) \cdot \exp\Big(\frac{-\nu_i - \beta_i}{\beta_i}\Big). \tag{27}$$

When

$$\lambda_i \log |\mathcal{U}_i| \leq \log J_i^{\max},$$

Eq. 7 instead admits optimal solutions supported entirely on the reward-maximizing loops. This regime is characterized separately in Sec. J through the $\beta_i \to 0^+$ limit.

Step 3: Applying simplex normalization. We now use the simplex constraint $\sum_{u \in \mathcal{U}_i} \alpha_{i,u} = 1$ to determine the common exponential factor in Eq. 27. Summing Eq. 27 over all $u \in \mathcal{U}_i$ yields

$$\exp\Big(\frac{-\nu_i - \beta_i}{\beta_i}\Big) = \frac{1}{\sum_{u' \in \mathcal{U}_i} \exp(r_{u'}/\beta_i)}.$$

Substituting this back into Eq. 27, we obtain

$$\alpha_{i,u}^\star = \frac{\exp(r_u/\beta_i)}{\sum_{u' \in \mathcal{U}_i} \exp(r_{u'}/\beta_i)}. \tag{28}$$

Restoring the original indices, we recover the exact Boltzmann-form solution in Eq. 8 of the main paper:

$$\alpha_{i,m,n_m}^\star = \frac{\exp\big(r_{m,n_m}/\beta_i\big)}{\sum_{(m',n'_{m'}) \in \mathcal{U}_i} \exp\big(r_{m',n'_{m'}}/\beta_i\big)}, \qquad (m, n_m) \in \mathcal{U}_i. \tag{29}$$

Overall, the above three-step derivation establishes the finite-temperature Boltzmann-form solution of Eq. 7. Sec. J further characterizes the existence and uniqueness of the corresponding finite β_i, together with the limiting cases that arise at the boundaries of the entropy constraint.

J Additional Details about Characterizing β_i

In this section, we further analyse the scalar coefficient β_i appearing in Eq. 8 in the main paper. Specifically, we characterize when a unique finite β_i exists and how the solution behaves outside this finite-temperature regime. When $|\mathcal{U}_i| = 1$, update allocation is trivial: the single remaining loop receives weight 1. We therefore bypass the normalized-entropy optimization in this case and focus below on $|\mathcal{U}_i| \geq 2$. For non-identical sub-rewards, let

$$J_i^{\max} = \left| \left\{ u \in \mathcal{U}_i : r_u = \max_{u' \in \mathcal{U}_i} r_{u'} \right\} \right|$$

denote the number of loops attaining the largest sub-reward. We show that when

$$\log J_i^{\max} < \lambda_i \log |\mathcal{U}_i| < \log |\mathcal{U}_i|,$$

there exists a unique finite $\beta_i > 0$ satisfying the entropy-matching equation. When

$$\lambda_i \log |\mathcal{U}_i| \le \log J_i^{\max},$$

Eq. 7 admits optimal solutions supported on the reward-maximizing loops; the $\beta_i \to 0^+$ Boltzmann limit selects the uniform distribution over these loops. At the other boundary, when $\lambda_i = 1$, Eq. 7 has the uniform distribution over $\mathcal{U}_i$ as its unique feasible solution, corresponding to the $\beta_i \to \infty$ limit. When all sub-rewards $\{r_{m,n_m}\}_{(m,n_m)\in\mathcal{U}_i}$ are identical, every feasible weight vector achieves the same objective value. In this case, we directly use the uniform probability vector as a canonical optimal solution, which is also the Boltzmann distribution for any finite $\beta_i > 0$.

For notational brevity, let $u \in \mathcal{U}_i$ index an active loop and write

$$\alpha_{i,u}(\beta_i) = \frac{\exp(r_u/\beta_i)}{\sum_{u'\in\mathcal{U}_i} \exp(r_{u'}/\beta_i)}.$$

Define the entropy of the Boltzmann weights as

$$\mathcal{E}(\beta_i) := -\sum_{u\in\mathcal{U}_i} \alpha_{i,u}(\beta_i) \log \alpha_{i,u}(\beta_i).$$

In the finite-temperature regime, determining β_i is equivalent to solving the entropy-matching equation

$$\mathcal{E}(\beta_i) = \lambda_i \log |\mathcal{U}_i|.$$

Step 1: Reparameterization and entropy definition. For analytical convenience, we reparameterize by

$$\tau_i := \frac{1}{\beta_i},$$

and define the partition function

$$Z(\tau_i) = \sum_{u\in\mathcal{U}_i} \exp(\tau_i r_u),$$

together with

$$A(\tau_i) := \log Z(\tau_i), \qquad W(\tau_i) := \sum_{u\in\mathcal{U}_i} \alpha_{i,u}(\tau_i)\, r_u.$$

