LEARNING TO ACT UNDER NOISE: ENHANCING AGENT ROBUSTNESS VIA NOISY ENVIRONMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in large language models (LLMs) have facilitated the widespread deployment of LLMs as interactive agents capable of reasoning, planning, and tool use. Despite strong performance on existing benchmarks, such agents often exhibit notable degradation when deployed in real-world settings, where environments are inherently stochastic and imperfect. We argue that this discrepancy arises from a fundamental mismatch between idealized training settings and real-world interaction dynamics, where current paradigms rely on carefully curated task instructions and stable, well-controlled environments. To address this gap, we propose NoisyAgent, an agentic training framework that explicitly incorporates environmental imperfections into the agent learning process. We identify two major sources of interaction noise in real-world scenarios: user noise, which captures ambiguity and variability in user interaction, and tool noise, which reflects failures and anomalies in tool execution. We introduce such perturbations into the training pipeline by modifying user interaction patterns and simulating tool execution results within the training environment. To stabilize training while encouraging agents to handle increasingly challenging imperfection, noise is applied to only a subset of rollouts and progressively increased in difficulty as the model adapts to the current noise level. Extensive experiments demonstrate that our approach consistently improves agent robustness under noisy environments. Our analysis reveals that training under noise condition also yields performance gains on idealized benchmarks, suggesting that controlled exposure to environmental noise promotes more generalizable reasoning and decision-making behaviors. Our findings highlight the importance of modeling interaction imperfections for bridging the gap between agent training and real-world deployment.

1 INTRODUCTION

Recent advances in large language models (LLMs) have transformed them from passive text generators into interactive agents capable of reasoning, planning, and tool use (OpenAI, 2025; Google DeepMind, 2026; Team et al., 2025b), enabling their widespread deployment in real-world applications. As these capabilities continue to improve (Team et al., 2025a; Zeng et al., 2025; Team, 2025), LLM agents have achieved strong performance across a wide range of benchmarks (Yao et al., 2024; Barres et al., 2025; He et al., 2026). However, this success does not consistently transfer to more realistic settings: when confronted with complex and dynamic environments, many agents exhibit notable performance degradation (Deng et al., 2023; Zhou et al., 2024; Xue et al., 2025).

We argue that current agent learning paradigms exhibit a fundamental gap between training conditions and real-world deployment. A common characteristic shared by existing agent training paradigms is their reliance on idealized assumptions, where agents are trained with carefully curated instructions and interact with stable, well-controlled environments (Zeng et al., 2024; Qi et al., 2025b; Jin et al., 2025). In contrast, real-world environments are inherently stochastic and imperfect. Users often exhibit diverse interaction styles and unpredictable behaviors (Gallois et al., 2005; Trippas et al., 2024; Wang et al., 2024), while external tools may return noisy, incomplete, or even failed outputs due to various uncontrollable factors (Vuddanti et al., 2025; Xiong et al., 2025). This discrepancy

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.

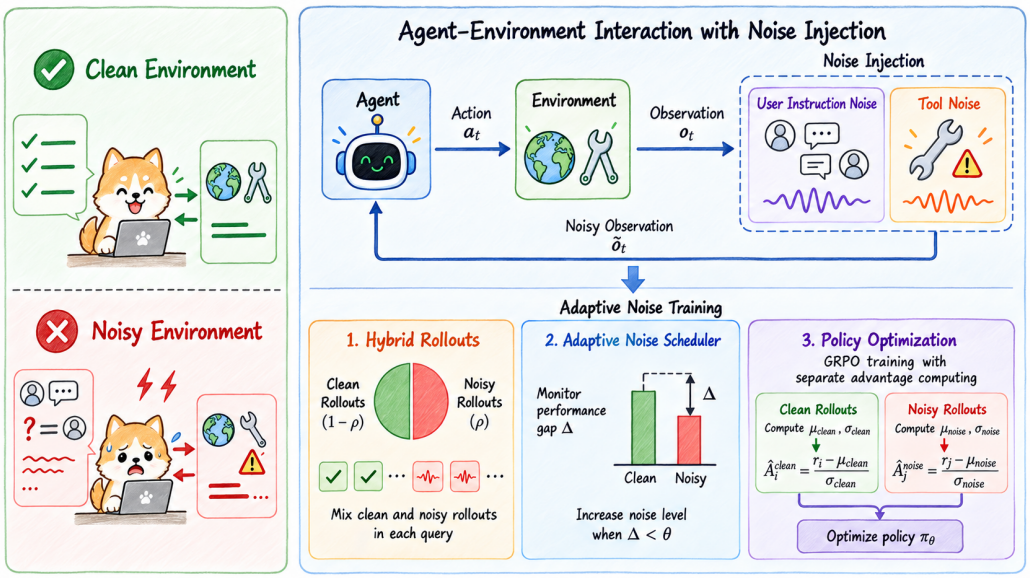


Figure 1: Overview of NoisyAgent. We inject structured perturbations into both user instructions and tool responses to simulate real-world imperfections. Training is conducted via hybrid rollouts that combine clean and noisy trajectories, together with an adaptive scheduler that increases noise difficulty based on the performance gap Δ. Policy optimization is performed with group-wise normalization to stabilize learning under heterogeneous interaction conditions.

between training conditions and deployment environments limits the robustness of current agents, often leading to degraded performance in practical applications (Rawles et al., 2025; Sun et al., 2025).

Inspired by the success of stochastic perturbations in reinforcement learning (Tobin et al., 2017; Sadeghi & Levine, 2016; Huang et al., 2022), we argue that agent robustness emerges from exposure to diverse imperfections in learning process. Rather than relying on idealized training settings and expecting agents to adapt post hoc, we explicitly incorporate environmental noise and uncertainty into the agentic training process. However, how to model and introduce such noise in agentic training remains underexplored, and naively injecting noise into the training environment can easily destabilize training dynamics, making it a non-trivial challenge.

Toward this goal, we propose NoisyAgent, an agentic RL framework for training under noisy environments. We begin by identifying representative forms of real-world noise and developing an automated pipeline to incorporate such imperfections into the training process. Concretely, we consider two major sources of interaction noise in real-world agent scenarios: user noise, which captures ambiguity and variability in user interactions, and tool noise, which simulates execution anomalies from external tools. These perturbations are introduced by modifying user instructions and simulating tool execution results within the training environment, with perturbations applied to only a subset of rollouts for each task. Training follows a curriculum schedule. Starting from mild perturbations, we progressively increase the difficulty and ratio of noise as the model exhibits sufficient robustness at each stage. Robustness is quantified by the performance gap between idealized and perturbed environments on the same tasks. This adaptive process ensures that training remains informative rather than overwhelming, while avoiding inefficient exploration of excessively noisy regimes.

Benefiting from our noise-aware training, agents achieve improved performance on benchmarks augmented with real-world noise, indicating enhanced robustness under imperfect and dynamic environments. Interestingly, we also observe consistent gains on standard, idealized benchmarks. We hypothesize that appropriately designed noise introduces controlled instability into the training environment and promotes more generalizable reasoning and decision-making. In particular, exposure to noisy and uncertain interactions encourages agents to recover from errors, resolve ambiguities, and adapt to unexpected outcomes. From this perspective, noise serves as a form of implicit difficulty augmentation, enriching the training distribution and improving robustness beyond idealized settings.

Overall, our contributions can be concluded as follows:

- We identify a fundamental gap between idealized agent training and real-world deployment, highlighting the importance of modeling environmental uncertainty for robust agent learning.
- We develop a noise-aware training framework that systematically incorporates instruction and tool perturbations into the training environment.
- Extensive experiments demonstrate that our approach consistently improves agent robustness under noisy and dynamic environments, while also yielding performance gains on standard benchmarks.

2 PRELIMINARY

2.1 AGENTIC REINFORCEMENT LEARNING

In representative agentic training paradigm, each taks can be formalized as a Partially Observable Markov Decision Process (POMDP) (Zhang et al., 2025a):

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R}). \tag{1}$$

At each step t, the agent maintains a state $s_t = (s_t^{\text{env}}, h_t, q) \in \mathcal{S}$, which captures the environment state s_t^{env}, the interaction history h_t, and the task prompt q. Based on the current observation $o_t \in \mathcal{O}$, the agent selects an action $a_t \in \mathcal{A}$, where the action space $\mathcal{A} = \mathcal{A}_{\text{user}} \cup \mathcal{A}_{\text{tool}}$ includes both user interaction and tool calling invocations. Correspondingly, the observation space $\mathcal{O} = \mathcal{O}_{\text{user}} \cup \mathcal{O}_{\text{tool}}$ consists of user-side feedback and tool execution results. Upon taking action a_t, the environment states evolves according to the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \times \mathcal{O}$, producing the next observation o_{t+1}. The training objective is to learn a policy π_θ that maximizes the expected cumulative reward $\mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} r_t\right]$ over trajectories $\tau = (o_0, a_0, o_1, a_1, \ldots, o_T)$.

