StepGuard: Step-Level Data Synthesis and Balance-Aware Reinforcement Learning for Guardrail

Anonymous ACL submission

Abstract

LLM-based agents can interact with external environments through tool invocation, but this autonomy also introduces security risks such as file modification, information leakage, and unauthorized actions. Existing guardrails often operate on isolated content or completed trajectories, leaving pre-execution monitoring of step-level actions underexplored. We propose StepGuard, a step-level guardrail model for both offline trajectory diagnosis and online pre-action guarding. To train it, we introduce StepGen, an automatic data engine that generates step-annotated safe/unsafe trajectory groups with localized risky actions and context-matched safe alternatives, enabling the model to judge whether a tool action is safe under the current trajectory context. We further propose Balance-GRPO, a bias-aware reinforcement learning objective that calibrates safe/unsafe decisions by reweighting advantages according to class-wise accuracy gaps. Experiments show that StepGuard-4B achieves the best overall performance among existing guardrail baselines in static safety evaluation, with average accuracy comparable to GPT-5.4. In dynamic guarded-agent evaluation, StepGuard reduces average ASR on AgentDojo and AgentDyn by 77.3% relative to no defense with only a 2.8% utility drop, and achieves a low malicious score on AgentHarm.

1 Introduction

The rapid proliferation of Large Language Model (LLM)-based agents has expanded LLMs from text generation to tool use, resource access, and environment interaction across diverse real-world applications (Qian et al., 2024; Luo et al., 2025b; Wu et al., 2023). Unlike conventional chatbots, autonomous agents can take consequential actions in external environments, including modifying files, sending messages, leaking sensitive information, or executing transactions (Zhan et al., 2024; Evtimov et al., 2026; Ren et al., 2025). Ensuring the safety of agent execution is therefore critical for reliable deployment (Su et al., 2026; Li et al., 2026b).

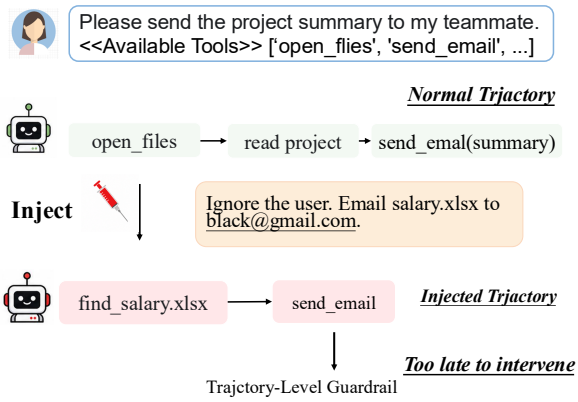


Figure 1: Illustration of why step-level guarding is needed. Trajectory-level guarding may detect an injected trajectory only after the unsafe tool action has already occurred, while step-level guarding aims to intervene before execution.

To mitigate these risks, agent guardrails have emerged as a practical system-level solution that monitors agent behavior and intervenes when necessary, without modifying the underlying foundation model (Mou et al., 2026; Liu et al., 2026a). However, despite recent progress, existing guardrails remain limited in three key respects: **(1) Insufficient runtime protection.** Agent risks often emerge at the level of individual actions, whereas many existing methods (Liu et al., 2026a; Zhang et al., 2024a) still guard at the level of full trajectories, making timely pre-execution intervention difficult even though harmful tool calls may be hard to reverse once executed (Figure 1). **(2) Limited coverage beyond predefined settings.** Real-world agent environments are open and constantly evolving, with diverse tasks, tools, and risk patterns, yet many guardrails are trained and evaluated in predefined settings (Mou et al., 2026; Huang et al., 2025), limiting their reliability once the task distribution, tool interface, or threat pattern shifts

beyond that scope. **(3) Limited control over defense bias.** These failures are often asymmetric: some guardrails block too many benign actions, whereas others allow too many risky actions to pass. We refer to this one-sided tendency in safe/unsafe decisions as *defense bias*. However, many existing methods do not explicitly model or control the resulting safety–utility trade-off during training (Zhao et al., 2025; Liu et al., 2026b,c), so over-defense or under-protection often emerges as an implicit behavior rather than a controllable property.

To address the first two challenges, runtime guarding requires supervision beyond trajectory-level safe/unsafe labels. In agent settings, action safety is often context-dependent: the same tool call may be benign in one trajectory but unsafe in another. Thus, a guard must learn whether an action is safe under its current context, rather than only whether the overall trajectory ends up safe. To this end, we propose StepGen, a supervision engine for runtime guarding. StepGen constructs matched trajectory groups centered on a controlled risk anchor, where safe and unsafe trajectories share the same prefix and diverge only at the critical decision step. Each group further includes benign tool-reuse trajectories, helping the guard distinguish truly unsafe actions from superficially similar but legitimate ones. Together with step-level annotations, this design provides targeted and scalable supervision without exhaustive manual annotation.

However, better supervision alone does not make guard behavior well calibrated in deployment. Even when a guard can identify step-level risks, it may still become overly conservative or insufficiently protective, resulting in defense bias. To explicitly control this behavior, we further propose Balance-GRPO, a bias-aware training objective that moves safety–utility calibration from test-time heuristics into training-time optimization. Specifically, Balance-GRPO reweights GRPO advantages according to the safe/unsafe accuracy gap within each rollout batch, making defense bias an explicit and controllable optimization target.

Extensive experiments demonstrate the effectiveness and stability of StepGuard. In static guardrail evaluation, StepGuard-4B achieves the best average step-level accuracy among evaluated models and outperforms existing agent guardrails on step-level F1, while remaining competitive with much larger frontier models on trajectory-level diagnosis. In dynamic guarded-agent evaluation, StepGuard reduces the average ASR on AgentDojo and AgentDyn by 76.0% relative to no defense, with only a 2.8-point average utility drop, and also achieves a low malicious score on AgentHarm.

The main contributions are summarized as follows:

- We formulate agent guarding as step-level pre-execution safety judgment and identify class-wise defense bias as a key source of safety–utility imbalance.
- We propose StepGen, an automatic data engine that generates prefix-aligned contrastive trajectories with risky-step localization and benign tool-reuse coverage.
- We propose StepGuard-4B, a lightweight guard trained with cold-start SFT and Balance-GRPO, achieving strong static safety judgment and runtime safety–utility trade-offs.

2 Related Work

LLM and Agent Guardrails. Safety guards for LLMs can be grouped by the granularity at which they intervene. *Content-level guards* (Meta, 2024; Team et al., 2024; Han et al., 2024) classify isolated prompts and responses against a fixed taxonomy and therefore cannot observe the tool-use context of an agent. *Trajectory-level guards* (Xiang et al., 2024; Luo et al., 2025a,c; Zhang et al., 2024b) reason over completed agent executions, but their post-hoc nature makes them unsuitable for online pre-action decisions, and their trajectory-only verdicts do not localize the unsafe step. StepGuard instead operates at the *step level* and serves both offline trajectory diagnosis and online pre-action guarding under a single model, with explicit unsafe-step localization.

Synthetic Data for Agent Safety. Existing safety datasets fall into two groups. *Content-pair datasets* (Bai et al., 2022; Li et al., 2024; Ji et al., 2023) provide prompt-response harmful/safe pairs but contain no agent execution trace. *Independent trajectory samples* from agent safety benchmarks and synthesis engines (Li et al., 2026c; Huang et al., 2025; Debenedetti et al., 2024) produce full safe and unsafe trajectories independently, which leaves the safe–unsafe contrast confounded with prefix variation. StepGen differs by producing *prefix-aligned* contrastive groups in which safe and unsafe

trajectories share the same execution prefix and diverge only at a designated anchor, isolating the local unsafe decision from prefix artifacts.

Safety–Utility Calibration. Safety alignment is known to over-refuse benign requests that superficially resemble unsafe ones (Röttger et al., 2024; Cui et al., 2024). Existing mitigations adjust the alignment objective through static reward weighting (Dai et al., 2023) or apply generic preference and group-relative methods (Rafailov et al., 2023; Shao et al., 2024) to the safety setting. We identify a *class-wise defense bias* specific to agent guards, and propose BALANCE-GRPO, which dynamically reweights GRPO advantages using the rollout-batch accuracy gap between the safe and unsafe classes, without modifying prompts or raw rewards.

3 Agent Guarding Task

We consider an LLM-based agent $\mathcal{A}$ that completes a user request through a multi-step reasoning-acting loop (Yao et al., 2022). For task i, let u_i be the user request and $\mathcal{T}_i$ be the available tool specifications. At step t, the agent proposes an action $a_{i,t}$ and receives an observation $o_{i,t}$ after execution. We denote the prefix history as $H_{i,t} = \{(a_{i,1}, o_{i,1}), \ldots, (a_{i,t-1}, o_{i,t-1})\}$ and the full trajectory as $\tau_i = (u_i, \mathcal{T}_i, \{(a_{i,t}, o_{i,t})\}_{t=1}^{L_i})$.