Using these definitions, the entropy can be rewritten as

$$\mathcal{E}(\tau_i) = A(\tau_i) - \tau_i W(\tau_i). \tag{30}$$

Step 2: Differentiating the entropy. We next analyze how $\mathcal{E}(\tau_i)$ changes with τ_i. Differentiating Eq. 30 gives

$$\frac{d\mathcal{E}}{d\tau_i} = A'(\tau_i) - W(\tau_i) - \tau_i W'(\tau_i).$$

Since

$$A'(\tau_i) = W(\tau_i),$$

the above expression simplifies to

$$\frac{d\mathcal{E}}{d\tau_i} = -\tau_i W'(\tau_i). \tag{31}$$

Step 3: Determining monotonicity with respect to β_i. We now compute $W'(\tau_i)$. Differentiating $\alpha_{i,u}(\tau_i)$ yields

$$\alpha'_{i,u}(\tau_i) = \alpha_{i,u}(\tau_i)\big(r_u - W(\tau_i)\big),$$

and thus

$$W'(\tau_i) = \sum_{u\in\mathcal{U}_i} \alpha_{i,u}(\tau_i)\big(r_u - W(\tau_i)\big)^2 \ge 0.$$

The inequality is strict whenever the sub-rewards are not all identical. Hence, Eq. 31 implies

$$\frac{d\mathcal{E}}{d\tau_i} < 0.$$

Applying the chain rule with $\tau_i = 1/\beta_i$, we obtain

$$\frac{d\mathcal{E}}{d\beta_i} = \frac{d\mathcal{E}}{d\tau_i} \cdot \Big(-\frac{1}{\beta_i^2} \Big) > 0. \tag{32}$$

Therefore, $\mathcal{E}(\beta_i)$ is strictly increasing for $\beta_i > 0$.

Step 4: Limiting behavior and uniqueness. We finally examine the limiting behavior of $\mathcal{E}(\beta_i)$. Let

$$J_i^{\max} := \big|\{u \in \mathcal{U}_i : \; r_u = \max_{u' \in \mathcal{U}_i} r_{u'}\}\big|$$

denote the number of loops attaining the largest sub-reward.

As $\beta_i \to 0^+$, the Boltzmann weights concentrate on these maximizers, and hence

$$\mathcal{E}(\beta_i) \to \log J_i^{\max}.$$

As $\beta_i \to \infty$, the Boltzmann weights become uniform over $\mathcal{U}_i$, and hence

$$\mathcal{E}(\beta_i) \to \log |\mathcal{U}_i|.$$

Therefore, $\mathcal{E}(\beta_i)$ is continuous and strictly increasing on $(0, \infty)$, with range

$$(\log J_i^{\max}, \log |\mathcal{U}_i|).$$

Consequently, whenever

$$\log J_i^{\max} < \lambda_i \log |\mathcal{U}_i| < \log |\mathcal{U}_i|,$$

there exists a unique finite $\beta_i > 0$ satisfying the entropy-matching equation.

When

$$\lambda_i \log |\mathcal{U}_i| \le \log J_i^{\max},$$

the maximum objective value in Eq. 7 is attained by distributions supported entirely on the reward-maximizing loops. When the inequality is strict, such an optimal distribution is generally not unique. The limit $\beta_i \to 0^+$ selects the uniform distribution over the $J_i^{\max}$ reward-maximizing loops, which is an optimal solution of Eq. 7.

At the other boundary, when $\lambda_i = 1$, the entropy requirement reaches its maximum value $\log |\mathcal{U}_i|$. The feasible set of Eq. 7 therefore consists only of the uniform distribution over $\mathcal{U}_i$, corresponding to the limit $\beta_i \to \infty$.

In particular, when the maximizer is unique, $J_i^{\max} = 1$, so a unique finite β_i exists for every

$$0 < \lambda_i < 1.$$

At $i = 0$, where $\lambda_0 = 1$, the solution is instead the uniform distribution corresponding to the $\beta_i \to \infty$ limit.

Overall, the above four-step analysis characterizes the entropy-constrained solution over the full range $\lambda_i \in (0, 1]$. In the finite-temperature regime

$$\log J_i^{\max} < \lambda_i \log |\mathcal{U}_i| < \log |\mathcal{U}_i|,$$

the corresponding finite β_i is uniquely determined by λ_i and $\{r_{m,n_m}\}_{(m,n_m) \in \mathcal{U}_i}$. For the remaining entropy levels, the optimal allocation is obtained directly from the corresponding boundary solution characterized above. In practice, a finite β_i can be obtained by numerically solving the resulting one-dimensional equation whenever the target entropy lies in the finite-temperature regime. Together, these results provide a complete characterization of the optimal allocation represented by Eq. 8 across the full entropy range.