A widely adopted training paradigm is Reinforcement Learning with Verifiable Rewards (RLVR) (Guo et al., 2025; Jiang et al., 2025), where a verifier evaluates whether the final environment state s_T^{env} or the full trajectory τ satisfies the task instruction given rubrics, providing a scalar reward at the trajectory level. To optimize the policy, a representative approach is Group Relative Policy Optimization (GRPO) (Shao et al., 2024), which extends PPO (Schulman et al., 2017) by computing advantages relative to a group of sampled rollouts. Concretely, given a task prompt q and N sampled trajectories $\{\tau_1, \ldots, \tau_N\}$, the advantage of each trajectory is computed as $\hat{A}_i = (r_i - \mu)/\sigma$, where μ and σ are the mean and standard deviation of the group rewards. The objective can be written as:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_q \left[\frac{1}{N} \sum_{i=1}^{N} \frac{1}{L_i} \sum_{t=1}^{L_i} \min\Big(\rho_{i,t}\, \hat{A}_i,\ \text{clip}(\rho_{i,t},\ 1 \pm \epsilon)\, \hat{A}_i \Big) \right]. \tag{2}$$

where $\rho_{i,t} = \frac{\pi_\theta(a_{i,t}|h_{i,t})}{\pi_{\text{old}}(a_{i,t}|h_{i,t})}$ and L_i is the length of trajectory τ_i. Building on this standard optimization paradigm, effective agentic training relies on access to a diverse set of interactive environments that support both user-agent interaction and tool-grounded execution (Team et al., 2026; Liu et al., 2025a).

2.2 SCALING ENVIRONMENT FOR AGENTIC TRAINING

Constructing interactive environments manually for agentic training is costly and difficult to scale. Recent work addresses this challenge by synthesizing executable environments from high-level domain specifications in a fully automated environment scaling pipeline (Tu et al., 2026). Given a domain definition, the pipeline initializes a domain-specific tool set together with a unified database schema, forming a structured domain graph $\mathcal{D}$ that serves as the foundation for executable environment generation. By sampling from this graph, each training environment can be instantiated as consisting of two tightly coupled components: a user-side construction that specifies task objectives and interaction patterns, and a tool-side construction that defines environment dynamics.

On the user side, tasks are synthesized by sampling tool chains from the domain graph and generating corresponding task queries together with interaction patterns, resulting in compositional objectives that specify both what to solve and how the user agent interacts within the environment. Formally, the user-side construction can be expressed as:

$$(q, u_{\text{int}}) = f_{\text{user}}(\mathcal{D}), \tag{3}$$

where q is the task prompt and u_{int} denotes the interaction pattern governing user-agent interactions. f_{user} denotes a simplified abstraction of the user-side construction process, which is typically assisted by strong LLMs.

On the tool side, complete executable environments are constructed by implementing structured tool APIs and underlying environment databases based on the domain graph. The sampled tool chains are instantiated as reference executions, and the tool set is further expanded along the domain graph while ensuring both correctness and verifiability of the execution process. Formally, the tool-side construction can be written as:

$$\mathcal{E} = f_{\text{tool}}(\mathcal{D}, q, u_{\text{int}}), \tag{4}$$

where $\mathcal{E}$ defines the executable environment grounded in the task specification, including tool APIs, valid state transitions, and verifiable execution paths. f_{tool} denotes a simplified abstraction of the tool-side construction process, which is typically assisted by strong LLMs.

While this design enables scalable and reliable task construction, it assumes that both components are well-specified: user interactions are restricted to be clear and helpful, while tool behaviors are stable. As a result, the resulting training environments are often idealized, leading to a mismatch between training and deployment, where real-world environments are inherently imperfect.

3 METHODOLOGY

To bridge the gap between idealized training and noisy deployment, we propose NoisyAgent, an agentic training framework that explicitly incorporates environmental imperfections into learning. We first introduce an automatic noise injection pipeline (Section 3.1) that augments training with user- and tool-side perturbations, and then present an adaptive training strategy (Section 3.2) that progressively adjusts noise difficulty to ensure stable and effective learning.

3.1 AUTOMATIC NOISE INJECTION

We systematically analyze common real-world noise and design an automated pipeline to explicitly incorporate these imperfections into any synthesized agentic training environment. Concretely, we consider two major sources of interaction noise in real-world agentic scenarios: user-side noise, which captures ambiguity and variability in user interaction patterns, and tool-side noise, which reflects failures and anomalies in external tool execution. To model such imperfections, we introduce a noise generator π_{noise} that stochastically perturbs the agent–environment interaction at each step to simulate imperfect observation from the real world.

User-side Injection. On the user side, noise is injected before the task starts by modifying the interaction patterns specified by the user. We simulate representative non-ideal interaction patterns observed in real-world scenarios, including: (1) *Ambiguous*, where user intent is underspecified; (2) *Inconsistent*, where user needs change or conflict over time; and (3) *Redundant*, where irrelevant or unnecessary information is included. Formally, given the interaction pattern u_{int} defined by any environment scaling pipeline, the injection of user-side noise can be expressed as:

$$\tilde{u}_{\text{int}} = \pi_{\text{noise}}(u_{\text{int}}), \tag{5}$$

where $\tilde{u}_{\text{int}}$ denotes the perturbed counterpart. This transformation introduces additional variability and ambiguity into user–agent interactions. To avoid inducing unreliable or misleading reward signals, we preserve the underlying task objective q, ensuring that the injected perturbations do not invalidate task solvability, but instead increase the difficulty and stochasticity of the interaction process.

Tool-side Injection. Tool-side noise is injected during agent rollouts by randomly perturbing a subset of tool execution results to simulate stochasticity in real-world environments. Specifically, we model common execution anomalies in real-world systems, including: (1) *Failures*, where tool requests return errors; (2) *Incomplete*, where outputs are truncated; (3) *Misleading*, where responses contain incorrect or inconsistent information; and (4) *Redundant*, where outputs include unnecessary details. Formally, the injection of tool-side noise can be formulated as:

$$\tilde{o}_t = \pi_{\text{noise}}(o_t), \tag{6}$$

where o_t denotes the original tool response and $\tilde{o}_t$ is the perturbed output. This process simulates imperfect tool behaviors while maintaining executable interaction dynamics.

3.2 ADAPTIVE NOISE TRAINING

Hybrid Training. The proposed automatic noise injection pipeline enables the incorporation of imperfections into agent training process. However, agent learning is highly sensitive to both task instructions and environment feedback, naively injecting uncontrolled noise can destabilize training dynamics. To preserve training stability while improving robustness, we adopt a hybrid training scheme that combines idealized and perturbed environments.

Concretely, given a task set $\mathcal{Q}$, we sample a task $q \in \mathcal{Q}$ and perform N independent rollouts in parallel environments. Among these, a subset of N_{noise} rollouts are perturbed by injecting user-side or tool-side noise, while the remaining $N - N_{\text{noise}}$ rollouts are conducted in clean, idealized environments.

Let $\mathcal{T}_{\text{clean}}$ and $\mathcal{T}_{\text{noise}}$ denote the index sets of clean and perturbed rollouts for task q; together they partition $\{1, \ldots, N\}$. In our setting, rollouts are partitioned into these two groups, and we modify the standard objective by computing advantages separately within each group while optimizing over their union. Using the sequence-level GSPO surrogate (Zheng et al., 2025), the objective is

$$\mathcal{J}(\theta) = \mathbb{E}_{q,\boldsymbol{\tau}\sim\pi_{\text{old}}}\left[\frac{1}{N}\left(\sum_{i\in\mathcal{T}_{\text{clean}}}\mathcal{L}_i(\theta, \hat{A}_i^{\text{clean}}) + \sum_{j\in\mathcal{T}_{\text{noise}}}\mathcal{L}_j(\theta, \hat{A}_j^{\text{noise}})\right)\right], \tag{7}$$

where $\boldsymbol{\tau}$ comprises rollouts collected under π_{old} in their respective environments, and

$$\mathcal{L}_k(\theta, \hat{A}) = \min\Big(s_k(\theta)\hat{A},\ \text{clip}(s_k(\theta), 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}})\,\hat{A}\Big), \tag{8}$$

$$s_k(\theta) = \exp\left(\frac{1}{L_k}\sum_{t=1}^{L_k}\log\frac{\pi_\theta(a_{k,t} \mid h_{k,t})}{\pi_{\text{old}}(a_{k,t} \mid h_{k,t})}\right). \tag{9}$$

Here, L_k counts the agent-generated tokens in rollout τ_k, $a_{k,t}$ is its t-th generated token, and $h_{k,t}$ includes the preceding interaction history. User and tool outputs provide context but are excluded from the likelihood ratio. GSPO thus uses a length-normalized sequence-level ratio and clipping, rather than clipping individual token ratios; ϵ_{low} and ϵ_{high} are the lower and upper clipping tolerances.

