DynSTEER: Dynamic Stage-wise Trajectory Evaluation and Execution-time Review for Agents

Anonymous authors
Paper under double-blind review

Abstract

Large language model agents are increasingly deployed for long-horizon task execution, raising a central granularity question for trajectory evaluation: whole-trajectory verification is too coarse to capture concrete failures and their associated evidence in long trajectories, while atomic-step scoring is too fine-grained, noise-sensitive, and computationally expensive. This granularity gap makes a single-reference trajectory paradigm inadequate for assessing the rich space of valid agent execution paths and delays timely feedback and early stopping in long-horizon tasks. To address these issues, we propose DynSTEER, a dynamic stage-wise framework for agent trajectory evaluation. DynSTEER bridges the granularity gap through stage-wise dynamic evaluation that segments rollouts at key execution nodes and adapts its multi-level review strategy based on stage-level results; it compiles a path-tolerant milestone graph from available task inputs to preserve diverse legal paths without reference leakage; and it supports terminating unrecoverable agent executions to curb resource waste. Experimental results show that DynSTEER improves evaluation discriminability by over 85% compared with whole-trajectory evaluation and saves 17.74% of execution steps. The code is available at `https://anonymous.4open.science/r/DynSTEER`

1 Introduction

LLM agents are increasingly undertaking complex, long-horizon tasks across real-world environments (Yao et al., 2023; Shinn et al., 2023; Jimenez et al., 2024; Gu et al., 2024; Zhou et al., 2024). As task horizons expand, evaluating only final output results becomes insufficient to identify agent deficiencies and diagnose failure modes, therefore bringing trajectory-level evaluation to the forefront of agent research (Qian et al., 2024; Shi et al., 2026).

However, existing trajectory evaluation paradigms face a central granularity problem, especially for long-horizon tasks. Whole-trajectory verification is too coarse to focus on concrete failures and their associated evidence in long trajectories, whereas atomic-step scoring is too fine-grained: isolated messages lack task context, so ordinary retries and harmless observations become noise. This granularity gap further raises two challenges. Agent executions often admit multiple valid and effective paths. Because of this granularity gap, evaluation typically offers only limited path references and therefore struggles to faithfully reflect the quality of diverse agent executions. Moreover, full-trajectory review delays diagnosis that could be made earlier, so an execution that has already gone wrong may continue, wasting model resources and triggering additional skill or tool calls.

As illustrated in Figure 1, we introduce **DynSTEER**, a DYNamic **S**tage-wise **T**rajectory **E**valuation and **E**xecution-time **R**eview framework. To address the granularity challenge, DynSTEER performs stage-wise dynamic evaluation: it segments rollouts at necessary execution nodes, preserves task context while isolating localized evidence for error attribution, and adapts its multi-level review strategy based on stage-level results. To avoid single-reference bias, it compiles a path-tolerant milestone graph from available task inputs via multi-candidate simulated planning and majority consensus, capturing necessary task invariants while accommodating diverse legitimate paths. Finally, it detects unrecoverable prefixes and terminates them to curb resource waste.

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.

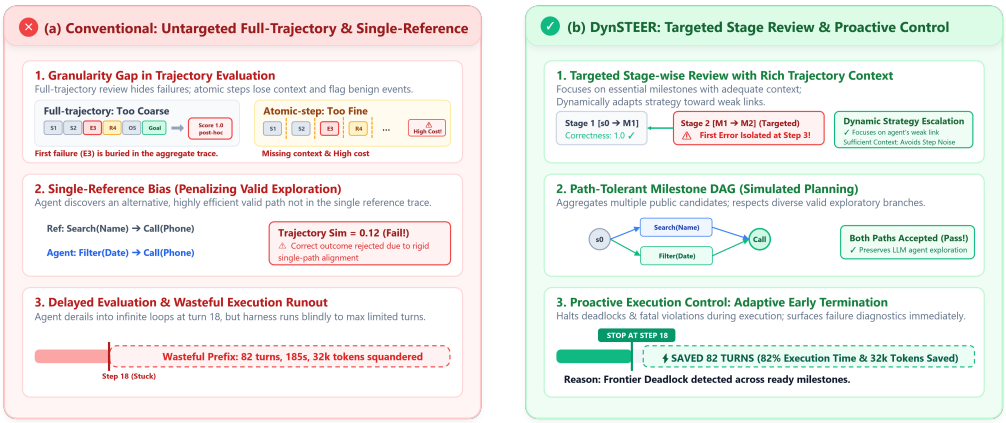

Figure 1: **Comparison of Agent Evaluation Paradigms.** (a) Conventional paradigms evaluate either coarse outcomes or noisy atomic steps, often against a single reference, and discover waste only after execution. (b) DynSTEER evaluates path-tolerant milestone-bounded stages and uses the resulting stage evidence to target review and stop hopeless prefixes early.

Experiments on ToolSandbox and SWE-bench Pro show that DynSTEER improves discriminability by over 85% compared with whole-trajectory evaluation and saves 17.74% and 8.40% of execution steps. Comparisons with expert-annotated graphs from ToolSandbox demonstrate the validity of DynSTEER's compiled milestone graphs. Extensive ablation studies further confirm the effectiveness of its dynamic adaptation and early-stopping mechanisms in better distinguishing the capabilities of different models as agent backbones.