Computational efficiency of solving β_i. Based on the characterization above, when β_i is determined through entropy matching, the resulting equation is one-dimensional and monotone, and can

therefore be solved efficiently in practice. Specifically, since $\mathcal{E}(\beta_i)$ is continuous and strictly increasing on $(0, \infty)$, we solve

$$\mathcal{E}(\beta_i) = \lambda_i \log |\mathcal{U}_i|$$

using a standard root-finding routine such as a Newton–Raphson method safeguarded by bisection (Press, 2007). The iteration is terminated once the scalar residual satisfies

$$|\mathcal{E}(\beta_i) - \lambda_i \log |\mathcal{U}_i|| \leq \zeta,$$

where ζ is a small tolerance (10^{-12}). For the remaining entropy levels, the corresponding optimal allocation is obtained directly from the solution characterization above, without requiring numerical root finding. Since the numerical solve involves only a scalar variable β_i and repeated evaluation of Boltzmann weights over $\mathcal{U}_i$, its computational overhead is negligible compared with the forward and backward passes of the generator during training. Specifically, in our evaluation (AMD EPYC 7282 CPU), the average solve time is 0.17 ms.

K Additional Details about Adaptation of Runtime-Prolonging Baseline Attacks

As noted in Sec. 5 of the main paper, we compare against three existing runtime-prolonging adversarial attacks adapted to our saturated evaluation looping scenario: Sponge (Shumailov et al., 2021), GradMDM (Pan et al., 2023), and SlothDoor (Guan et al., 2025). We build each adaptation on top of the baseline adversary in Sec. 4.1 of the main paper.

Sponge adaptation. Sponge (Shumailov et al., 2021) originally studies energy-latency attacks using genetic algorithms. To adapt Sponge (Shumailov et al., 2021), we replace the baseline adversary's gradient-based optimization with an evolutionary search procedure. Candidate perturbations are evaluated using the reward function defined in Eq. 3 in the main paper, and populations evolve through selection, crossover, and mutation, filtering out any offspring that violate the $\mathcal{C}(\epsilon)$ constraint.

GradMDM adaptation. GradMDM (Pan et al., 2023) uses a Power Loss to rectify loss magnitudes. Following this idea, we optimize a power-transformed reward, $R_{\text{GradMDM}} = \sum_{m=1}^{M} \sum_{n_m=1}^{N_m} (r_{m,n_m})^{\rho}$, where $\rho = 4$ as in the original GradMDM configuration (Pan et al., 2023). Larger sub-rewards therefore receive stronger optimization emphasis. For direction rectification, we follow GradMDM (Pan et al., 2023), except that the rectification operates on gradients derived from our power-transformed reward rather than from loss.

SlothDoor adaptation. SlothDoor (Guan et al., 2025) adopts a factorized attack parameterization with multiple generators. Following this idea, we associate a separate perturbation generator with each sub-task and combine their outputs as the final perturbation $\delta = \sum_{m=1}^{M} \delta_m$, allowing each generator to focus on a simpler component.

L Additional Details of the Non-Interference Mechanism

In this section, we provide additional details about the Non-Interference Mechanism (NIM) of our attack method. Specifically, we outline the three practical steps used to perform the null-space projection during training.

Step 1: Gradient collection. At training iteration i, we first compute the original update direction g_i from the NRUA reward in Eq. 9 in the main paper. For each protected loop $(m, n_m) \in \mathcal{S}_i$, we independently sample an auxiliary perturbation $\tilde{\delta}_{m,n_m} \sim p_\theta(\delta \mid x)$. We then compute its loop-specific gradient direction g_{m,n_m} from the corresponding auxiliary perturbation, using the REINFORCE estimator defined in Appendix Sec. B with the corresponding loop-level reward r_{m,n_m}. Intuitively, g_{m,n_m} characterizes the local update direction that would further change the behaviour of this already-saturated loop. We stack all such protected gradient vectors row by row to construct

$$G_i = [\, g_{m,n_m}^{\top} \,]_{(m,n_m)\in\mathcal{S}_i} \in \mathbb{R}^{|\mathcal{S}_i| \times d},$$

where d is the dimension of the update direction.

Step 2: Efficient null-space projection. Although Eq. 11 in the main paper expresses the projection using an orthonormal null-space basis V_0, we do not construct this basis explicitly. Instead, we use

the equivalent orthogonal projector

$$V_0 V_0^\top = I_d - G_i^\top (G_i G_i^\top)^\dagger G_i,$$

where $(G_i G_i^\top)^\dagger$ is the Moore–Penrose pseudoinverse. The pseudoinverse is computed on the $|\mathcal{S}_i| \times |\mathcal{S}_i|$ Gram matrix, which is small when the number of protected loops is much less than the update dimension d. A numerical rank tolerance is used when evaluating the pseudoinverse.