The advantages are computed separately within each group:

$$\hat{A}_i^{\text{clean}} = \frac{r_i - \mu_{\text{clean}}}{\sigma_{\text{clean}}}, \quad \hat{A}_j^{\text{noise}} = \frac{r_j - \mu_{\text{noise}}}{\sigma_{\text{noise}}}, \tag{10}$$

where $\mu_{\text{clean}}, \sigma_{\text{clean}}$ and $\mu_{\text{noise}}, \sigma_{\text{noise}}$ denote the mean and standard deviation of rewards computed within each group. This group-wise normalization prevents the dominance of either clean or noisy rollouts during optimization, and stabilizes training under heterogeneous interaction conditions.

Noise Scheduling. To adaptively introduce noise while maintaining training stability, we quantify the model's robustness to different noise types and adjust the noise level accordingly.

For nonempty clean and noisy rollout groups, we measure the performance gap associated with a specific noise type on the same task:

$$\Delta = \frac{1}{|\mathcal{T}_{\text{clean}}|}\sum_{i\in\mathcal{T}_{\text{clean}}}\mathbf{1}(r_i = 1) - \frac{1}{|\mathcal{T}_{\text{noise}}|}\sum_{j\in\mathcal{T}_{\text{noise}}}\mathbf{1}(r_j = 1), \tag{11}$$

where $r_k = 1$ indicates successful task completion. This gap reflects the extent to which current noise degrades task performance.

Based on this measure, we adopt a progressive noise scheduling strategy. Training is initialized in fully idealized environments as a warmup. At each stage, we control two factors: (i) the noise scale, defined as the proportion of perturbed rollouts $\rho = N_{\text{noise}}/N$; and (ii) the noise difficulty, characterized by the frequency of tool-side perturbations and the severity of user-side interaction anomalies. Once both rollout groups are available, $\Delta < \eta$, with η denoting a predefined threshold, is the performance-gap criterion for increasing both the difficulty and the proportion of that noise type. This yields a curriculum over noise, progressively increasing interaction complexity while maintaining training stability.

Table 1: Main results in noisy environments on AgentNoiseBench. We report Avg@4 and Pass@4 averaged across four runs. Best results are in bold, and second-best are underlined.

Method	AgentNoiseBench-τ^2						AgentNoiseBench-Vita					
	Retail		Airline		Telecom		Delivery		In-Store		OTA	
	Avg@4	Pass@4	Avg@4	Pass@4	Avg@4	Pass@4	Avg@4	Pass@4	Avg@4	Pass@4	Avg@4	Pass@4
Qwen3-8B	24.12	44.74	23.00	42.00	21.05	41.23	11.75	18.00	8.50	12.00	0.75	2.00
+ GRPO	30.48	50.88	33.50	54.00	31.58	53.51	15.25	24.00	14.25	23.00	2.50	4.00
+ DAPO	29.39	53.51	31.00	50.00	34.21	57.89	15.75	25.00	12.75	19.00	2.25	4.00
+ GSPO	31.80	54.39	32.50	52.00	34.43	56.14	16.00	26.00	15.00	22.00	2.75	5.00
+ Ours	**36.40**	**61.40**	**38.00**	**56.00**	**38.38**	**64.91**	**21.50**	**34.00**	**16.25**	**25.00**	**4.75**	**8.00**
Qwen3-32B	31.14	52.63	31.50	56.00	26.54	45.61	19.50	30.00	14.75	21.00	5.50	9.00
+ GRPO	38.16	61.40	37.00	62.00	36.84	62.28	23.25	35.00	19.50	28.00	7.25	11.00
+ DAPO	36.18	57.89	39.50	66.00	38.16	66.67	24.00	36.00	16.75	24.00	7.50	11.00
+ GSPO	37.72	60.53	39.00	64.00	39.25	65.79	23.75	36.00	17.50	25.00	7.50	12.00
+ Ours	**43.20**	**65.79**	**46.00**	**70.00**	**43.42**	**70.18**	**28.75**	**42.00**	**22.00**	**31.00**	**9.50**	**14.00**

4 EXPERIMENTS

4.1 EXPERIMENT SETTINGS

Training Environment. Our training environment follows the environment scaling pipeline of Tu et al. (2026). Within the synthesis pipeline, we leverage a diverse suite of high-performance LLMs for different roles. Specifically, GPT-4.1 is used for environment construction due to its favorable trade-off between cost and efficiency, while Claude-Sonnet-4.5 serves as a verifier given its strong evaluation capability. GLM-4.6 is employed to synthesize diverse instructions, forming the basis of our RL task set. Building on the synthesized tasks, we use Qwen2.5-72B-Instruct as a noise injector to introduce controlled perturbations into the interaction process. During training, Qwen2.5-72B-Instruct also acts as the user simulator to generate natural language feedback, while a Qwen3-32B model is trained as an evaluator to assign rewards based on the synthesized rubrics. Reward evaluator validation is provided in Appendix A.2.

Evaluation. We evaluate the robustness of the model on AgentNoiseBench (Wang et al., 2026), a benchmark designed to assess agent performance under real-world noise. We select two representative subsets, AgentNoiseBench-τ^2 and AgentNoiseBench-Vita for evaluation. To assess performance in idealized environments, we evaluate on representative standard agent benchmarks: (i) τ^2-Bench, a dual-control conversational benchmark where both the user and the agent can invoke tools in customer-service domains such as retail, airline, and telecom; (ii) Vita-Bench, a multi-tool agent benchmark covering real-world scenarios including food delivery, in-store services, and travel. Across all benchmarks, GPT-4.1 is used as the user simulator, and Claude-Sonnet-4.5 is used as the evaluator. Each experiment is repeated four times. We report Avg@4 and Pass@4 metrics averaged across tasks.

Implementation Details and Baselines. We adopt Qwen3-8B and Qwen3-32B as backbone models. On these backbones, we compare several representative training methods, including GRPO, DAPO, and GSPO, where our method is based on GSPO. The training batch size is set to 32, with 64 rollouts per sample. The proportion of noisy trajectories is capped at 50% of the total rollouts. The maximum prompt length is 8,192 tokens, and the maximum response length is 32,768 tokens. Detailed training configurations are provided in Appendix A. Wall-clock time analysis is provided in Appendix A.3.

4.2 MAIN RESULTS

Table 1 and Table 2 present the evaluation results under noisy and ideal settings, respectively. Additional held-out noise results are provided in Appendix B.1. We have the following observations.

Noise-aware training significantly improves robustness under imperfect environments. Across all domains and both model scales, NoisyAgent consistently achieves the best performance on AgentNoiseBench, outperforming strong baselines such as GSPO and DAPO by a clear margin in both Avg@4 and Pass@4. In contrast, while standard RL methods improve performance under clean settings, their gains diminish substantially in the presence of noise, often exhibiting notable

Table 2: Main results in ideal environments on standard agent benchmarks. We report Avg@4 and Pass@4 averaged across four runs. Best results are in bold, and second-best are underlined.