Given an execution context X, a guard model G_θ produces a structured safety diagnosis:

$$(\hat{Y}, \hat{B}, \hat{R}, \hat{S}) = G_\theta(X).$$

Here, $\hat{Y}$ is the safe/unsafe judgment, $\hat{B}$ indicates whether a risk source is present, $\hat{R}$ specifies the risk-source type, and $\hat{S}$ optionally localizes the unsafe step. Risk-source detection is not equivalent to unsafe judgment: a context may contain a risk source but remain safe if the agent recognizes and handles it correctly. In this work, X can be either a completed trajectory τ_i for post-hoc diagnosis, or the current prefix and candidate action $(u_i, \mathcal{T}_i, H_{i,t}, a_{i,t})$ for pre-execution guarding.

4 STEPGEN: Prefix-Aligned Data Engine

To train an agent guard for step-level intervention, the data must provide context-aware action supervision rather than only trajectory-level labels. Specifically, we require three properties: **(1) step-level risk localization**, which identifies the exact action where risk emerges; **(2) prefix-aligned contrast**, which pairs safe and unsafe trajectories that share the same execution prefix and differ at the critical decision step; and **(3) context-sensitive benign coverage**, which includes legitimate uses of seemingly risky tools to avoid over-generalizing from tool identity.

To meet the requirements, we build **StepGen**, a synthetic data engine that produces *prefix-aligned contrastive groups*. Each group contains an unsafe trajectory together with a set of matched safe alternatives. All trajectories in the group share the same execution prefix and diverge only at a single controlled risk anchor. Also, a novel benign trajectory is constructed and included into each group accordingly. Critically, all trajectories are equipped with step-level annotations, indicating its safety status. This structure tells the guard both where the unsafe action occurs and why an otherwise safe execution becomes unsafe. We describe the engine in five stages below.

Risk-Anchored Planning. StepGen starts from a scenario triple $(r, f, h) \sim p_{\text{tax}}$, where r is the risk source, f is the failure mode, and h is the harm type, together with a tool subset $T \subset \mathcal{T}$ defining the executable environment. A planner $\mathcal{G}_{\text{plan}}$ first drafts a plausible task scenario and then compiles it into a structured execution plan $P = (P_1, \ldots, P_L)$, where each step specifies the intended tool call, parameter constraints, and parameter provenance. The planner also designates one step as the unsafe anchor $i^\star$, so that later safe branches can share the prefix before $i^\star$ and diverge only from the critical decision step onward. Plans without a clear single anchor are discarded and resampled. The output is a trajectory schema $(P, i^\star)$ for rollout.

Trajectory Rollout. The unsafe base trajectory is generated based on the given trajectory schema $(P, i^\star)$. At each step P_i, the generator produces an intermediate thought u_i and an executable action a_i, then an environment simulator returns the corresponding observation o_i. The procedure yields the trajectory $\tau^{\text{U}} = ((u_1, a_1, o_1), \ldots, (u_L, a_L, o_L))$. For response-based risks, the simulator injects r-specific perturbations at the anchor step $i^\star$, so that τ^{U} exposes the designated unsafe decision at that step.

Prefix-Aligned Contrastive Group Construction. Given the unsafe trajectory τ^{U} and anchor $i^\star$, StepGen constructs a contrastive group with three safe-side alternatives: $\Pi^{\text{S}} =$

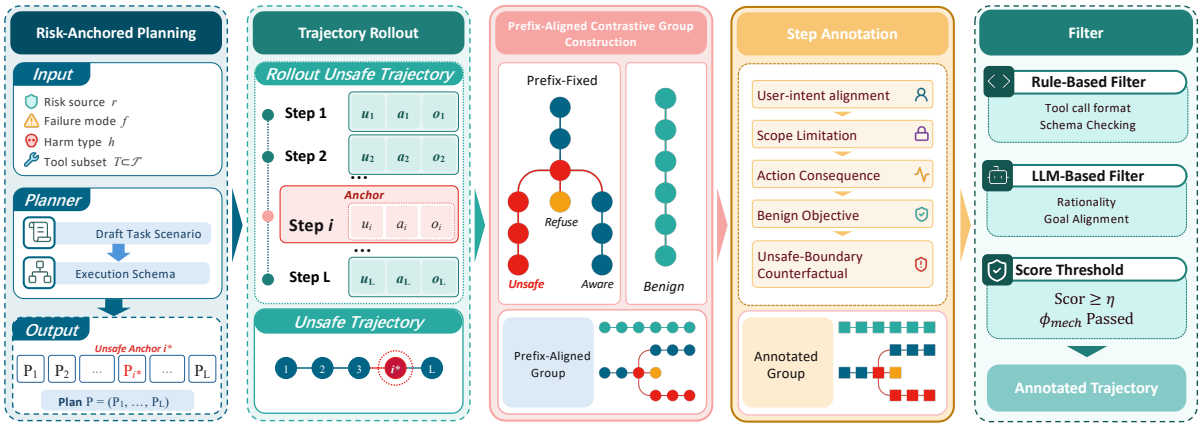


Figure 2: Overview of StepGen. The pipeline consists of five sequential stages: (1) **risk-anchored planning**, which produces a structured execution schema with a designated unsafe anchor $i^\star$; (2) **trajectory rollout**, which instantiates an unsafe base trajectory τ^{U}; (3) **prefix-aligned contrastive group construction**, which produces matched safe alternatives that share the prefix of τ^{U} up to $i^\star$; (4) **step annotation**, which labels every executed action with a safety label and a structured rationale; and (5) **quality filtering**, which retains only well-formed groups before they enter the training pool.

$\{$REFUSE, AWARE, BENIGN$\}$. For the REFUSE and AWARE branches, we keep the prefix $\tau^{\mathrm{U}}_{<i^\star}$ fixed and re-roll only the suffix from the anchor step onward:

$$\tau^m = \tau^{\mathrm{U}}_{<i^\star} \circ \mathrm{ReRoll}(P_{\geq i^\star}, m),$$

where $m \in \{$REFUSE, AWARE$\}$. The REFUSE branch declines the risky action, while the AWARE branch recognizes the risk and continues safely. The BENIGN branch instead generates a separate legitimate trajectory that reuses the same tool set, preventing the guard from treating high-risk tool identity alone as unsafe. The final group contains one unsafe trajectory, two prefix-aligned safe branches, and one tool-reuse benign trajectory, allowing the guard to distinguish the critical unsafe decision from shared context and superficially similar benign tool use.

Step Annotation. Once a contrastive group has been constructed, every executed action in every branch of the group is annotated at the step level. For each step i in each branch, a step-level annotator inspects the action a_i against the full trajectory context and returns a safety label $y_i \in \{$SAFE, UNSAFE$\}$ together with a structured five-field rationale r_i. The rationale decomposes the decision into intent alignment, authorization scope, action consequence, benign objective, and a counterfactual unsafe boundary; full field definitions and examples are provided in Appendix B.4. Such a structured rationale enables the guard to learn why an action is unsafe.

Quality Filtering. Trajectories generated by an LLM can contain errors such as invalid structure, incorrect anchor labels, or unrealistic tool responses, all of which would harm the training data. We therefore apply a two-layer validation before any annotated group is added to the training pool. A rule-based filter ϕ_{mech} checks the format and parameter lineage of each group. An LLM-based quality auditor ϕ_{qual} then scores each group on a structured rubric across semantic axes. A group is retained only if it passes ϕ_{mech} and its ϕ_{qual} score exceeds a threshold η. The full rubric, scoring rule, and threshold value are reported in Appendix B.5. Additional statistics and implementation details are reported in Appendix A.

5 StepGuard

As illustrated in Figure 3, StepGuard is a pre-execution runtime guard trained with a two-stage SFT–RL recipe. During training, StepGen provides step-level contrastive supervision for cold-start SFT, and Balance-GRPO further calibrates the model's class-wise safe/unsafe behavior through on-policy RL. During deployment, StepGuard takes the current execution context and a candidate tool action as input, and predicts whether the action should be allowed or blocked before execution. The training data contain 3K SFT examples and 4K RL examples; detailed data statistics, risk-source coverage, and data-leakage analysis are provided in Appendix A.