Step 3: Update projection. Using the efficient projector above, we compute

$$g_i^{\mathrm{proj}} = g_i - G_i^\top (G_i G_i^\top)^\dagger G_i g_i.$$

This computation is algebraically equivalent to Eq. 11 in the main paper but avoids explicitly materializing the full null-space basis V_0. If there are no protected loops, the projection reduces to $g_i^{\mathrm{proj}} = g_i$. The projected direction g_i^{proj} is then used to update the perturbation generator. Since g_i^{proj} lies in $\mathrm{Null}(G_i)$, it is orthogonal to the gradient direction g_{m,n_m} associated with every loop $(m, n_m) \in \mathcal{S}_i$. Therefore, the projected update removes the components of the original update direction that lie in the span of the protected-loop gradients, while retaining the component orthogonal to this span for optimizing evaluation loops in $\mathcal{U}_i$. The null-space projection computation takes only about 30ms per training iteration, introducing only a very minor training overhead.

M Additional Details about Evaluation Metrics

In this section, we provide additional details on the evaluation metrics used in our paper: (1) Total Loop Count. We measure the total number of evaluation loops executed by the system, defined as the number of times a "revision-required" decision is triggered during execution. (2) Total Runtime. We measure the wall-clock time from when the input is provided to the system until the final output is generated. (3) Total Token Consumption. We collect the API usage records for each agent and compute the total number of tokens consumed across all agents.

N Additional Visualization about Loop-Triggering Behaviour of the Proposed Attack

In this section, we present additional visualizations of the loop-triggering behaviour of our attack method. Specifically, Fig. 4 provides an enlarged version of Fig. 3 in the main paper, illustrating the loop-triggering behaviour on the EmoAgent multi-agent system (Mao et al., 2025). In Fig. 5 and Fig. 6, we provide additional visualizations on EmoAgent. We additionally illustrate the loop-triggering behaviour of our attack on other multi-agent systems, including AutoFigure (Zhu et al., 2026), CoAgent (Zeng et al., 2025), and Preacher (Liu et al., 2025a), in Fig. 7, Fig. 8, and Fig. 9, respectively.

As shown, across these systems, the original benign inputs (Fig. 4(a), Fig. 5(a), Fig. 6(a), Fig. 7(a), Fig. 8(a), Fig. 9(a)) trigger only a limited number of evaluation loops. In contrast, our generated perturbed inputs (Fig. 4(b), Fig. 5(b), Fig. 6(b), Fig. 7(b), Fig. 8(b), Fig. 9(b)) induce many more evaluation loops across the decomposed sub-tasks, further demonstrating the effectiveness of our attack.

Beyond at the sub-task level, we also visualize the loop-triggering behaviour at the loop level. Specifically, in Fig. 10, we visualize the loop-triggering behaviour in sub-task 1 of the example illustrated in Fig. 4; in Fig. 11, we visualize the loop-triggering behaviour in sub-task 1 of the example illustrated in Fig. 5. As shown in Fig. 10(a) and Fig. 11(a), when the EmoAgent multi-agent system is fed with the original benign input, loops are only limitedly triggered. In contrast, as shown in Fig. 10(b) and Fig. 11(b), when our generated perturbed inputs are fed into the system, many more evaluation loops are triggered, In these examples, later revisions exhibit only limited visible changes despite repeated "revision-required" decisions. This further demonstrates the effectiveness of our attack method.

O Visualization of evaluation-loop activation during training

To further understand the training behaviour of our attack method, we visualize the training dynamics when optimizing the perturbation generator to attack the Preacher multi-agent system (Liu et al.,

2025a) in Fig. 12. For each training iteration, we record, for every sub-task in Preacher that employs an evaluation-looping scheme, the number of evaluation loops that enter the "revision-required" state when the perturbed input is evaluated. We report the results for the baseline adversary, the *w/o NIM* ablated variant of our attack method, and our full attack method. Each sub-plot corresponds to one sub-task in the Preacher system. Since both the baseline adversary and our attack method evaluate each perturbed input K times per training iteration, the values shown in the figure correspond to the average over these K evaluations. As shown, as training progresses, our full attack method tends to trigger an increasing number of evaluation loops that enter the "revision-required" state across the sub-tasks. Moreover, compared with both the baseline adversary and the *w/o NIM* ablated variant, our full attack method generally induces more such loops throughout training. Together, these trends provide additional empirical evidence for the effectiveness of our attack method.