Method	τ^2-Bench						VitaBench					
	Retail		Airline		Telecom		Delivery		In-Store		OTA	
	Avg@4	Pass@4	Avg@4	Pass@4	Avg@4	Pass@4	Avg@4	Pass@4	Avg@4	Pass@4	Avg@4	Pass@4
Qwen3-8B	35.31	59.65	27.00	52.00	22.59	42.98	13.75	22.00	15.50	24.00	1.75	4.00
+ GRPO	46.05	73.68	36.50	62.00	37.28	57.89	21.00	33.00	22.75	35.00	4.25	7.00
+ DAPO	44.52	71.05	38.00	66.00	39.47	63.16	21.50	34.00	23.25	36.00	4.00	7.00
+ GSPO	46.49	74.56	37.50	64.00	39.04	61.40	21.25	33.00	23.00	35.00	4.50	8.00
+ Ours	**47.59**	**77.19**	**40.00**	**68.00**	**40.79**	**64.91**	**22.25**	**35.00**	**24.00**	**37.00**	**5.00**	**9.00**
Qwen3-32B	49.12	72.81	38.00	66.00	28.95	49.12	23.00	35.00	26.00	38.00	7.00	12.00
+ GRPO	58.11	83.33	45.00	72.00	41.67	68.42	27.00	40.00	30.25	43.00	8.75	14.00
+ DAPO	56.58	80.70	47.50	76.00	43.42	71.93	27.75	41.00	29.50	42.00	9.25	**15.00**
+ GSPO	58.55	84.21	46.50	74.00	43.86	70.18	27.25	40.00	30.50	44.00	9.00	14.00
+ Ours	**60.31**	**86.84**	**49.50**	**78.00**	**45.39**	**78.07**	**29.00**	**43.00**	**32.25**	**46.00**	**9.75**	**15.00**

Table 3: Ablation study of key components on Delivery domain of both AgentNoiseBench-Vita and VitaBench with Qwen3-8B. We report Avg@4 and Pass@4 averaged across four runs.

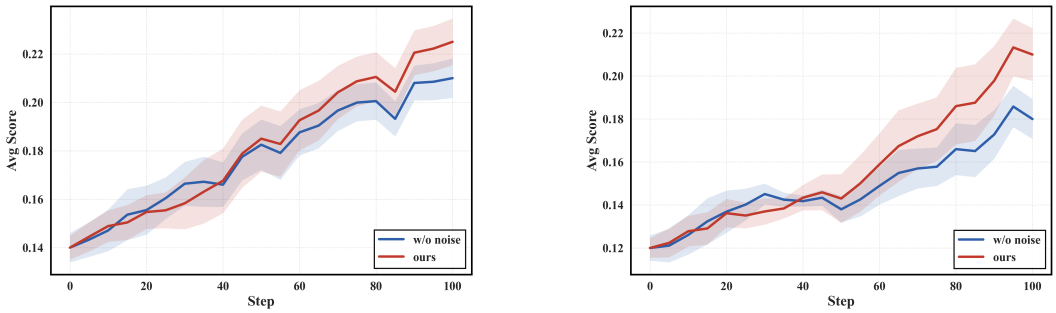
Method	AgentNoiseBench-Vita		VitaBench	
	Avg@4	Pass@4	Avg@4	Pass@4
Ours	**21.50**	**34.00**	**22.25**	**35.00**
w/o controlled injection	13.25	21.00	14.75	24.00
w/o scheduling	20.00	31.00	21.50	33.00
w/o noise	16.00	26.00	21.25	33.00
w/o training	11.75	18.00	13.75	22.00

relative degradation across domains compared with their gains in idealized settings. This suggests that existing training paradigms are less effective when facing ambiguous user instructions and imperfect tool feedback, both of which are common in real-world environments. By incorporating structured perturbations during training, our method achieves better performance in noisy environments.

Training with noise leads to consistent gains even in idealized settings. Despite being designed for noisy environments, NoisyAgent also achieves consistent improvements on standard benchmarks without noise. Across both τ^2-Bench and VitaBench, our method outperforms all baselines across domains and metrics. This indicates that training with appropriately controlled noise does not harm performance in ideal settings, and can improve overall agent capability. We attribute this to the fact that exposure to diverse and imperfect interaction patterns encourages the agent to learn more robust and effective decision-making strategies, rather than relying on brittle interaction assumptions. Further discussion is provided in Appendix C.2.

4.3 Analysis

Ablation Study. To isolate the effect of each component, we perform ablations by removing individual elements from our framework. *w/o controlled injection* removes the hybrid training scheme, applying noise to all rollouts instead of mixing clean and noisy trajectories. *w/o scheduling* removes the curriculum over noise training, using perturbations of fixed complexity throughout training. *w/o noise* reduces training to an idealized setting without any perturbations. *w/o training* evaluates the base model without RL optimization. Overall, removing any component leads to performance degradation, indicating that each part contributes to the final performance. In particular, uncontrolled noise injection (*w/o controlled injection*) causes the largest drop, suggesting that naively introducing perturbations can destabilize training. In contrast, incorporating a proper scheduling strategy further improves performance, showing that progressively adjusting noise leads to more effective and stable learning. Extended ablations and noise scheduling results are provided in Appendix B.2 and Appendix B.3.



(a) Idealized setting

(b) Noisy setting

Figure 2: Training dynamics on Vita-Bench Delivery (Qwen3-8B). We compare NoisyAgent with a baseline trained without noise under both ideal (no-noise) and noisy evaluations. Shaded bands indicate variability around the plotted curves.

Training Dynamics. Figure 2 compares the training dynamics of NoisyAgent and the baseline trained without noise under both ideal and noisy evaluations. In the early stage of training, the two methods exhibit comparable performance, as optimization is largely conducted on clean trajectories serving as a warm-up phase. The initial introduction of moderate noise may even lead to a slight degradation in performance, reflecting the increased difficulty of the perturbed trajectories. As training progresses, the model gradually adapts to noisy conditions, and the curriculum introduces increasingly challenging perturbations, raising the requirements for successful task completion. While the baseline continues to improve, its gains remain moderate. In contrast, NoisyAgent achieves more substantial improvements, particularly under the noisy evaluation, where the performance gap becomes increasingly pronounced. This trend indicates that learning in noisy environments provides informative training signals, enabling the agent to develop stronger robustness and improved performance under challenging conditions.

Interaction Pattern. Table 4 compares interaction patterns on Retail. Under noisy conditions, NoisyAgent reduces tool calls from 13.9 to 11.4 per episode (18%) relative to the base model, while GSPO yields little change. Tool usage remains similar across methods in ideal environments (6.7–7.4 calls). NoisyAgent also generates longer responses: output tokens increase from 2,014 to 4,248 under noise (2.1×), with a similar trend in ideal environments. Together with the gains in task success, these patterns suggest that additional dialogue may help NoisyAgent clarify user requirements and complete tasks with fewer tool calls. A case study and further discussion are provided in Appendix C.2.

5 Related Work

5.1 LLM as Agent

With the rapid improvement in reasoning and instruction-following capabilities, LLMs have evolved from passive text generators into agents capable of tool use, multi-step planning, and interaction with dynamic environments (Yao et al., 2022; Schick et al., 2023; Shinn et al., 2023; Wang et al., 2023). Early approaches primarily rely on hand-crafted pipelines, where reasoning–action patterns, tool schemas, and memory mechanisms are manually designed on top of frozen models (Yao et al., 2022; Shinn et al., 2023; Wu et al., 2023; Hong et al., 2024; Park et al., 2023). While effective, such prompt-level designs are brittle and do not fundamentally improve the underlying policy. More recent work instead trains agent behaviors directly via reinforcement learning with verifiable rewards (RLVR) (Guo et al., 2025; Lambert et al., 2024; Shao et al., 2024). One line of research focuses on stabilizing long-horizon training and improving credit assignment, with a variety of algorithmic advances (Shao et al., 2024; Yu et al., 2026; Zheng et al., 2025; Liu et al., 2025b; Yue et al., 2025; Yao et al., 2026). In parallel, another line of work explores scalable environment design and task construction, enabling RL training over increasingly diverse and realistic agent scenarios, including tool use and retrieval (Feng et al., 2026; Jin et al., 2025; Song et al., 2025; Shi et al., 2026), software engineering (Jimenez et al., 2024; Pan et al., 2024; Wei et al., 2026), and web or

Table 4: Interaction patterns on Retail domain with Qwen3-8B.

Method	Tool Calls	Output Tokens	Reasoning Tokens
Noisy setting:			
Base	13.9	2,014	10,897
GSPO	13.7	2,180	11,012
Ours	11.4	4,248	10,964
Ideal setting:			
Base	7.1	1,931	7,091
GSPO	7.4	1,982	7,265
Ours	6.7	3,923	7,534

GUI interaction (Zhou et al., 2024; Xie et al., 2024; Trivedi et al., 2024). Despite these advances, existing work is largely conducted under idealized settings, leaving a gap between training conditions and real-world noisy environments.