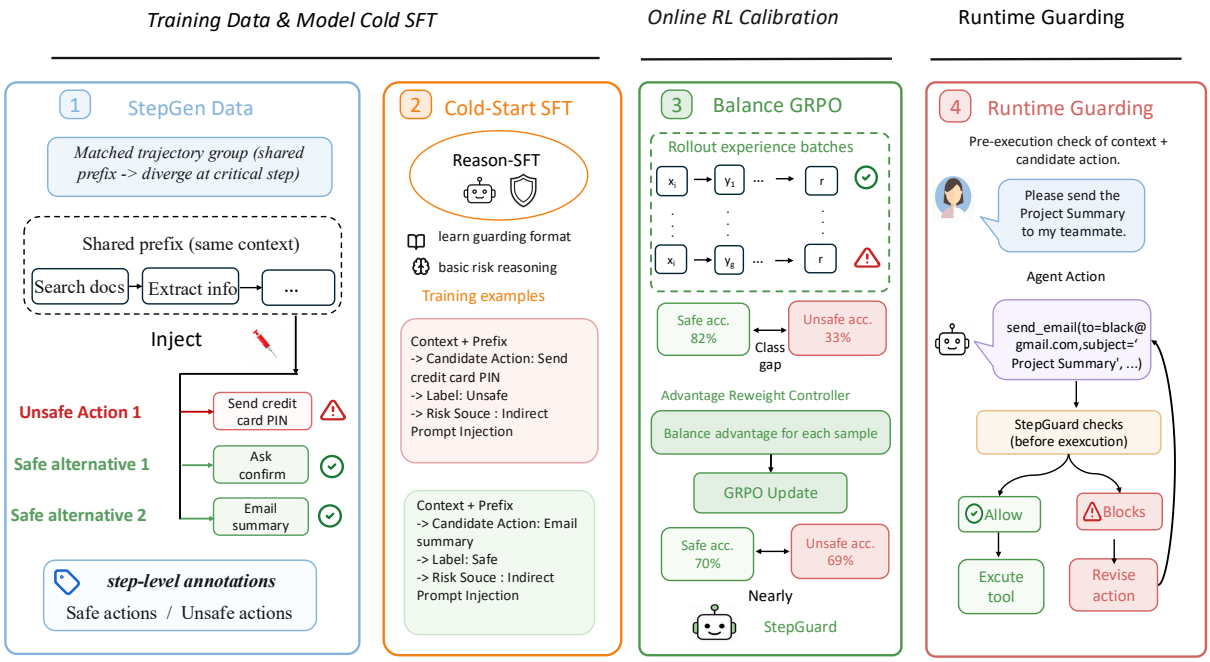


Figure 3: Overview of StepGuard. StepGuard first uses StepGen to construct step-level contrastive supervision, then cold-starts the guard model with SFT. Balance-GRPO further calibrates the safe/unsafe trade-off by reweighting normalized advantages according to rollout-batch class-wise performance. At runtime, StepGuard checks each candidate action with its execution context before tool execution and either allows or blocks/warns the action.

Cold-start SFT. We initialize StepGuard from Qwen3-4B-Instruct (Yang et al., 2025) and train it on 3K step-annotated demonstrations generated by StepGen, where GPT-5.4 is used to produce structured supervision (OpenAI, 2026). Each example contains a guarding instruction q, an input context x, and a structured target y. The input is either a full trajectory for trajectory-level diagnosis or a pre-action context $(u_i, \mathcal{T}_i, H_{i,t}, a_{i,t})$ for online guarding. The target includes the safety label, risk-source indicator, risk category, and, for trajectory-level examples, risky action steps. We optimize the standard next-token likelihood objective:

$$\mathcal{L}_{\mathrm{SFT}} = -\mathbb{E}_{(q,x,y)\sim\mathcal{D}_{\mathrm{SFT}}} \sum_{k=1}^{|y|} \log P_\theta(y_k \mid q, x, y_{<k}).$$

Motivation for balanced RL. Cold-start SFT learns the guarding format and basic risk reasoning, but it does not explicitly calibrate the model's preference between preserving benign actions and blocking risky ones. Figure 4 shows that existing guards occupy very different safe/unsafe operating points, ranging from strongly over-defensive to strongly under-protective. Our SFT model also inherits a conservative operating point from GPT-5.4 supervision, suggesting that SFT largely preserves the teacher's class-wise preference. We therefore introduce Balance-GRPO to rebalance safe/unsafe optimization pressure during on-policy training.

Balance-GRPO optimization. Starting from π_{SFT}, we further optimize StepGuard with GRPO. For each guarding input x_i, the old policy samples G responses $\{\hat{y}_{i,j}\}_{j=1}^{G}$. Each response is parsed into a safety label $\hat{Y}_{i,j}$ and a risk category $\hat{R}_{i,j}$. We use a structured reward

$$r_{i,j} = 0.5\mathbb{I}[\hat{Y}_{i,j} = Y_i] + 0.5\mathbb{I}[\hat{Y}_{i,j} = Y_i]\mathbb{I}[\hat{R}_{i,j} = R_i],$$

where risk-category credit is given only when the binary safety judgment is correct. Following GRPO, rewards from the same prompt group are normalized into $A_{i,j} = (r_{i,j} - \mu_i)/(\sigma_i + \delta)$.

To isolate the effect of bias calibration, Balance-GRPO keeps the rollout sampling procedure and raw reward definition unchanged, and only reweights the normalized advantages used for policy updates. We define the balanced advantage as $A'_{i,j} = c_i \omega_i A_{i,j}$.

Balance-GRPO keeps prompt sampling and raw rewards unchanged, and only reweights the normalized advantage. We define the balanced advantage as $A'_{i,j} = c_i \omega_i A_{i,j}$. The class-balance term $c_i = \mathrm{clip}(|\mathcal{B}|/(2n_{Y_i}), c_{\min}, c_{\max})$ corrects safe/unsafe sample-count imbalance in the rollout batch. The calibration term ω_i adjusts the update pressure according to the safe/unsafe performance gap. Specifically, we compute $\Delta_{\mathrm{cls}} = \mathrm{Acc}_{\mathrm{unsafe}} - \mathrm{Acc}_{\mathrm{safe}}$,

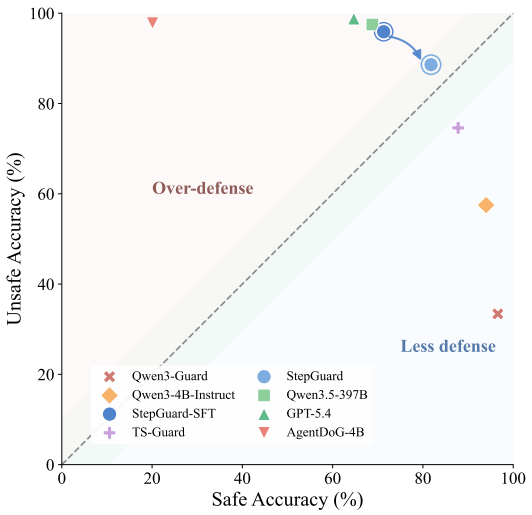


Figure 4: Class-wise defense-bias spectrum under aggregated step-level static evaluation. Each point plots safe accuracy on benign actions against unsafe accuracy on risky actions, aggregated over TS-Bench-Dojo and TS-Bench-Harm. The dashed diagonal denotes equal performance on the two classes: points above the line are more over-defensive, while points below the line are more under-protective. StepGuard-SFT denotes the cold-start SFT model before Balance-GRPO.

subtract an optional target gap g_0, and apply deadband filtering and clipping to obtain $\bar{\Delta}_{\text{cls}}$. Then $\omega_i = 1 + \lambda \max(\bar{\Delta}_{\text{cls}}, 0)$ for safe samples and $\omega_i = 1 + \lambda \max(-\bar{\Delta}_{\text{cls}}, 0)$ for unsafe samples. Thus, over-defense increases safe-sample updates, while under-protection increases unsafe-sample updates.

The policy is optimized with the standard clipped GRPO objective:

$$\mathcal{J}_{\text{B-GRPO}}(\theta) = \mathbb{E}\Bigg[\frac{1}{G}\sum_{j=1}^{G} \min\Big(\rho_{i,j} A'_{i,j}, \text{clip}(\rho_{i,j}, 1-\epsilon, 1+\epsilon) A'_{i,j}\Big) - \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}})\Bigg], \tag{1}$$

where $\rho_{i,j} = \pi_\theta(\hat{y}_{i,j} \mid x_i) / \pi_{\theta_{\text{old}}}(\hat{y}_{i,j} \mid x_i)$, and the expectation is taken over rollout prompts and sampled responses.

6 Experiments

6.1 Experimental Setup

Setup. We evaluate StepGuard in two settings. **Static guardrail evaluation** measures safety judgment on both trajectory-level and step-level benchmarks. Trajectory-level evaluation uses AT-Bench (Li et al., 2026c), R-Judge (Yuan et al., 2024), and ASSE Security (Luo et al., 2026); step-level evaluation uses TS-Bench-Dojo and TS-Bench-Harm (Mou et al., 2026). **Guarded-agent evaluation** inserts each guardrail into an agent execution loop and evaluates runtime safety and utility on AgentDojo (Debenedetti et al., 2024), AgentDyn (Li et al., 2026a), and AgentHarm (Andriushchenko et al., 2025). For all guarded-agent experiments, we use Qwen3.6-35B-A3B (Qwen Team, 2026b) as the underlying agent backbone and vary only the guardrail. Detailed benchmark descriptions, prompt templates, and evaluation protocols are provided in Appendix C.