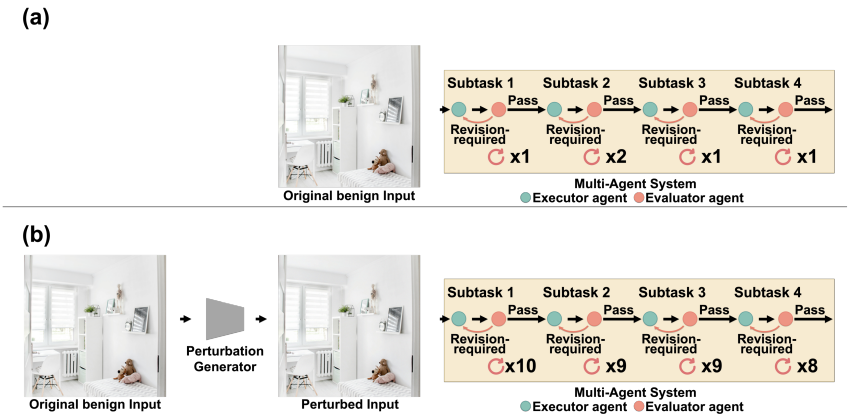

Figure 4: Visualization of loop-triggering behaviour on EmoAgent.

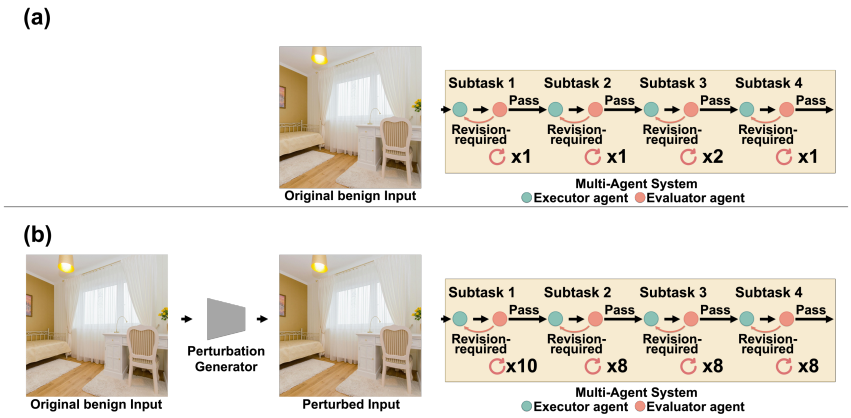

Figure 5: Visualization of loop-triggering behaviour on EmoAgent.

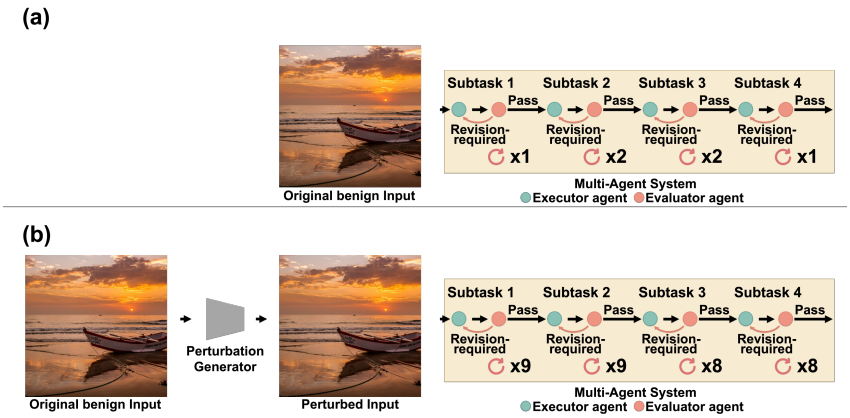

Figure 6: Visualization of loop-triggering behaviour on EmoAgent.

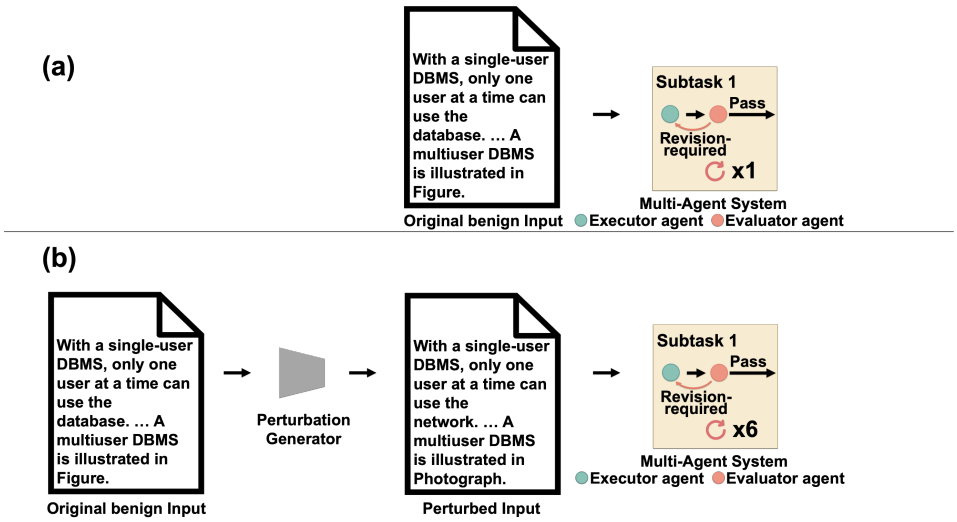

Figure 7: Visualization of loop-triggering behaviour on AutoFigure.