5.2 Robustness of Agent

As LLM-based agents are increasingly deployed in complex real-world settings, *robustness* has emerged as a critical concern alongside raw capability (Li et al., 2025b; Levy et al., 2025; Agrawal et al., 2025). Recent evaluations reveal robustness gaps when agents face asynchronous environmental changes (Froger et al., 2026), stochastic tool failures and flawed workflow instructions (Ao et al., 2026), or perturbed web interaction conditions (Bai et al., 2026). On the user side, prior work investigates how perturbations in prompts, clarifications, and multi-turn dialog interactions affect agent behavior (Li et al., 2025a; Deshpande et al., 2025; Qi et al., 2025a; Wang et al., 2024; Gan et al., 2026; Zhang et al., 2024a; Yang et al., 2026). These studies suggest that realistic user interactions are often noisy, under-specified, and evolving, exposing agents to a broader and more dynamic input distribution than curated settings. On the *execution side*, reliance on external tools introduces an additional source of instability, as tools may return incomplete, outdated, or erroneous outputs (Xu et al., 2024; Zhang et al., 2024b; Vuddanti et al., 2025; Kokane et al., 2024; Xiong et al., 2025; Zhan et al., 2024; Zhang et al., 2025b). Tool-parameter errors can disrupt agent interactions, motivating explicit failure analysis and trajectory-level recovery (Xiong et al., 2025; Song et al., 2024). To systematically characterize these effects, recent work proposes robustness benchmarks and diagnostic protocols (Nalbandyan et al., 2025; Wen et al., 2025; Yu et al., 2025; Lunardi et al., 2025; Siska et al., 2024). AgentNoiseBench (Wang et al., 2026) further introduces a unified taxonomy of user-side and tool-side noise with controllable perturbations, revealing consistent performance degradation across a wide range of models under realistic noise. However, existing approaches primarily focus on evaluation, leaving the problem of learning robust agent behaviors under realistic noise largely underexplored. The relationship to noisy and adversarial training is discussed in Appendix C.1.

6 Conclusion

In this work, we investigate the fundamental gap between idealized agentic training and real-world deployment, and identify the lack of environmental imperfections during training as a key factor limiting agent robustness. To address this issue, we propose a noise-aware training framework that explicitly incorporates stochasticity and imperfections into the agent learning process. By systematically modeling instruction noise and tool noise, and introducing them through an automatic noise injection pipeline, our approach exposes agents to more realistic interaction dynamics. To ensure stable optimization, we further design an adaptive training strategy that combines clean and perturbed rollouts while progressively increasing noise difficulty based on the model's robustness. Extensive experiments demonstrate that our method consistently improves agent performance under noisy and dynamic environments, validating its effectiveness in enhancing robustness. Notably, we also observe consistent gains on standard, idealized benchmarks, suggesting that controlled exposure to environmental noise promotes more generalizable reasoning and decision-making behaviors. Overall, this work highlights the importance of aligning training conditions with real-world interaction characteristics, and provides a practical framework for improving the robustness of LLM-based agents in realistic deployment settings.

LIMITATION

While our framework demonstrates consistent improvements in robustness, we note several aspects that could be further explored in future work. First, our primary goal is to investigate whether incorporating real-world interaction noise can improve the robustness of agent policies. To this end, we focus on two representative sources of noise—user-side and tool-side perturbations—and model a set of common failure patterns observed in practice. While this design captures a broad range of realistic imperfections, it does not aim to exhaustively cover all possible forms of uncertainty. In real-world environments, noise can be more complex, compositional, and dynamically evolving. Extending the framework to model richer and more diverse interaction patterns is an important direction for future work. Also, our experiments are primarily conducted in synthesized environments that approximate real-world interaction dynamics. In principle, the proposed framework is general and can be applied to any agentic environment by augmenting it with structured noise. However, due to the high cost of agentic training and the need to systematically evaluate robustness under out-of-distribution conditions, we focus on controlled settings rather than extensively benchmarking across multiple in-domain training and testing datasets. We believe that applying our framework to broader real-world and in-domain benchmarks is an important direction for future work. We leave these directions for future work. Further discussion of these limitations is provided in Appendix C.3.

AI USE STATEMENT

Generative AI models are used throughout the training and evaluation pipeline described in Section 4. GPT-4.1 supports environment construction, GLM-4.6 synthesizes task instructions, and Claude-Sonnet-4.5 serves as a verifier. Qwen2.5-72B-Instruct generates interaction perturbations and simulates users during training, while a Qwen3-32B evaluator trained on human-annotated trajectories provides reward signals. During benchmark evaluation, GPT-4.1 simulates users and Claude-Sonnet-4.5 evaluates task outcomes. The training-time reward evaluator is validated against human judgments in Appendix A.2. We also used generative AI tools to assist with manuscript drafting and editing. The authors take responsibility for the research methodology, AI-assisted materials, and reported results.

ETHICS STATEMENT

This work studies agent robustness using synthesized training environments and controlled benchmarks. User- and tool-side perturbations are introduced within these environments to model ambiguity, changing requests, incomplete information, and unreliable tool responses. Although the goal is to improve reliable assistance, greater robustness can also increase an agent's ability to persist in harmful or unauthorized workflows. Downstream deployment therefore requires appropriate authorization boundaries, monitoring, and human oversight. Generated environments and model-based reward signals may contain biases or errors, and agreement with human judgments on the evaluated samples does not establish universal reliability. Our results concern the studied perturbations and task domains and do not establish safety under all real-world conditions.

REPRODUCIBILITY STATEMENT

Section 3 specifies the noise-injection pipeline, hybrid training objective, and adaptive noise schedule. Section 4 describes the models used for environment synthesis, noise generation, user simulation, and evaluation, together with the benchmarks and baseline methods. Appendix A reports the optimization hyperparameters, rollout settings, method-specific configurations, evaluation protocol, reward-evaluator validation, and computational requirements. These details identify the training and evaluation settings used in our experiments. In particular, the four evaluation repetitions correspond to repeated rollouts per task, as clarified in Appendix A.2.

REFERENCES

Aryan Agrawal, Lisa Alazraki, Shahin Honarvar, and Marek Rei. Enhancing llm robustness to perturbed instructions: An empirical study. *arXiv preprint arXiv:2504.02733*, 2025.

Ruicheng Ao, Zeping Min, Tingyu Zhu, Wotao Yin, and Xinshang Wang. Resilibench: Evaluating agentic workflow adaptation in stochastic environments. In *International Conference on Learning Representations*, volume 2026, pp. 91062–91103, 2026.

Haoyue Bai, Dong Wang, Long Chen, Bingguang Hao, Pengyang Shao, Yonghui Yang, Yicheng He, and Chenyi Zhuang. Stressweb: A diagnostic benchmark for web agent robustness under realistic interaction variability. *arXiv preprint arXiv:2604.16385*, 2026.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Chris M Bishop. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

Qi-Zhi Cai, Min Du, Chang Liu, and Dawn Song. Curriculum adversarial training. *arXiv preprint arXiv:1805.04807*, 2018.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

Kaustubh Deshpande, Ved Sirdeshmukh, Johannes Baptist Mols, Lifeng Jin, Ed-Yeremai Hernandez-Cardona, Dean Lee, Jeremy Kritz, Willow E Primack, Summer Yue, and Chen Xing. Multichallenge: A realistic multi-turn conversation evaluation benchmark challenging to frontier llms. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 18632–18702, 2025.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. In *International Conference on Learning Representations*, volume 2026, pp. 37909–37926, 2026.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, et al. Gaia2: Benchmarking llm agents on dynamic and asynchronous environments. In *International Conference on Learning Representations*, volume 2026, pp. 119758–119789, 2026.

Cindy Gallois, Tania Ogay, and Howard Giles. Communication accommodation theory. *Theorizing about intercultural communication*, pp. 121–148, 2005.

Yujian Gan, Changling Li, Jinxia Xie, Luou Wen, Matthew Purver, and Massimo Poesio. Clarq4llm: A benchmark for models clarifying and requesting information in task-oriented dialog. *IEEE Transactions on Audio, Speech and Language Processing*, 2026.

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

Google DeepMind. Gemini 3 Pro Model Card, 2026. URL `https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Model-Card.pdf`.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Hui Su, and Xunliang Cai. Vitabench: Benchmarking llm agents with versatile interactive tasks in real-world applications. In *International Conference on Learning Representations*, volume 2026, pp. 91654–91681, 2026.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024.

Peide Huang, Mengdi Xu, Fei Fang, and Ding Zhao. Robust reinforcement learning as a stackelberg game via adaptively-regularized adversarial training. *arXiv preprint arXiv:2202.09514*, 2022.