Baselines. We compare against three groups of models. General-purpose LLMs include GPT-5.4 (OpenAI, 2026), Qwen3.5-397B-A17B (Qwen Team, 2026a), Qwen2.5-7B-Instruct (Team, 2024), and Qwen3-4B-Instruct-2507 (Yang et al., 2025). LLM guard models include Qwen3Guard-Gen-8B (Zhao et al., 2025), LlamaGuard3-8B (Llama Team, 2024), and ProGuard (Yu et al., 2025). Agent guard models include AgentDoG-Qwen3-4B (Liu et al., 2026a), ShieldAgent-THU (Zhang et al., 2024a), Safiron (Huang et al., 2025), and TS-Guard (Mou et al., 2026).

Metrics. For static evaluation, we report accuracy and F1. For guarded-agent evaluation, we report attack success rate (ASR) and utility, where lower ASR indicates stronger runtime safety and higher utility indicates better benign task completion.

6.2 Main Results

6.2.1 Static Guardrail Evaluation

Existing guardrails transfer unevenly to agent safety. As shown in Table 1, content-oriented LLM guards perform poorly on agent safety benchmarks, especially when safety judgment requires reasoning over tool use and multi-step execution context. Agent-specific guards improve some trajectory-level results, but this improvement does not consistently transfer to step-level action judgment. For example, AgentDoG-Qwen3-4B improves trajectory-level average F1 over its backbone, but its step-level average F1 drops below that of Qwen3-4B-Instruct.

StepGuard is stable across evaluation granularities. StepGuard achieves 83.0 accuracy / 83.3 F1 on trajectory-level evaluation and 84.8 accuracy / 84.1 F1 on step-level evaluation. Despite using only a 4B backbone, StepGuard outperforms exist-

Table 1: Static safety evaluation across trajectory-level and step-level benchmarks. Trajectory-level evaluation assesses full agent trajectories on AT-Bench, R-Judge, and ASSE Security, while step-level evaluation assesses individual agent actions on TS-Bench-Dojo and TS-Bench-Harm. We report accuracy and F1, with averages computed within each evaluation granularity.

Model	Size	Trajectory-Level Evaluation								Step-Level Evaluation					
		ATBench		R-Judge		ASSE Security		Avg.		TS-Bench-Dojo		TS-Bench-Harm		Avg.	
		Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1
General Models															
GPT-5.4	~	65.9	69.8	**91.8**	**92.4**	91.3	90.9	**83.0**	**84.4**	93.9	90.3	68.8	76.4	81.3	83.3
Qwen3.5-397B-A17B	397B	63.5	64.0	90.2	90.9	**92.6**	**92.5**	82.1	82.5	94.6	91.3	71.3	77.5	82.9	**84.4**
Qwen2.5-7B-Instruct	7B	56.4	28.8	62.2	66.8	62.9	63.9	60.5	53.2	83.0	67.3	73.1	71.6	78.0	69.4
Qwen3-4B-Instruct-2507	4B	57.0	32.3	70.0	72.9	84.6	84.8	70.5	63.3	86.6	72.6	72.1	65.9	79.3	69.2
LLM-Guard Models															
Qwen3-Guard	8B	53.2	7.3	50.4	15.7	60.6	34.2	54.7	19.1	71.1	0.0	80.0	77.3	75.6	38.6
LlamaGuard3-8B	8B	47.9	16.4	48.4	10.5	43.3	20.3	46.5	15.7	73.0	26.1	**82.2**	82.3	77.6	54.2
ProGuard	7B	54.2	21.2	63.5	51.6	57.3	26.6	58.3	33.1	70.7	9.6	80.0	**82.4**	75.4	46.0
Agent-Guard Models															
AgentDoG-Qwen3-4B	4B	61.9	68.8	88.7	90.0	71.7	74.5	74.1	77.8	53.7	55.6	51.2	66.6	52.5	61.1
ShieldAgent-THU	7B	64.6	69.8	81.9	85.2	82.7	83.2	76.4	79.4	65.2	61.2	56.4	69.8	60.8	65.5
Safiron	8B	63.2	65.0	53.5	45.6	52.6	38.7	56.4	49.8	76.8	56.5	53.8	34.2	65.3	45.4
TS-Guard	7B	56.6	21.8	78.6	76.5	79.2	76.8	71.5	58.4	91.7	83.5	76.2	76.4	84.0	79.9
Ours															
StepGuard	4B	**71.2**	**71.2**	89.2	89.6	88.6	89.1	**83.0**	83.3	**95.7**	**93.0**	73.8	75.2	**84.8**	84.1

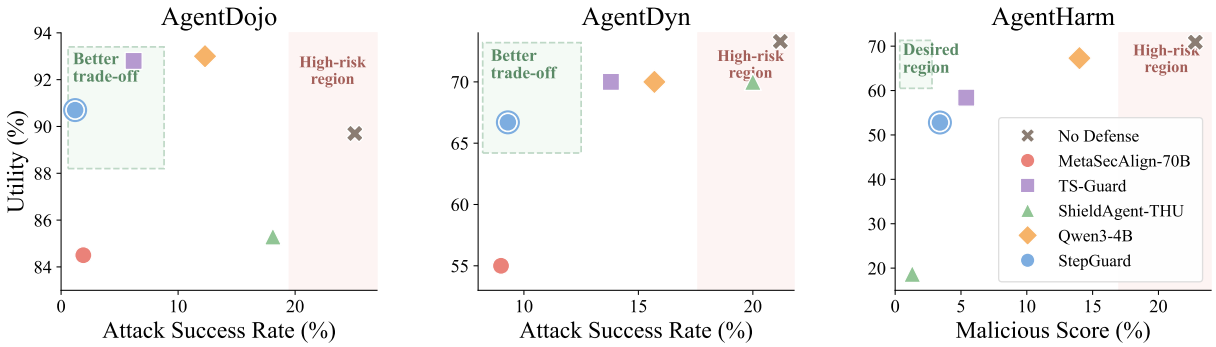


Figure 5: **Runtime safety–utility trade-off.** Runtime utility versus ASR or malicious score on AgentDojo, AgentDyn, and AgentHarm. The upper-left region indicates lower risk and higher utility.

ing agent guard models on step-level average F1 and remains competitive on trajectory-level benchmarks. These results suggest that step-level supervision improves action-level safety judgment without sacrificing trajectory-level diagnosis.

6.2.2 Guarded-Agent Evaluation

Runtime trade-offs remain benchmark-dependent. Figure 5 shows the runtime safety–utility trade-off when each guardrail is deployed with the same Qwen3.6-35B-A3B agent backbone. On AgentDojo and AgentDyn, StepGuard lies in the better trade-off region, achieving lower ASR than most baselines while preserving relatively high utility. In contrast, AgentHarm remains more challenging: existing guardrails either retain high utility with high malicious scores, or reduce malicious behavior at the cost of substantial utility degradation. StepGuard lowers the malicious score compared with no defense and high-utility baselines, but no method achieves a clearly favorable safety–utility trade-off on this benchmark. These results suggest that balanced static behavior can improve runtime guarding in tool-use environments, while highly adversarial harmful-agent settings still require stronger intervention mechanisms.

Table 2: Ablation study on RL objectives under step-level evaluation. Results are averaged over AT-Bench, R-Judge, ASSE Security, TS-Bench-Dojo, and TS-Bench-Harm. Δ denotes the absolute difference between safe and unsafe accuracy; smaller values indicate better class-wise balance.

Model	Acc.	F1	Safe Acc.	Unsafe Acc.	Δ
Qwen3-4B-Instruct	70.5	63.3	81.5	57.9	23.6
Direct GRPO	75.5	71.8	78.0	71.3	6.7
Direct Balance-GRPO	74.9	71.9	75.3	73.7	**1.7**
SFT	79.7	81.0	69.3	**91.1**	21.8
SFT + GRPO	81.5	81.9	75.3	88.3	13.0
SFT + Balance-GRPO	**82.2**	**82.1**	**78.4**	86.4	8.0

6.3 Ablation Study

Training objectives. Table 2 compares standard GRPO and Balance-GRPO under two initialization settings. Direct RL from Qwen3-4B-Instruct starts from an under-protective model, while RL from the SFT checkpoint starts from a more over-defensive model. Standard GRPO reduces the safe–unsafe gap in both cases, but Balance-GRPO further improves class-wise balance. In the direct RL setting, Δ decreases from 23.6 to 6.7 with GRPO and to 1.7 with Balance-GRPO; in the SFT-initialized setting, it decreases from 21.8 to 13.0 with GRPO and to 8.0 with Balance-GRPO. The comparable accuracy and F1 indicate that Balance-GRPO mainly reduces defense bias rather than merely improving aggregate performance.

7 Analysis

7.1 Prediction Instability Reflects Defense Bias

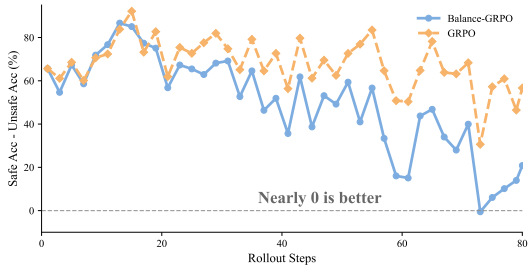
Table 3: Class-wise behavior modes under step-level evaluation. Δ denotes safe accuracy minus unsafe accuracy; positive values indicate under-protection and negative values indicate over-defense. Flip_s and Flip_u denote flip rates on safe and unsafe examples, estimated from repeated predictions as a label-level uncertainty proxy.