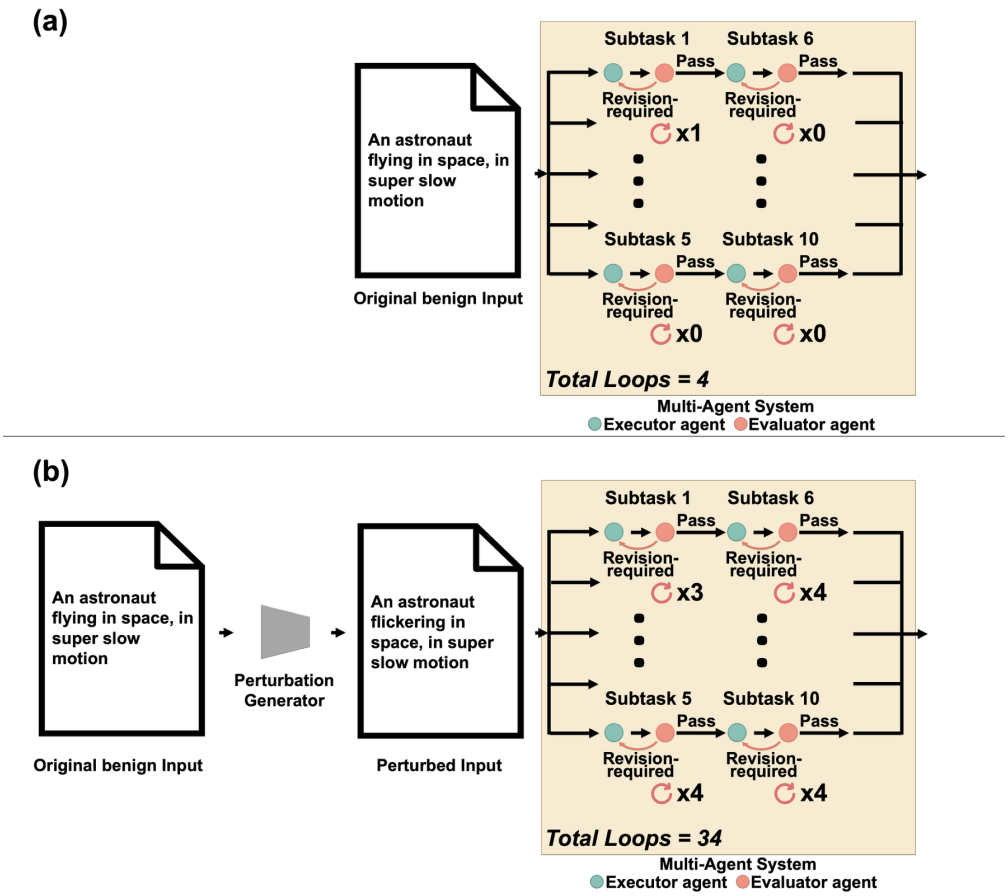

Figure 8: Visualization of loop-triggering behaviour on CoAgent.

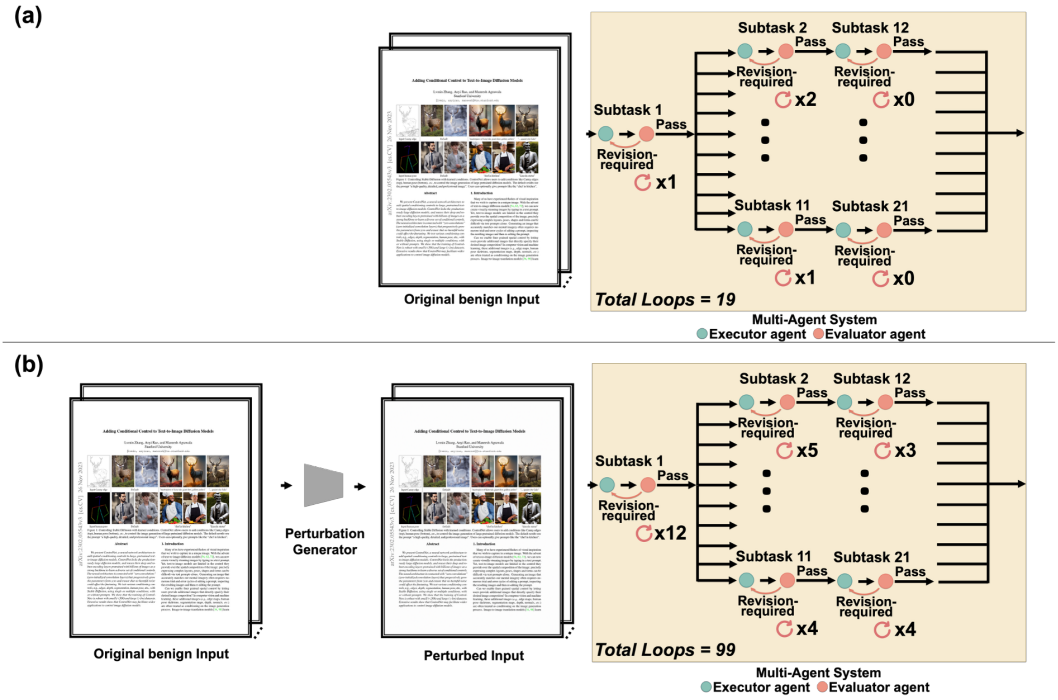

Figure 9: Visualization of loop-triggering behaviour on Preacher.