Dongfu Jiang, Yi Lu, Zhuofeng Li, Zhiheng Lyu, Ping Nie, Haozhe Wang, Alex Su, Hui Chen, Kai Zou, Chao Du, et al. Verltool: Towards holistic agentic reinforcement learning with tool use. *arXiv preprint arXiv:2509.01055*, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Shirley Kokane, Ming Zhu, Tulika Awalgaonkar, Jianguo Zhang, Thai Hoang, Akshara Prabhakar, Zuxin Liu, Tian Lan, Liangwei Yang, Juntao Tan, et al. Toolscan: A benchmark for characterizing errors in tool-use llms. *arXiv preprint arXiv:2411.13547*, 2024.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Natan Levy, Adiel Ashrov, and Guy Katz. Statistical runtime verification for llms via robustness estimation. In *International Conference on Runtime Verification*, pp. 457–476. Springer, 2025.

Jinnan Li, Jinzhe Li, Yue Wang, Yi Chang, and Yuan Wu. Structflowbench: A structured flow benchmark for multi-turn instruction following. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 9322–9341, 2025a.

Zike Li, Mingwei Liu, Anji Li, Kaifeng He, Yanlin Wang, Xin Peng, and Zibin Zheng. A preliminary study on the robustness of code generation by large language models. *arXiv preprint arXiv:2503.20197*, 2025b.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025a.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-Like Training: A Critical Perspective. *arXiv preprint arXiv:2503.20783*, 2025b. URL https://arxiv.org/abs/2503.20783.

Riccardo Lunardi, Vincenzo Della Mea, Stefano Mizzaro, and Kevin Roitero. On robustness and reliability of benchmark-based evaluation of llms. In *ECAI 2025: 28th European Conference on Artificial Intelligence 25-30 October 2025, Bologna, Italy-Including 14th Conference on Prestigious Applications of Intelligent Systems (PAIS 2025) Proceedings*, pp. 4603–4610. SAGE Publications 1 Oliver's Yard, 55 City Road, London, EC1Y 1SP, 2025.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Takeru Miyato, Andrew M Dai, and Ian Goodfellow. Adversarial training methods for semi-supervised text classification. *arXiv preprint arXiv:1605.07725*, 2016.

Grigor Nalbandyan, Rima Shahbazyan, and Evelina Bakhturina. Score: Systematic consistency and robustness evaluation for large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pp. 470–484, 2025.

OpenAI. Introducing GPT-5.2, 2025. URL https://openai.com/index/introducing-gpt-5-2.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*, 2024.

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.

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *International conference on machine learning*, pp. 2817–2826. PMLR, 2017.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Amy Xin, Youfeng Liu, Bin Xu, Lei Hou, and Juanzi Li. Agentif: Benchmarking instruction following of large language models in agentic scenarios. *arXiv preprint arXiv:2505.16944*, 2025a.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pp. 79791–79821, 2025b.

Chris Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations*, volume 2025, pp. 406–441, 2025.

Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Yaorui Shi, Yuxin Chen, Siyuan Wang, Sihang Li, Hengxing Cai, Qi Gu, Xiang Wang, and An Zhang. Look back to reason forward: Revisitable memory for long-context llm agents. In *International Conference on Learning Representations*, volume 2026, pp. 78055–78082, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Charlotte Siska, Katerina Marazopoulou, Melissa Ailem, and James Bono. Examining the robustness of llm evaluation to the distributional assumptions of benchmarks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10406–10421, 2024.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7584–7600, 2024.

Yuchen Sun, Shanhui Zhao, Tao Yu, Hao Wen, Samith Va, Mengwei Xu, Yuanchun Li, and Chongyang Zhang. Gui-xplore: Empowering generalizable gui agents with one exploration. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19477–19486. IEEE, 2025.

Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025a.

Meituan LongCat Team. Longcat-flash technical report, 2025. URL https://arxiv.org/abs/2509.01322.

Meituan LongCat Team, Anchun Gui, Bei Li, Bingyang Tao, Bole Zhou, Borun Chen, Chao Zhang, Chengcheng Han, Chenhui Yang, Chi Zhang, et al. Introducing longcat-flash-thinking: A technical report. *arXiv preprint arXiv:2509.18883*, 2025b.

Meituan LongCat Team, Anchun Gui, Bei Li, Bingyang Tao, Bole Zhou, Borun Chen, Chao Zhang, Chen Gao, Chen Zhang, Chengcheng Han, et al. Longcat-flash-thinking-2601 technical report. *arXiv preprint arXiv:2601.16725*, 2026.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30. IEEE, 2017.

Johanne R Trippas, Sara Fahad Dawood Al Lawati, Joel Mackenzie, and Luke Gallagher. What do users really ask large language models? an initial log analysis of google bard interactions in the wild. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2703–2707, 2024.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Dunwei Tu, Hongyan Hao, Hansi Yang, Yihao Chen, Yi-Kai Zhang, Zhikang Xia, Yu Yang, Yueqing Sun, Xingchen Liu, Furao Shen, et al. Scaleenv: Scaling environment synthesis from scratch for generalist interactive tool-use agent training. *arXiv preprint arXiv:2602.06820*, 2026.

Sri Vatsa Vuddanti, Aarav Shah, Satwik Kumar Chittiprolu, Tony Song, Sunishchal Dev, Kevin Zhu, and Maheep Chaudhary. Paladin: self-correcting language model agents to cure tool-failure cases. *arXiv preprint arXiv:2509.25238*, 2025.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Jiayin Wang, Weizhi Ma, Peijie Sun, Min Zhang, and Jian-Yun Nie. Understanding user experience in large language model interactions. *arXiv preprint arXiv:2401.08329*, 2024.

Ruipeng Wang, Yuxin Chen, Yukai Wang, Chang Wu, Junfeng Fang, Xiaodong Cai, Qi Gu, Hui Su, An Zhang, Xiang Wang, et al. Agentnoisebench: Benchmarking robustness of tool-using llm agents under noisy condition. *arXiv preprint arXiv:2602.11348*, 2026.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida Wang. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. *Advances in Neural Information Processing Systems*, 38:78500–78525, 2026.

Zhihua Wen, Zhizhao Liu, Zhiliang Tian, Shilong Pan, Zhen Huang, Dongsheng Li, and Minlie Huang. Scenario-independent uncertainty estimation for llm-based question answering via factor analysis. In *Proceedings of the ACM on Web Conference 2025*, pp. 2378–2390, 2025.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Qian Xiong, Yuekai Huang, Ziyou Jiang, Zhiyuan Chang, Yujia Zheng, Tianhao Li, and Mingyang Li. Butterfly effects in toolchains: A comprehensive analysis of failed parameter filling in llm tool-agent systems. In *EMNLP (Findings)*, pp. 16712–16729, 2025.

Hongshen Xu, Zichen Zhu, Lei Pan, Zihan Wang, Su Zhu, Da Ma, Ruisheng Cao, Lu Chen, and Kai Yu. Reducing tool hallucination via reliability alignment. *arXiv preprint arXiv:2412.04141*, 2024.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. *arXiv preprint arXiv:2504.01382*, 2025.

Chenyang Yang, Yike Shi, Qianou Ma, Michael Xieyang Liu, Christian Kästner, and Tongshuang Wu. What prompts don't say: Understanding and managing underspecification in llm prompts. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 9072–9101, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Zhiyuan Yao, Yi-Kai Zhang, Yuxin Chen, Yueqing Sun, Zishan Xu, Yu Yang, Tianhao Hu, Qi Gu, Hui Su, and Xunliang Cai. Coba-rl: Capability-oriented budget allocation for reinforcement learning in llms. *arXiv preprint arXiv:2602.03048*, 2026.

Ori Yoran, Tomer Wolfson, Ori Ram, and Jonathan Berant. Making retrieval-augmented language models robust to irrelevant context. In *International Conference on Learning Representations*, volume 2024, pp. 29862–29883, 2024.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

Tong Yu, Yongcheng Jing, Xikun Zhang, Wentao Jiang, Wenjie Wu, Yingjie Wang, Wenbin Hu, Bo Du, and Dacheng Tao. Benchmarking reasoning robustness in large language models. *arXiv preprint arXiv:2503.04550*, 2025.

Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, et al. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks. *arXiv preprint arXiv:2504.05118*, 2025.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, 2024.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506, 2024.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, et al. The landscape of agentic reinforcement learning for llms: A survey. *arXiv preprint arXiv:2509.02547*, 2025a.

Rupeng Zhang, Haowei Wang, Junjie Wang, Mingyang Li, Yuekai Huang, Dandan Wang, and Qing Wang. From allies to adversaries: Manipulating llm tool-calling through adversarial injection. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 2009–2028, 2025b.

Tong Zhang, Peixin Qin, Yang Deng, Chen Huang, Wenqiang Lei, Junhong Liu, Dingnan Jin, Hongru Liang, and Tat-Seng Chua. Clamber: A benchmark of identifying and clarifying ambiguous information needs in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10746–10766, 2024a.