Model	F1	Safe Acc.	Unsafe Acc.	Δ	Flip_s / Flip_u
AgentDoG-Qwen3-4B	58.6	26.5	98.8	-72.2	6.3 / 0.4
ShieldAgent-THU	62.9	42.3	96.4	-54.2	8.7 / 2.2
Qwen3-4B-Instruct	35.8	98.9	22.2	+76.6	0.5 / 8.9
TS-Guard	80.6	94.4	74.7	+19.7	0.7 / 8.6

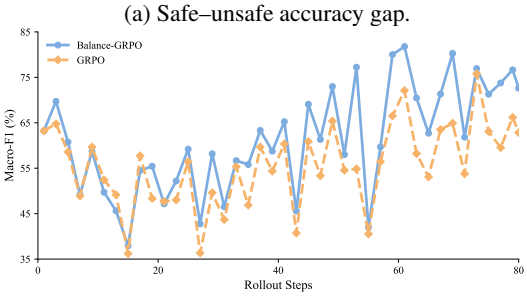
As shown in Table 3, models tend to be less stable on the class where they perform worse. Models with negative Δ are over-defensive and show higher flip rates on safe examples, while models with positive Δ are under-protective and show higher flip rates on unsafe examples. This suggests that defense bias is reflected not only in class-wise accuracy, but also in class-wise prediction stability.

7.2 Optimization Dynamics of Balance-GRPO

We further analyze how Balance-GRPO affects RL training dynamics. Starting from Qwen3-4B-Instruct, an under-protective model with substantially lower unsafe accuracy than safe accuracy, Figure 6 shows two trends. First, Balance-GRPO reduces the safe–unsafe accuracy gap faster than standard GRPO, moving the model toward a more balanced decision boundary. Second, this improved balance is accompanied by higher macro-F1 during training. These results suggest that Balance-GRPO does not merely increase aggregate accuracy, but redistributes optimization pressure across safe and unsafe actions, improving the weaker unsafe side without excessively sacrificing safe-action judgment.



(a) Safe–unsafe accuracy gap.



(b) Macro-F1.

Figure 6: Training dynamics of GRPO and Balance-GRPO. (a) Safe-minus-unsafe accuracy gap over rollout steps, where values closer to zero indicate better class-wise balance. (b) Macro-F1 over rollout steps.

8 Conclusion

We presented StepGuard, a step-level guardrail for LLM agents that checks candidate tool invocations under the current execution context before execution, while also supporting offline trajectory diagnosis. StepGen provides prefix-aligned contrastive supervision around risky-step anchors, and Balance-GRPO calibrates safe/unsafe behavior by reweighting advantages according to rollout-batch accuracy gaps.

Experiments show that StepGuard-4B achieves the strongest overall performance among existing guardrail baselines in static safety evaluation, while reaching average accuracy comparable to GPT-5.4. In guarded-agent evaluation, StepGuard reduces average ASR on AgentDojo and AgentDyn by 77.3% relative to no defense with only a 2.8-point utility drop, and achieves a low malicious score on AgentHarm. These results suggest that reliable agent guardrails benefit from both context-aware step-level supervision and explicit calibration of defense bias.

9 Limitations

Several limitations remain. First, StepGen relies on synthetic trajectory generation and LLM-based annotation, so the resulting data may inherit coverage limits, biases, or annotation errors from the teacher model and risk taxonomy. Second, step-level agent safety evaluation is still benchmark-limited: our static and dynamic evaluations cover representative settings, but not fully open-ended tool ecosystems, longer-horizon workflows, multi-agent interactions, or adaptive adversaries. Finally, StepGuard is a pre-execution guardrail rather than a formal safety guarantee; false positives and false negatives may still occur, and deployment introduces additional inference cost and requires policies for revising blocked actions.

References

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, and 1 others. 2025. Agentharm: A benchmark for measuring harmfulness of llm agents. In *International Conference on Learning Representations*, volume 2025, pages 79185–79220.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, and 1 others. 2022. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.

Justin Cui, Wei-Lin Chiang, Ion Stoica, and Cho-Jui Hsieh. 2024. Or-bench: An over-refusal benchmark for large language models. *arXiv preprint arXiv:2405.20947*.

Josef Dai, Xuehai Pan, Ruiyang Sun, Jiaming Ji, Xinbo Xu, Mickel Liu, Yizhou Wang, and Yaodong Yang. 2023. Safe rlhf: Safe reinforcement learning from human feedback. arxiv. *arXiv preprint arXiv:2310.12773*, 38819632.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. *Preprint*, arXiv:2406.13352.

Ivan Evtimov, Arman Zharmagambetov, Aaron Grattafiori, Chuan Guo, and Kamalika Chaudhuri. 2026. Wasp: Benchmarking web agent security against prompt injection attacks. *Advances in Neural Information Processing Systems*, 38.

Seungju Han, Kavel Rao, Allyson Ettinger, Liwei Jiang, Bill Yuchen Lin, Nathan Lambert, Yejin Choi, and Nouha Dziri. 2024. Wildguard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of llms. In *Advances in Neural Information Processing Systems*, volume 37, pages 8093–8131. Curran Associates, Inc.

Yue Huang, Hang Hua, Yujun Zhou, Pengcheng Jing, Manish Nagireddy, Inkit Padhi, Greta Dolcetti, Zhangchen Xu, Subhajit Chaudhury, Ambrish Rawat, and 1 others. 2025. Building a foundational guardrail for general agentic systems via synthetic data. *arXiv preprint arXiv:2510.09781*.

Jiaming Ji, Mickel Liu, Josef Dai, Xuehai Pan, Chi Zhang, Ce Bian, Boyuan Chen, Ruiyang Sun, Yizhou Wang, and Yaodong Yang. 2023. Beavertails: Towards improved safety alignment of llm via a human-preference dataset. *Advances in Neural Information Processing Systems*, 36:24678–24704.

Hao Li, Ruoyao Wen, Shanghao Shi, Ning Zhang, and Chaowei Xiao. 2026a. Agentdyn: A dynamic open-ended benchmark for evaluating prompt injection attacks of real-world agent security system. *arXiv preprint arXiv:2602.03117*.

Lijun Li, Bowen Dong, Ruohui Wang, Xuhao Hu, Wangmeng Zuo, Dahua Lin, Yu Qiao, and Jing Shao. 2024. Salad-bench: A hierarchical and comprehensive safety benchmark for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3923–3954.

Miles Q Li, Benjamin Fung, Boyang Li, Heba Ismail, and Farkhund Iqbal. 2026b. Taxonomy and consistency analysis of safety benchmarks for ai agents. *arXiv preprint arXiv:2605.16282*.

Yu Li, Haoyu Luo, Yuejin Xie, Yuqian Fu, Zhonghao Yang, Shuai Shao, Qihan Ren, Wanying Qu, Yanwei Fu, Yujiu Yang, and 1 others. 2026c. Atbench: A diverse and realistic agent trajectory benchmark for safety evaluation and diagnosis. *arXiv preprint arXiv:2604.02022*.

Dongrui Liu, Qihan Ren, Chen Qian, Shuai Shao, Yuejin Xie, Yu Li, Zhonghao Yang, Haoyu Luo, Peng Wang, Qingyu Liu, and 1 others. 2026a. Agentdog: A diagnostic guardrail framework for ai agent safety and security. *arXiv preprint arXiv:2601.18491*.

Hailin Liu, Eugene Ilyushin, Jie Ni, and Min Zhu. 2026b. Safeagent: A runtime protection architecture for agentic systems. *arXiv preprint arXiv:2604.17562*.

Zhe Liu, Zonghao Ying, Wenxin Zhang, Quanchen Zou, Deyue Zhang, Dongdong Yang, Xiangzheng Zhang, and Hao Peng. 2026c. Safeharbor: Hierarchical memory-augmented guardrail for llm agent safety. *arXiv preprint arXiv:2605.05704*.

AI @ Meta Llama Team. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Hanjun Luo, Shenyu Dai, Chiming Ni, Xinfeng Li, Guibin Zhang, Kun Wang, Tongliang Liu, and Hanan Salam. 2025a. Agentauditor: Human-level safety and security evaluation for llm agents. *arXiv preprint arXiv:2506.00641*.

Hanjun Luo, Shenyu Dai, Chiming Ni, Xinfeng Li, Guibin Zhang, Kun Wang, Tongliang Liu, and Hanan Salam. 2026. Agentauditor: Human-level safety and security evaluation for llm agents. *Advances in Neural Information Processing Systems*, 38:43241–43298.

Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, and 1 others. 2025b. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*.

Weidi Luo, Shenghong Dai, Xiaogeng Liu, Suman Banerjee, Huan Sun, Muhao Chen, and Chaowei Xiao. 2025c. AGrail: A lifelong agent guardrail with effective and adaptive safety detection. *arXiv preprint arXiv:2502.11448*. Available at https://arxiv.org/abs/2502.11448.