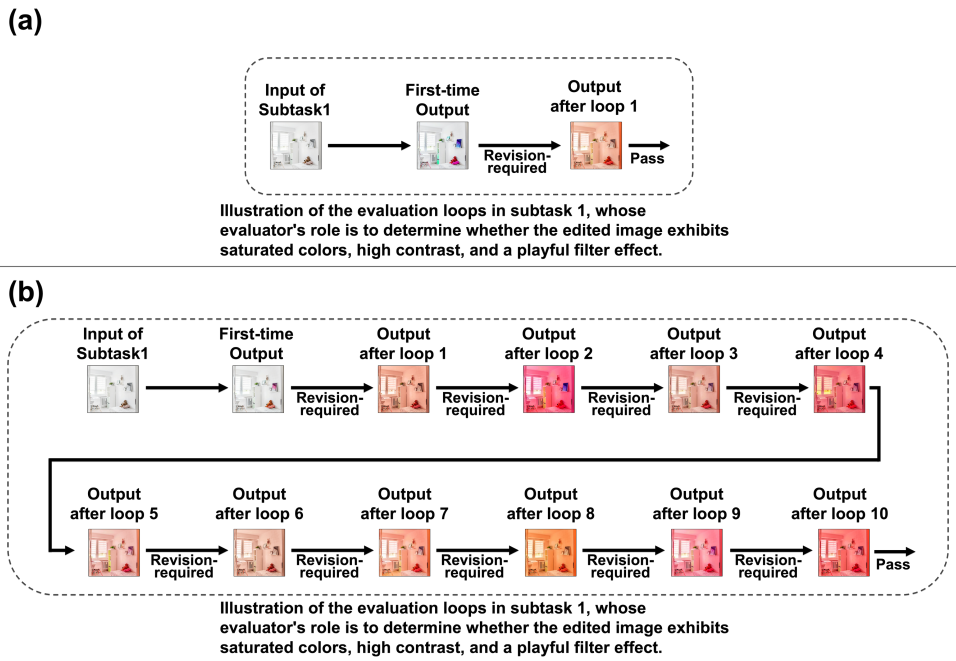

Figure 10: Loop-level visualization of loop-triggering behaviour in the EmoAgent multi-agent system. (a) The case where the system is fed with the original benign input. (b) The case where the system is fed with our generated perturbed inputs.

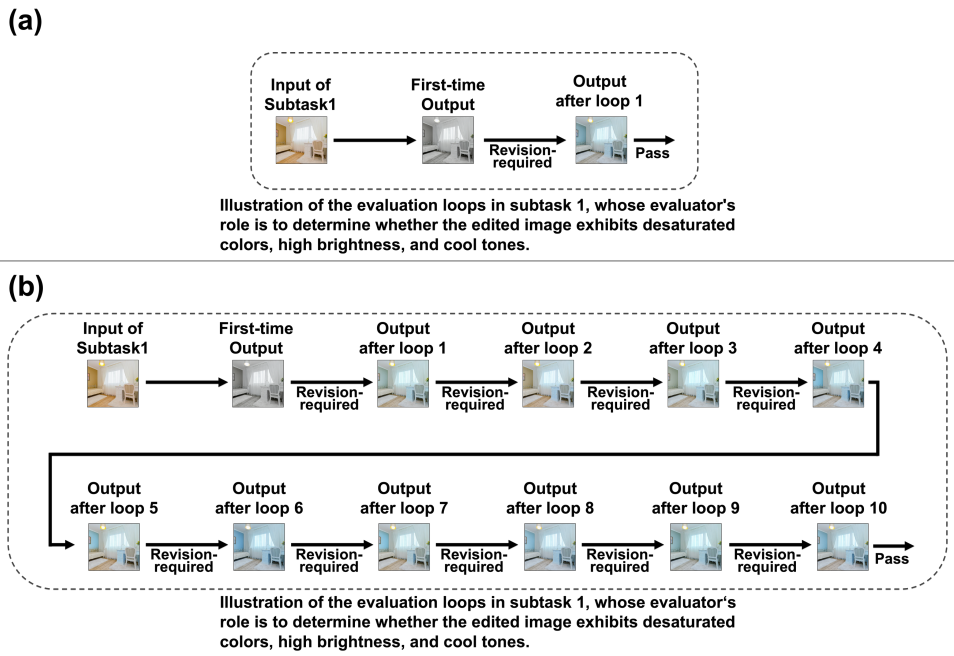

Figure 11: Loop-level visualization of loop-triggering behaviour in the EmoAgent multi-agent system. (a) The case where the system is fed with the original benign input. (b) The case where the system is fed with our generated perturbed inputs.