Yuxiang Zhang, Jing Chen, Junjie Wang, Yaxin Liu, Cheng Yang, Chufan Shi, Xinyu Zhu, Zihao Lin, Hanwen Wan, Yujiu Yang, et al. Toolbehonest: A multi-level hallucination diagnostic benchmark for tool-augmented large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 11388–11422, 2024b.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization, 2025. URL https://arxiv.org/abs/2507.18071.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

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.

A IMPLEMENTATION

This section provides the implementation details needed to interpret and reproduce the experiments. We summarize the training configuration, evaluation protocol and reward evaluator validation, and computational requirements.

A.1 TRAINING CONFIGURATION

We specify the numerical settings used for optimization, rollout generation, and noise-aware training. These settings complement the training procedure in Section 3.2.

Both backbone models are trained in BF16 precision, using vLLM (v0.8.5) for rollout generation, RoPE with $\theta = 10^6$, and RMSNorm with $\epsilon = 10^{-6}$. Tables 5–7 provide the optimization, rollout, and method-specific loss settings. Groups in which all rollouts receive identical rewards are filtered out. The default noise-ratio cap is $\rho_{\max} = 0.5$, and the scheduling threshold is $\eta = 0.2$. To avoid advancing the curriculum when both clean and noisy rollouts fail, we require at least one successful rollout in the noisy group before applying the scheduling criterion. If the noisy-group success rate is zero, the current noise configuration is retained. The scheduler in Section 3.2 operates on the noise configuration with fixed underlying LLMs; its parameter-selection results are reported in Appendix B.3.

Table 5: Optimization hyperparameters.

Hyperparameter	**Value**
Optimizer	Adam
β_1, β_2	$0.9, 0.95$
Adam ϵ	10^{-8}
Learning rate	1×10^{-6}
LR schedule	Constant
Weight decay	0.01
Gradient clipping	1.0
KL coefficient	0.0
Discount factor γ	1.0
GAE λ	1.0
PPO epochs per step	1
Data reuse epochs	2
Gradient accumulation steps	2
Total training steps	100

Table 6: Rollout and generation configuration.

Parameter	**Value**
Training batch size	32
Rollouts per sample	64
Micro batch size	1
Max prompt length	8,192 tokens
Max response length	32,768 tokens
Max sequence length	40,960 tokens
Sampling temperature (rollout)	1.0
Sampling temperature (eval)	0.0
Top-p	1.0
Max interaction turns	100

A.2 EVALUATION PROTOCOL AND REWARD EVALUATOR VALIDATION

We describe how agent performance is measured and how the training-time reward evaluator is validated against human judgments.

Table 7: Method-specific loss configurations.

Method	**Loss Type**	**Clip Range**	**Clip Ratio** c	**Loss Agg.**
GRPO	grpo	$[0.2, 0.2]$	10.0	token-mean
DAPO	dapo	$[0.2, 0.28]$	3.0	token-mean
GSPO	gspo	$[0.2, 0.28]$	3.0	seq-mean-token-mean
Ours	gspo	$[0.2, 0.28]$	3.0	seq-mean-token-mean

During training, evaluation is performed every five steps with four rollouts per task under both ideal and noisy conditions. The final evaluation also uses four runs per task: Avg@4 averages their scores, while Pass@4 measures the fraction of tasks solved at least once. These repetitions refer to evaluation rollouts, not independent training runs.

The training-time Qwen3-32B reward evaluator is further trained on internal human-annotated trajectories, with labels assigned according to task-specific rubrics. This annotation step is distinct from the synthesis of tasks and environments. Table 8 compares the original backbone and the trained evaluator against human judgments on VitaBench trajectories. The higher agreement and Cohen's κ of the trained evaluator support its use for task-completion assessment on the evaluated trajectories.

As complementary evidence, we observe sustained reward improvement during RL training, accompanied by gains on held-out benchmarks scored by a distinct evaluator, Claude-Sonnet-4.5 (Tables 1 and 2). These observations provide indirect support for the usefulness of the reward signal for optimization. They do not by themselves establish evaluator accuracy and should be interpreted alongside the human-agreement results.

Table 8: Agreement between reward evaluators and human judgments on VitaBench trajectories.

Evaluator	Human agreement	Cohen's κ
Qwen3-32B	80.6%	0.68
Trained evaluator	92.1%	0.85

A.3 Computational Resources and Cost

We report the hardware requirements and training times to characterize the computational demands of the experiments. The step-time comparison quantifies the additional cost of noise-aware training in the measured setting.

Qwen3-8B and Qwen3-32B training use 32 and 64 NVIDIA H800 GPUs, respectively. A 100-step run takes approximately 3–5 days depending on the model and domain; the reported resource usage for a Qwen3-32B run is approximately 8,000 H800 GPU-hours.

Table 9 reports average wall-clock time per step in the timing comparison. The additional three minutes for NoisyAgent amount to approximately 5% of the GSPO step time in this comparison. User-side perturbations are generated at the start of each task, and the tool-side generator is invoked when a returned response is selected for perturbation. These calls use short contexts and responses, while the agent rollout accumulates context across successive interactions.

Table 9: Average wall-clock time per training step.

Method	Time per training step
GSPO	62 min
NoisyAgent	65 min

B Additional Results

This section extends the main evaluation with results on held-out noise categories, component ablations across domains and model scales, and noise-scheduling parameters. Together, these results characterize the generality of the observed gains and their dependence on the training configuration.

B.1 Generalization to Held-out Noise

We examine generalization across distinct perturbation instances and across noise categories excluded from training. The distinction between category-level and instance-level overlap provides context for interpreting the held-out-category results.

Robustness in deployment involves handling foreseeable interaction failures as well as transferring to failures absent from training. AgentNoiseBench evaluates representative user- and tool-side imperfections that can be instantiated in interactive environments, while NoisyAgent studies how exposure to such imperfections during training affects agent performance. Consequently, some overlap at the level of broad noise categories is intentional and difficult to avoid, given the limited set of representative real-world failure modes that can be faithfully modeled in current agent environments. Training on anticipated failures is a practical objective alongside generalization to entirely unseen categories, since likely deployment problems can be analyzed and incorporated into the training environment.

Crucially, category-level overlap does not imply instance-level overlap. A perturbation depends on the user role, task objective, tool schema, available observations, and current interaction state; the same category can therefore produce different manifestations across tasks, domains, tools, and trajectories. We train on synthesized tasks using training environments from the environment-scaling pipeline of Tu et al. (2026), and evaluate on benchmark tasks from AgentNoiseBench-τ^2 and AgentNoiseBench-Vita. The evaluation perturbations and trajectories are not used during training. The generators also differ: training uses Qwen2.5-72B-Instruct to support large-scale rollouts, whereas AgentNoiseBench uses GPT-4.1 to generate evaluation-time noise. The evaluation thus tests performance across distinct perturbation instances and generation models within the benchmark protocols.

To examine transfer beyond the training categories, we evaluate *Topic Drift* on the user side and *Erroneous Outputs* on the tool side. Both categories occur in AgentNoiseBench and are excluded from NoisyAgent training. Table 10 reports Avg@4 averaged across domains for Qwen3-8B and Qwen3-32B. Telecom is excluded from this additional AgentNoiseBench-τ^2 evaluation because its user-side workflow itself invokes tools, which produced unstable evaluation outcomes. All methods in the table use the same retained domains.

Table 10: Generalization to held-out noise categories. We report Avg@4 on AgentNoiseBench-τ^2 and AgentNoiseBench-Vita.

Backbone	Method	Topic Drift		Erroneous Outputs	
		τ^2	Vita	τ^2	Vita
Qwen3-8B	Base	28.35	9.17	30.79	8.50
	GSPO	37.68	14.50	41.64	13.67
	NoisyAgent	**40.74**	**16.17**	**43.43**	**15.42**
Qwen3-32B	Base	38.83	16.67	43.20	15.67
	GSPO	47.00	20.08	52.17	19.00
	NoisyAgent	**50.54**	**22.50**	**54.54**	**21.42**

NoisyAgent improves over GSPO for both held-out categories, model scales, and benchmark families, supporting transfer to these two unseen noise types. The ideal-environment results in Table 2 provide complementary evidence that the gains extend beyond corrupted interactions.

B.2 Extended Ablation Studies

We examine whether the effects of individual training components extend across task domains and backbone scales. Table 11 extends the Qwen3-8B Delivery ablation in Table 3 to Qwen3-8B Retail

and to both Retail and Delivery with Qwen3-32B. We retain the component definitions from the main analysis and report the additional Avg@4 results.