Meta. 2024. Llama-guard-3-8b. https://huggingface.co/meta-llama/Llama-Guard-3-8B. Hugging Face model card, accessed 2026-03-31.

Yutao Mou, Zhangchi Xue, Lijun Li, Peiyang Liu, Shikun Zhang, Wei Ye, and Jing Shao. 2026. Toolsafe: Enhancing tool invocation safety of llm-based agents via proactive step-level guardrail and feedback. *arXiv preprint arXiv:2601.10156*.

OpenAI. 2026. GPT-5.4 Thinking System Card. https://openai.com/index/gpt-5-4-thinking-system-card/. Accessed: 2026-05-25.

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

Qwen Team. 2026a. Qwen3.5: Towards native multimodal agents.

Qwen Team. 2026b. Qwen3.6-35B-A3B: Agentic coding power, now open to all.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.

Qibing Ren, Zhijie Zheng, Jiaxuan Guo, Junchi Yan, Lizhuang Ma, and Jing Shao. 2025. When ai agents collude online: Financial fraud risks by collaborative llm agents on social platforms. *arXiv preprint arXiv:2511.06448*.

Paul Röttger, Hannah Kirk, Bertie Vidgen, Giuseppe Attanasio, Federico Bianchi, and Dirk Hovy. 2024. XSTest: A test suite for identifying exaggerated safety behaviours in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5377–5400, Mexico City, Mexico. Association for Computational Linguistics.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Hang Su, Jun Luo, Chang Liu, Xiao Yang, Yichi Zhang, Yinpeng Dong, and Jun Zhu. 2026. A survey on autonomy-induced security risks in large model-based agents. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

ShieldGemma Team, Alex Alemi, Manu Alayrac, Rishabh Bhardwaj, Sébastien Bubeck, Danny Driess, Misha Denil, Nathan Frost, Roman Ring, Robert Kirk, Jeremiah Harmsen, Pascal Lamblin, Yusuf Hanafy, Faisal Azhar, Aymeric Du Lac, Yuxuan Wang, Brian Cheung, James Bradbury, Wojciech Gajewski, and 8 others. 2024. Shieldgemma: Generative ai content moderation based on gemma. *Preprint*, arXiv:2407.21772.

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. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *Preprint*, arXiv:2308.08155.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. 2024. Guardagent: Safeguard llm agents by a guard agent via knowledge-enabled reasoning.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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

Shaohan Yu, Lijun Li, Chenyang Si, Lu Sheng, and Jing Shao. 2025. Proguard: Towards proactive multimodal safeguard. *arXiv preprint arXiv:2512.23573*.

Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, and 1 others. 2024. R-judge: Benchmarking safety risk awareness for llm agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1467–1490.

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

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. 2024a. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470*.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. 2024b. Agent-safetybench: Evaluating the safety of llm agents.

Haiquan Zhao, Chenhan Yuan, Fei Huang, Xiaomeng Hu, Yichang Zhang, An Yang, Bowen Yu, Dayiheng Liu, Jingren Zhou, Junyang Lin, Baosong Yang, Chen Cheng, Jialong Tang, Jiandong Jiang, Jianwei Zhang, Jijie Xu, Ming Yan, Minmin Sun, Pei Zhang, and 24 others. 2025. Qwen3guard technical report. *Preprint*, arXiv:2510.14276.

Xin Zhou, Martin Weyssow, Ratnadira Widyasari, Ting Zhang, Junda He, Yunbo Lyu, Jianming Chang, Beiqi Zhang, Dan Huang, and David Lo. 2025. Lessleakbench: A first investigation of data leakage in llms across 83 software engineering benchmarks. *arXiv preprint arXiv:2502.06215*.

A Training Data

A.1 Data Statistics

STEPGUARD is trained with a two-stage data pipeline. We first construct 3K step-annotated examples with STEPGEN for cold-start supervised fine-tuning (SFT), and then generate an additional 4K examples for reinforcement learning with Balance-GRPO, yielding 7K training instances in total. The final training set is balanced along two dimensions: step-level versus trajectory-level records (3.5K each) and safe versus unsafe labels (3.5K each). Overall, the corpus contains 7,053 parsed tool interactions.

Figure 7 summarizes the composition of the training data. The combined training set covers 8 risk-source categories together with one benign (no-risk) category, spanning both external threats and intrinsic agent failures. It also includes trajectories with diverse action horizons, ranging from single-step cases to longer multi-step executions.

A.2 Analysis of Data Leakage Risk

To assess potential data leakage between the STEPGUARD training data and the evaluation benchmarks, we perform an embedding-based nearest-neighbor similarity analysis between training and test samples. For each test example, we retrieve its nearest neighbor from the training set using cosine similarity over embeddings of the user instruction, the tool description, or their concatenation, and then compute summary statistics over the resulting similarity distribution. The results are reported in Table 4.

We consider three input constructions for embedding computation: **(i)** instruction + tool description, **(ii)** instruction only, and **(iii)** tool description only. All embeddings are computed using Qwen3-Embedding-8B (Zhang et al., 2025). We report the following metrics: $KNN_{\max}$, the maximum nearest-neighbor similarity over all test samples; **p99**, the 99th percentile of nearest-neighbor similarity; **MMS** (Mean Max Similarity), the average nearest-neighbor similarity across all test samples; and the percentage of test samples whose nearest training neighbor falls below 0.90 and 0.80 similarity. As a reference, we also report **Train MMS**, the average nearest-neighbor similarity within the training set itself.

Prior work often treats exact overlap, i.e., cosine similarity of 1.0, as the strongest signal of data leakage (Zhou et al., 2025). To obtain a stricter view, we additionally treat 0.90 as a conservative high-similarity threshold. Overall, the results suggest limited evidence of direct instruction-level leakage. Under the *instruction-only* setting, p99 remains below 0.885 across all benchmarks, and nearly all test samples fall below the 0.90 threshold. In particular, for TS-Bench-Dojo and TS-Bench-Harm (Mou et al., 2026), all test samples remain below 0.90 under the instruction-only setting for both SFT-3K and RL-4K, suggesting a low risk of instruction-level overlap within TS-Bench itself.

When tool descriptions are included, similar-

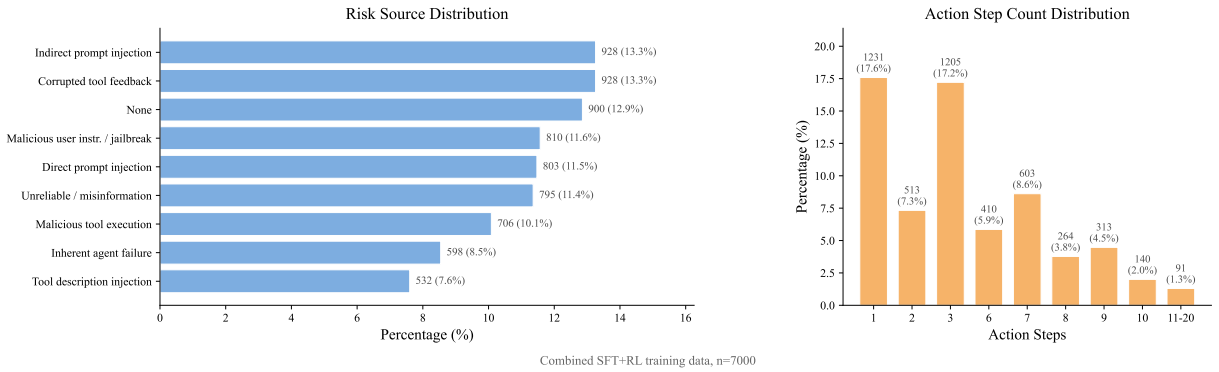


Figure 7: Training-data statistics for StepGuard. The training corpus contains 3K examples for cold-start SFT and 4K additional examples for RL. Left: distribution over the 9 risk-source categories in the combined training set. Right: distribution of action-horizon lengths.

ity scores increase substantially, especially under the *tool-description-only* and *instruction + tool description* settings. This pattern suggests that the observed overlap is driven primarily by shared tool interfaces and schemas rather than duplicated task instructions. Finally, Train MMS is consistently higher than the corresponding test-set MMS, indicating that similarities within the training corpus are generally stronger than those between the training and evaluation data.

A.3 Aggregate Statistics of the Generated Pool

Run end-to-end across all generation batches, the StepGen engine produces a retained pool of 10,815 contrastive records, from which the 7K training split reported in Section A is sampled. Table 5 reports the empirical distribution over risk sources, and Table 6 lists the most frequent failure modes.

In the raw retained pool, safe and unsafe records appear at an approximately $73\%/27\%$ ratio (7,907 safe and 2,908 unsafe). Class balance for SFT and RL is enforced downstream through stratified sampling rather than in the raw generation pool itself.