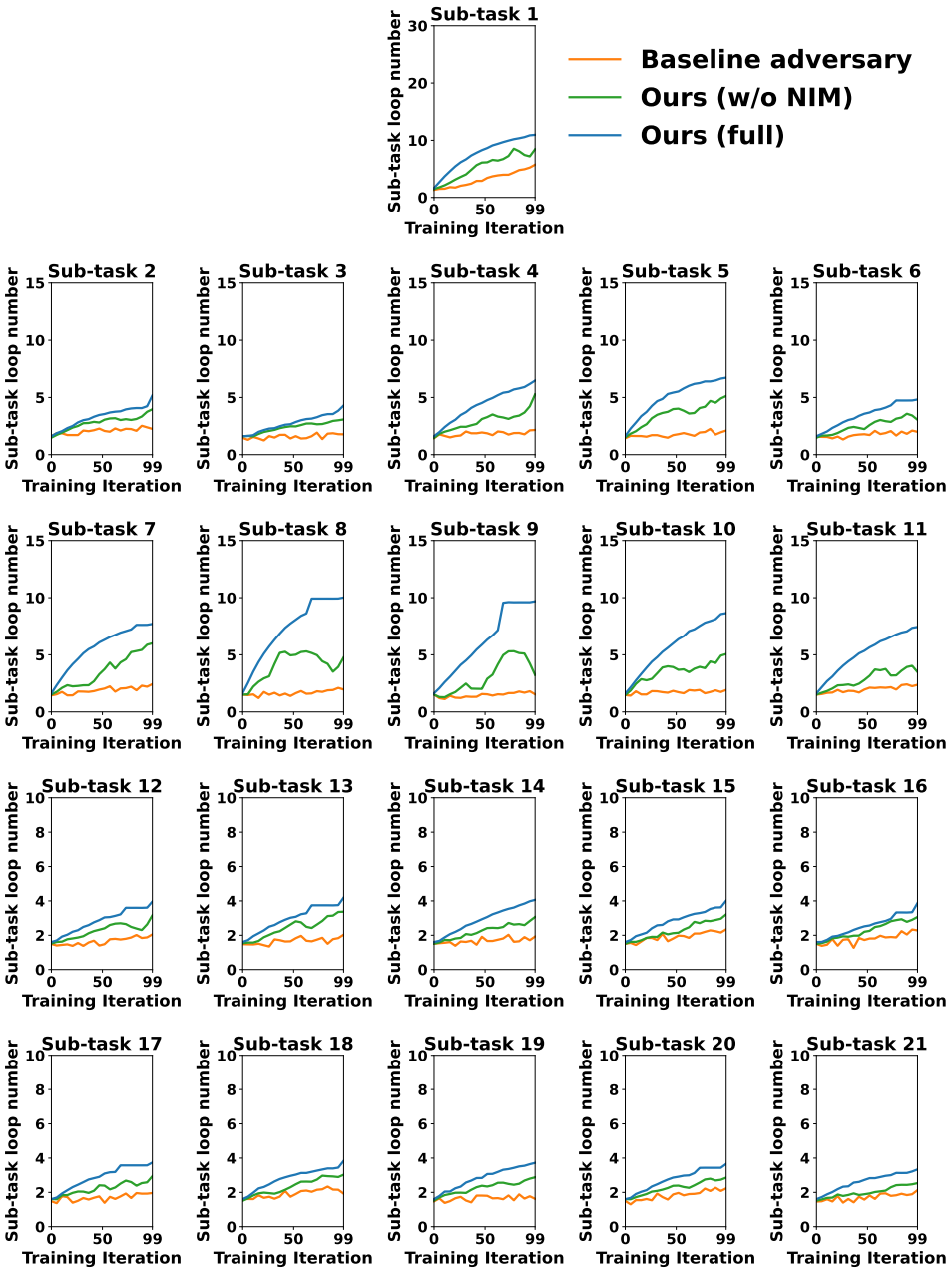

Figure 12: Training dynamics when optimizing the perturbation generator for attacking the Preacher multi-agent system (Liu et al., 2025a). Each sub-plot corresponds to a sub-task that employs an evaluation-looping scheme and shows the number of evaluation loops that enter the "revision-required" state over training iterations. We plot the results for the baseline adversary, the *w/o NIM* ablated variant of our attack method, and our full attack method. The value at each training iteration is averaged over the K evaluations of the perturbed input.