Table 11: Additional component ablations across domains and model scales. Retail uses AgentNoiseBench-τ^2/τ^2-Bench, and Delivery uses AgentNoiseBench-Vita/VitaBench for noisy/ideal evaluation.

Backbone	Domain	Method	Noisy Avg@4	Ideal Avg@4
Qwen3-8B	Retail	NoisyAgent	**36.40**	**47.59**
		w/o controlled injection	28.95	38.82
		w/o scheduling	34.43	46.27
		w/o noise	31.80	46.49
Qwen3-32B	Retail	NoisyAgent	**43.20**	**60.31**
		w/o controlled injection	34.65	54.82
		w/o scheduling	41.23	58.99
		w/o noise	37.72	58.55
Qwen3-32B	Delivery	NoisyAgent	**28.75**	**29.00**
		w/o controlled injection	22.50	25.75
		w/o scheduling	27.75	28.00
		w/o noise	23.75	27.25

Removing controlled injection produces the largest drop in each configuration, and removing scheduling also reduces performance. The scheduling gains are more pronounced under noisy evaluation in several comparisons, consistent with its role in learning from perturbed trajectories. In ideal environments, the w/o scheduling and w/o noise variants can remain close, while the complete method obtains the highest score in each reported configuration. VitaBench also remains intrinsically challenging for smaller models, as reflected by their low absolute scores even in ideal environments (Table 2). The additional benefit of scheduling may therefore be limited by fundamental task-solving capabilities beyond noise robustness.

B.3 Noise-Ratio Sensitivity and Threshold Selection

We consider two choices in the noise curriculum: the maximum proportion of perturbed rollouts and the threshold for advancing the schedule. We report a comparison of noise-ratio caps and the pilot observations used to select the threshold.

Maximum noise proportion. We compare Qwen3-8B training with $\rho_{\max} \in \{0.25, 0.50, 0.75\}$. All configurations use the same adaptive curriculum, with the noise proportion increasing from zero; only the upper bound changes. Table 12 characterizes sensitivity to this cap under adaptive scheduling.

Table 12: Sensitivity to the maximum noise proportion with Qwen3-8B. All configurations retain adaptive noise scheduling.

Maximum noise proportion	Noisy Avg@4	Ideal Avg@4
$\rho_{\max} = 0.25$	19.75	22.00
$\rho_{\max} = 0.50$ (default)	**21.50**	**22.25**
$\rho_{\max} = 0.75$	20.50	20.75

The default cap of 0.50 gives the highest scores among the tested settings. Increasing it to 0.75 reduces both noisy- and ideal-environment scores, indicating that additional exposure to noisy trajectories is not uniformly beneficial. These comparisons concern the maximum noise proportion within the curriculum, rather than a choice between adaptive and fixed-ratio training.

Scheduling threshold. A full-scale threshold sweep is computationally expensive because each configuration requires a separate agentic RL training run with long, multi-turn rollouts (Appendix A.3). Given this cost, we select the default $\eta = 0.2$ through pilot calibration. With $\eta = 0.4$, the criterion is satisfied too readily, advancing the curriculum before sufficient adaptation; with $\eta = 0.05$, it is rarely

satisfied and limits progression to more challenging noise. The intermediate value provides a practical progression rate in these pilot runs. This calibration yields a practical default within the available computational budget; it does not establish threshold optimality or sensitivity across full training runs.

C Discussion & Analysis

This section places the approach in the context of prior robustness research and discusses how to interpret the empirical findings. We examine a possible explanation for clean-environment gains through an illustrative case, then consider the scope of synthetic environments and conditions under which noise can be harmful.

C.1 Relationship to Noisy and Adversarial Training

Section 5 reviews LLM agent training and robustness evaluation under user- and tool-side perturbations. Here, we complement that discussion by relating our approach to the broader literature on noisy and adversarial training, focusing on multi-turn interaction and trajectory-level rewards.

Controlled perturbations have long been used as training signals. Input-noise training admits a regularization interpretation (Bishop, 1995), while adversarial training optimizes against label-preserving worst-case perturbations (Goodfellow et al., 2014; Madry et al., 2017). Curriculum adversarial training adjusts perturbation strength during learning (Cai et al., 2018), and robust adversarial RL applies disturbances to sequential decision-making environments (Pinto et al., 2017). In language models, embedding-space perturbations (Miyato et al., 2016; Zhu et al., 2019) and training with irrelevant retrieved context (Yoran et al., 2024) address complementary forms of input corruption.

Multi-turn agentic RL introduces an additional dependency: a perturbed user instruction or tool response changes subsequent decisions and therefore the trajectory on which reward is computed. This couples perturbation design to task solvability and to the availability of informative reward variation during optimization. Our approach applies controlled perturbation training to this setting. The comparisons with GRPO, DAPO, and GSPO retain common backbone models, tasks, agent architecture, and reward functions to examine the effect of noise-aware training within the same agentic RL setup.

C.2 Recovery and Goal-Directed Execution under Noise

This case illustrates the behavior targeted by noise-aware training: recovering from unreliable tool feedback while continuing toward the original task objective. We examine how the agents respond to failures and carry an understood request through to completion.

Failure recovery and task completion. Figure 3 summarizes a Retail return task in which intermittent API failures and corrupted fields disrupt tool feedback while the return objective remains unchanged. Both agents verify the user, retrieve the two orders, identify the requested items, and receive confirmation to proceed. After encountering tool failures, the base model shifts to recommendations about desk lamps and student discounts and never submits the returns. NoisyAgent retries failed tool calls, continues the return workflow, and executes both required return operations. Its advantage in this example is maintaining task progress through imperfect feedback and following through on the confirmed request.

Among the 23 noisy Retail tasks where NoisyAgent succeeds and the base model fails, 78% exhibit the same gap between acquiring the necessary information and executing the critical action. This pattern connects the illustrated failure to a broader set of observed execution errors.

Connection to ideal environments. Recovery and goal retention are consistent with viewing controlled noise as interaction-level difficulty augmentation: the agent must preserve task constraints and continue toward completion when an expected interaction is disrupted. These behaviors may also help in ideal environments, where tasks still require sustained attention to the original objective. For Qwen3-8B on clean Retail tasks, the gains in Table 2 accompany fewer average tool calls than GSPO (Table 4). The case provides qualitative evidence of recovery under noise; its connection to the clean-environment gains remains a behavioral interpretation.

ORIGINAL OBJECTIVE τ^2**-Bench Retail**

Return the Mechanical Keyboard (#W5490111) and Gaming Mouse (#W7387996), with refunds to the original payment method.

NOISY TOOL FEEDBACK

Intermittent API failures (e.g., `Error 429`) and corrupted fields.

Observed in both trajectories: user verified, both orders and requested items identified, and return confirmation obtained.

Base model

Task drift after tool failures

Response to disruption

Shifts to desk lamps and student discounts.

Unrelated dialogue, turns 15–30.

Follow-through on the request

The required information and user confirmation do not lead to the requested operation.

No return API call.

Both returns remain unprocessed.

Task incomplete Reward: 0.0

NoisyAgent

Recovery and task completion

Response to disruption

Retries failed tool calls

and continues the original return workflow.

Follow-through on the request

Executes both returns at turn 22:

```
return_delivered_order_items(
 #W5490111, [keyboard])
return_delivered_order_items(
 #W7387996, [mouse])
```

Task completed Reward: 1.0

Figure 3: Recovery and task completion under noisy tool feedback. NoisyAgent retries failed tool calls and completes the original return request, while the base model drifts into unrelated recommendations despite obtaining the required information. The figure summarizes observed behaviors across the trajectories; API arguments are abbreviated.

C.3 Synthetic Environments and Failure Conditions

We discuss how the use of synthetic environments and the choice of perturbations shape the scope of the results. This connects the practical reasons for simulated training with the conditions that may limit its benefits in deployment.

Synthesized environments make it possible to scale training across executable tasks and to reproduce interactions that would be costly to obtain from real users. Production environments and logs are often proprietary or difficult to convert into standardized training tasks. Sharing environments, simulators, and evaluators across methods supports controlled comparisons, while artifacts of these LLM-based components may still affect absolute performance and transfer to production systems.

Perturbations can become harmful when their strength exceeds the policy's current capacity, their content poorly matches deployment conditions, or they invalidate a solution path despite task-preserving instructions. The controlled-injection ablations in Appendix B.2 and the cap comparison in Appendix B.3 illustrate the sensitivity of performance to noise exposure. The reported benefits apply to the evaluated perturbations and configurations; applying the framework in production would also require assessing the validity of generated disturbances and the resulting agent decisions in that environment.