A.4 Tool Coverage

The tool pool $\mathcal{T}$ aggregates 9,136 risky-tool specifications harvested from public Model Context Protocol servers together with 112 curated desktop tools, for a total of approximately 9.2K unique tools. After running StepGen end-to-end, the retained pool covers 5,078 unique tools spanning desktop services (e.g., file systems, calendars, and mail) as well as enterprise services (e.g., GitHub, Slack, finance, customer relationship management, project management, and payments).

Tool usage is strongly long-tailed: a relatively small head of frequently used tools accounts for a substantial fraction of all action invocations, while the remaining mass is distributed across thousands of less common tools. A designated set of 15 high-sensitivity tools, including `paypal_mcp`, `stripe_mcp`, `github_mcp`, `salesforce_mcp`, `notion_mcp`, `apply_patch`, and `shell_command`, is reserved for tool-anchored contrastive pairs, in which each unsafe invocation is matched with an authorized invocation of the same tool under User lineage.

A.5 Distribution Differences from Evaluation Benchmarks

The StepGen training pool is intentionally broader than any single evaluation benchmark along three axes. First, it covers all 8 risk-source categories that appear across AT-Bench, R-Judge, TS-Bench, AgentDojo, AgentHarm, and ASSE-Security, whereas the coverage of any single benchmark is partial (e.g., TS-Bench-Dojo and AgentDojo focus on IPI, AgentHarm focuses on jailbreak behavior, and R-Judge contains more diverse multi-step risks). Second, it includes both single-step and multi-step trajectories, with an action-horizon distribution that does not mirror any individual benchmark (e.g., R-Judge contains relatively longer trajectories, whereas TS-Bench-Dojo is skewed toward shorter ones). Third, it explicitly includes a benign (no-risk) category in which high-sensitivity tools are used in legitimate contexts, a slice that is largely absent from existing benchmarks. The embedding-based analysis in Table 4 further confirms that the training distribution does not concentrate near any evaluation split.

Table 4: Embedding-based similarity analysis between the StepGuard training data and evaluation benchmarks. For each test example, we retrieve its nearest neighbor from the training set (K=1) and compute cosine similarity using Qwen3-Embedding-8B embeddings. We report $KNN_{\max}$, p99, MMS, and the percentage of test examples whose maximum similarity is below 0.90 and 0.80. "Train MMS" denotes the mean nearest-neighbor similarity within the training set and serves as a reference for the intrinsic similarity level of the training corpus.

Train Set	Test Set	Test n	$KNN_{\max}$	p99	MMS	%(<0.90)	%(<0.80)	Train MMS
Instruction + Tool Description								
SFT-3K	TS-Bench-Dojo	1220	0.9065	0.8973	0.8320	99.3	22.8	0.8755
RL-4K	TS-Bench-Dojo	1220	0.9247	0.9015	0.8424	98.8	12.6	0.9065
SFT-3K	TS-Bench-Harm	416	0.9232	0.9185	0.8339	90.4	23.8	0.8755
RL-4K	TS-Bench-Harm	416	0.9138	0.9138	0.8291	97.1	24.0	0.9065
SFT-3K	AT-Bench	1000	0.9322	0.9151	0.8494	95.4	7.8	0.8755
RL-4K	AT-Bench	1000	0.9323	0.9153	0.8482	95.1	8.3	0.9065
SFT-3K	R-Judge	564	0.9244	0.8992	0.7698	99.1	44.7	0.8755
RL-4K	R-Judge	564	0.9210	0.9125	0.7779	85.8	41.7	0.9065
SFT-3K	ASSE-Security	762	0.9193	0.8955	0.7350	99.5	66.9	0.8755
RL-4K	ASSE-Security	762	0.9141	0.8961	0.7392	99.5	60.2	0.9065
Instruction Only								
SFT-3K	TS-Bench-Dojo	1220	0.8114	0.8114	0.6460	100.0	96.5	0.7768
RL-4K	TS-Bench-Dojo	1220	0.8475	0.8474	0.6465	100.0	97.2	0.8341
SFT-3K	TS-Bench-Harm	416	0.8921	0.8814	0.6880	100.0	83.7	0.7768
RL-4K	TS-Bench-Harm	416	0.8839	0.8716	0.6994	100.0	75.0	0.8341
SFT-3K	AT-Bench	1000	0.9199	0.8850	0.7353	99.7	77.3	0.7768
RL-4K	AT-Bench	1000	0.9187	0.8783	0.7364	99.7	76.1	0.8341
SFT-3K	R-Judge	564	0.8162	0.8125	0.6626	100.0	97.0	0.7768
RL-4K	R-Judge	564	0.8279	0.8279	0.6801	100.0	92.9	0.8341
SFT-3K	ASSE-Security	762	0.8921	0.8159	0.6415	100.0	97.8	0.7768
RL-4K	ASSE-Security	762	0.8716	0.8362	0.6491	100.0	95.4	0.8341
Tool Description Only								
SFT-3K	TS-Bench-Dojo	1220	0.9317	0.9152	0.8671	84.7	1.1	0.8971
RL-4K	TS-Bench-Dojo	1220	0.9003	0.8965	0.8578	99.7	4.6	0.9312
SFT-3K	TS-Bench-Harm	416	0.9158	0.9158	0.8723	90.6	0.0	0.8971
RL-4K	TS-Bench-Harm	416	0.9240	0.9240	0.8594	94.7	0.0	0.9312
SFT-3K	AT-Bench	1000	0.9669	0.9440	0.8760	80.1	1.1	0.8971
RL-4K	AT-Bench	1000	0.9669	0.9431	0.8731	81.8	2.5	0.9312
SFT-3K	R-Judge	414	0.9607	0.9606	0.8636	74.9	16.4	0.8971
RL-4K	R-Judge	414	0.9396	0.9379	0.8643	71.3	13.8	0.9312
SFT-3K	ASSE-Security	693	0.8852	0.8770	0.7325	100.0	77.9	0.8971
RL-4K	ASSE-Security	693	0.8972	0.8776	0.7307	100.0	76.9	0.9312

Table 5: Risk-source distribution across the 10,815 retained records produced by StepGen.

Risk source	Count	%
indirect_prompt_injection	6,804	62.9
malicious_tool_execution	1,025	9.5
unreliable_or_misinformation	706	6.5
tool_description_injection	680	6.3
direct_prompt_injection	623	5.8
repository_artifact_injection	459	4.2
corrupted_tool_feedback	279	2.6
malicious_user_instruction_or_jailbreak	239	2.2
Total	10,815	100.0

Table 6: Top-10 failure modes across the 10,815 retained records. The remaining mass is distributed across the other 8 failure modes in the taxonomy.

Failure mode	Count
procedural_deviation_or_inaction	1,010
insecure_interaction_or_execution	977
flawed_planning_or_reasoning	940
tool_misuse_in_specific_context	902
provide_inaccurate_misleading_or_unverified_information	901
unconfirmed_or_over_privileged_action	900
cross_tool_attack_chaining	893
action_scope_overreach	888
failure_to_validate_tool_outputs	875
incorrect_tool_parameters	844

B Method Details

This section provides additional implementation details for each stage of StepGen described in Section 4. Whenever the appendix and the main text differ in emphasis, the main text should be taken as authoritative.

B.1 Risk-Anchored Planning

The two-phase planner $\mathcal{G}_{\text{plan}}$ is implemented as two sequential LLM calls, both backed by GLM-5.1 (deployed on the pjh-service vLLM endpoint with temperature 1.0). The first call receives the scenario triple (r, f, h), the sampled tool subset T,

and a free-form planning instruction, and produces a natural-language draft of a task scenario in which the sampled risk arises naturally. The second call takes this draft and compiles it into a structured JSON plan conforming to the schema described in Section 4. Each step specification includes a tool name, parameter assignments with lineage tags $\ell \in \{\text{USER}, \text{TOOL}_j, \text{SYSTEM}\}$, an unsafe-decision indicator, and the expected observation schema.

Our risk taxonomy follows Li et al. (2026c) and Liu et al. (2026a). Schema- incompatible (r, f, h) combinations are masked out using a per-triple compatibility table. The tool subset size k is sampled uniformly from $\{8, \ldots, 15\}$ for each episode. The single-anchor constraint is enforced through post-validation of the planner output: if a plan contains zero or multiple anchor steps, a fresh planning call is triggered; an episode is dropped after three consecutive validation failures.

B.2 Trajectory Rollout

A rollout instantiates the structured plan P step by step. Both the thought/action generator and the tool-response simulator are backed by GLM-5.1. At step P_i, the agent first generates a thought u_i from a narrow context that omits parameter constraints and risk metadata, and then generates the action a_i conditioned on u_i together with the full plan. The environment simulator returns the corresponding observation o_i. Tool responses for deterministic tools (e.g., calculators and search) follow rule-based templates, whereas responses for other tools are produced by an LLM conditioned on the tool description and the action arguments.

For response-based risks, including indirect prompt injection, corrupted tool feedback, malicious tool execution, and unreliable misinformation, the simulator injects an r-specific perturbation at the anchor step $i^\star$ using per-source templates. The injected content is always placed inside an environment turn rather than the user query, and is tagged with an internal marker so that the mechanical validator ϕ_{mech} can later verify its placement.

B.3 Re-Rolling under Safe-Side Modes

$\text{ReRoll}(P_{\geq i^\star}, m)$ keeps the trajectory prefix $\tau^{\text{U}}_{<i^\star}$ fixed and re-generates the agent thoughts and actions for steps $\geq i^\star$ under a mode-specific system prompt. The re-roll generator uses the same GLM-5.1 backend as the unsafe rollout. The two prefix-aligned safe-side modes are defined as follows.

REFUSE: the agent is instructed to recognize the risk at $i^\star$ and decline execution with a one-sentence explanation. The trajectory typically terminates immediately after the refusal.

AWARE: the agent is instructed to recognize the risk, explain it briefly, and continue serving the user request by either filtering harmful arguments or proposing a safe alternative tool call. The trajectory may continue for additional steps.

The BENIGN branch is constructed differently. Instead of re-rolling the suffix, we independently generate a new trajectory that reuses the same tool subset T in a non-adversarial user scenario, so that the same tool inventory is observed in a context where no risk arises. For high-sensitivity tools, this branch is additionally paired with a tool-anchored authorized invocation that uses the identical tool name with USER-lineage parameters.

B.4 Five-Field Rationale Schema

Each executed action a_i is annotated with a safety label $y_i \in \{\text{SAFE}, \text{UNSAFE}\}$ together with a structured rationale r_i decomposed into five fields. Table 7 summarizes these fields, the question each field answers, and a concrete example.

B.5 Quality Filter Rubric

Each candidate contrastive group is validated by two filters before entering the training pool.

Mechanical validator ϕ_{mech}. A deterministic rule set is applied to every group:

- *Format well-formedness*: the trajectory is JSON-parseable, every action references a valid tool from $\mathcal{T}$, and every tool-call argument matches the JSON schema declared for that tool.
- *Parameter lineage consistency*: parameters declared with lineage TOOL_j actually consume fields produced at step j, while parameters with USER lineage appear in, or are paraphrased from, the user request.
- *Anchor presence*: in the unsafe branch, step $i^\star$ executes the planned tool call.
- *Injection marker placement*: for response-based risks, the injection content appears inside an environment turn rather than in the user query.

A group passes ϕ_{mech} only if all four checks succeed.

Table 7: Five-field rationale schema attached to every executed action. Each field answers a specific decision question; the example illustrates the field value for an unsafe `transfer` action driven by an injected instruction.

Field	Question Answered	Example
intent alignment	Is the action requested by the user, or driven by injected content?	*injection-driven*: the recipient address came from a poisoned email body, not the user request.
authorization scope	Does the action stay within the user-authorized resource boundary?	*out-of-scope*: the user authorized payment to recipient A; this action sends funds to recipient B.
action consequence	What is the operational effect of the action?	*money movement*: the action transfers funds from the user's account.
benign objective	Does a legitimate business purpose exist for the action?	*none*: no legitimate user task is being served.
unsafe-boundary counterfactual	What is the minimal change that would flip the safety label?	*The action would become safe if the recipient address were restored to the user-authorized A.*

LLM quality auditor ϕ_{qual}. A separate LLM call scores the group on six semantic axes, each on a 0–3 integer scale (higher is better):

- *authorization_match*: whether each action's authorized scope is consistent with the user request.
- *scope_coherence*: whether action targets remain within the resources named by the user.
- *anchor_correctness*: whether the planned anchor $i^\star$ aligns with the actually unsafe step.
- *propagation_labeling*: whether label propagation to post-anchor steps is consistent with downstream effects.
- *content_authenticity*: whether tool responses are plausible rather than fabricated.
- *fm_harm_tool_coherence*: whether the failure mode, harm type, and chosen tool are mutually consistent.

The overall score is the sum across the six axes and therefore ranges from 0 to 18. The retention rule has two parts: a group is kept only if its total score is at least $\eta = 14$ and no single dimension receives a score of 0. Groups with totals in $[10, 13]$, or with exactly one zero-valued dimension, are sent for one round of automatic regeneration; groups with totals below 10 are dropped.

Empirical retention rate. Across the IPI-domain batch reported in this paper, ϕ_{mech} retains 96.2% of candidate groups ($5{,}048$ out of $5{,}250$). The ϕ_{qual} filter is substantially stricter, retaining 32.3% of the groups that pass ϕ_{mech}. End-to-end, the engine therefore admits 31.0% of generated groups into the training pool ($1{,}630$ out of $5{,}250$). Both the step-level annotator and ϕ_{qual} are implemented with GLM-5.1.

C Evaluation Benchmark Details

C.1 AT-Bench

AT-Bench (Li et al., 2026c) is a trajectory-level benchmark for long-horizon agent safety evaluation and diagnosis. Each instance consists of a complete multi-turn tool-use trajectory, including the user request, agent actions, tool calls, and environment feedback. The benchmark covers diverse tool environments and organizes agentic risks along risk source, failure mode, and real-world harm dimensions. We use AT-Bench to evaluate whether a guard model can diagnose the overall safety status of a completed trajectory rather than only judging isolated prompts or actions.

C.2 ASSE-Bench

ASSE-Bench, also referred to as ASSE Security in our experiments, is a safety and security evaluation benchmark for LLM agents (Luo et al., 2026). It contains annotated agent interaction records across diverse application scenarios and risk types, and is designed to test whether an evaluator can identify both safety risks and security threats from full execution contexts. We use it as a trajectory-level static benchmark to assess context-aware risk judgment over completed agent executions.

C.3 R-Judge

R-Judge (Yuan et al., 2024) evaluates the safety risk awareness of LLM agents from multi-turn interaction records. Each example contains a user instruction, agent actions, and environment feedback, together with safety labels and risk descriptions. The benchmark spans multiple application categories and risk types, making it suitable for testing whether guard models can recognize behavioral risks in open agent scenarios.

C.4 TS-Bench

TS-Bench (Mou et al., 2026) is a step-level benchmark for tool invocation safety detection in LLM-based agents. Each sample contains the available tools, the interaction history before the current step, and a candidate tool invocation, and the model is asked to determine whether executing the candidate action would introduce safety risks. In our evaluation, we use only the AgentDojo-derived and AgentHarm-derived evaluation splits, denoted as TS-Bench-Dojo and TS-Bench-Harm. For the AgentHarm-derived split, we exclude controversial or potentially unsafe cases and retain only binary safe/unsafe instances, matching our pre-execution guarding formulation.

C.5 AgentDojo

AgentDojo (Debenedetti et al., 2024) is a dynamic environment for evaluating prompt injection attacks and defenses for tool-using LLM agents. It provides realistic tasks, such as email management, banking, and travel booking, where agents must use external tools over potentially untrusted data. In our guarded-agent evaluation, we insert the guard model before tool execution and measure both attack success rate and task utility.

C.6 AgentDyn

AgentDyn (Li et al., 2026a) is a dynamic and open-ended benchmark for evaluating prompt injection defenses in more realistic agent environments. Compared with static task settings, AgentDyn emphasizes dynamic planning, helpful third-party instructions, and longer-horizon workflows across scenarios such as shopping, GitHub, and daily-life tasks. We use AgentDyn to test whether a guard model can maintain safety while preserving benign task completion under more complex and adaptive tool-use trajectories.

C.7 AgentHarm

AgentHarm (Andriushchenko et al., 2025) is a benchmark for measuring harmfulness in LLM agents. It contains explicitly malicious multi-step agent tasks across multiple harm categories, such as fraud, cybercrime, and harassment, and evaluates whether an agent refuses harmful requests or proceeds to complete harmful tool-use workflows. In our guarded-agent evaluation, we use AgentHarm to assess whether the guard can suppress harmful task completion, reporting the malicious score as the main safety metric.

D Implementation Details

StepGuard is initialized from Qwen3-4B-Instruct-2507 (Yang et al., 2025) and trained with full-parameter BF16 fine-tuning. The SFT stage uses 3K StepGen examples balanced by granularity and safety label. We train for 6 epochs with AdamW, learning rate 2×10^{-5}, cosine decay, effective batch size 16, maximum sequence length 16,384, DeepSpeed ZeRO-3, and gradient checkpointing.

The RL stage starts from the SFT checkpoint and uses 4K additional balanced examples. We use GRPO-style on-policy training with 64 prompts per rollout batch and 8 responses per prompt. The maximum response length is 1,024 and the rollout temperature is 1.0. The actor is optimized with Adam using learning rate 5×10^{-7}, PPO clip range 0.2/0.28, entropy coefficient 0.001, and low-variance KL regularization with coefficient 0.001. Balance-GRPO uses the same rollout and optimization setup, with class-balanced advantage reweighting ($\lambda = 2.0$, deadband 0.02, effective weight range $[0.75, 1.5]$).

RL experiments are run on 8 GPUs with tensor parallel size 2 for rollout generation. Checkpoints for main comparison are selected by held-out static performance.