In summary, our contributions are as follows:

- **Stage-wise Dynamic Evaluation:** We segment rollouts at necessary execution-step nodes and adapt multi-level review routing from stage-level results, isolating context-rich evidence windows for multidimensional scoring and first-error localization.
- **Path-Tolerant Task Blueprint Compilation:** We compile a milestone DAG from available task inputs via multi-candidate planning and strict-majority consensus, preserving diverse valid paths without reference leakage.
- **Counterfactual Early Termination:** We detect unrecoverable prefixes from stage-level evidence and terminate them, substantially reducing wasted computation.

2 Related Work

2.1 Agent Trajectory Evaluation

Unlike outcome benchmarks that inspect only terminal code patches or database records (Jimenez et al., 2024; Qin et al., 2024; Liu et al., 2024; Zhou et al., 2024), trajectory evaluation examines intermediate planning decisions, tool invocations, and error recoveries generated during agent execution. Recent benchmarks expand assessment: BFCL (Patil et al., 2025) evaluates multi-step tool calls via AST checks; TRAJECT-Bench (Li et al., 2025) assesses tool selection and dependency order; AgentRewardBench (Zhou et al., 2025) benchmarks trajectory judges; and DeepRed (Kuo et al., 2026) and Claw-Eval (Zhang et al., 2026) leverage environment snapshots to award partial credit along execution traces.

Despite providing intermediate visibility, existing evaluators face two key limitations. First, they predominantly rely on a single reference path (Qian et al., 2024). In open environments, tasks frequently admit multiple valid solution routes; single-path matching inherently penalizes legitimate exploratory strategies. Second, they operate post-hoc across the full rollout and lack dynamic intervention to adapt evaluation strategies to an agent's specific weak links.

2.2 Dynamic Evaluation for Agents

Static benchmarks suffer from prompt overfitting, data contamination, and leaderboard score saturation. To address these vulnerabilities, researchers have begun exploring dynamic evaluation paradigms. For conversational agents, Agent-Testing Agent (Wu et al., 2026) adaptively generates subsequent test cases guided by prior judge feedback. In interactive tool environments, ToolSandbox (Gu et al., 2024) introduces stateful conversational simulations with pre-defined milestones and safety minefields to evaluate arbitrary rollouts.

However, existing dynamic evaluation approaches primarily concentrate on dynamically selecting or perturbing benchmark test suites (Gu et al., 2024; Wu et al., 2026). During the actual execution of an individual task, trajectory evaluation remains largely passive: evaluators typically check simple milestone reachability at the end of rollouts, lacking a formal mechanism to dynamically steer evaluation focus, update dimension weights, or control execution progression in real time.

2.3 Efficient Evaluation and Early Stopping Strategies

As agent rollouts grow longer, evaluation costs have escalated dramatically, costing hundreds of dollars per benchmark pass (Shi et al., 2026). Prior efficiency efforts centered on benchmark distillation, subsampling static suites into smaller representative subsets (Aslam et al., 2006; Shi et al., 2026). To improve within-task efficiency, EarlyEval (Shi et al., 2026) introduces early outcome prediction, training dual LightGBM classifiers on prefix features to predict binary task success/failure and halt execution early.

Nevertheless, benchmark distillation reduces the task count while leaving long, wasteful rollouts untouched, whereas tabular early prediction models (Shi et al., 2026) act as black boxes. They predict binary outcomes without understanding domain semantics, cannot provide causal failure attribution, and lack awareness of tool dependency constraints or hazardous minefields.

In summary, existing trajectory evaluators either inspect isolated atomic steps that suffer from local noise, or evaluate full traces post-hoc against limited reference paths that penalize valid diversity. Meanwhile, early stopping methods lack domain semantics and failure attribution. To bridge this divide, we propose DynSTEER. DynSTEER performs stage-wise dynamic evaluation that adapts review capacity to weak or ambiguous stages. It compiles path-tolerant milestone graphs from available task inputs. Finally, it uses execution-time early stopping to eliminate wasteful computation.

3 Methodology

DynSTEER establishes a dynamic evaluation framework for agent trajectories, as illustrated in Figure 2. Rather than treating rollouts as opaque black boxes or enforcing rigid single-path matching, DynSTEER coordinates three core components: stage-wise dynamic evaluation to resolve the evidence-granularity problem, which Sections 3.3 and 3.4 present; path-tolerant milestone-graph generation to preserve execution diversity, which Section 3.2 presents; and execution-time early termination to curb resource waste, which Section 3.4 presents.

3.1 Task Inputs and Dual-Clock Observation

Following interactive tool-using agent environments (Gu et al., 2024; Yang et al., 2023; Liu et al., 2024), a task provides instruction x, initial state $\mathcal{S}_0$, API catalog $\mathcal{A}$, and safety contracts $\mathcal{C}$. DynSTEER evaluates the task inputs:

$$V_{\text{task}} = (x, \mathcal{A}, \mathcal{I}_0), \tag{1}$$

where $\mathcal{I}_0$ is evaluator-visible initial-state metadata; hidden reference paths τ^* and terminal unit tests are never used for graph construction.

During execution, the trace $\tau = \langle e_1, \ldots, e_K \rangle$ is observed on two clocks. The raw event clock k indexes every observed event and triggers immediate fatal-action interception, whereas the closed-action clock t advances only after a tool return is observed and the agent yields control. Milestone settlement and scoring use t. For example, in $\langle$tool call, tool return, agent yield$\rangle$, the three observations receive raw indices $k = 1, 2, 3$, so a hazardous tool call can be intercepted at $k = 1$. The action

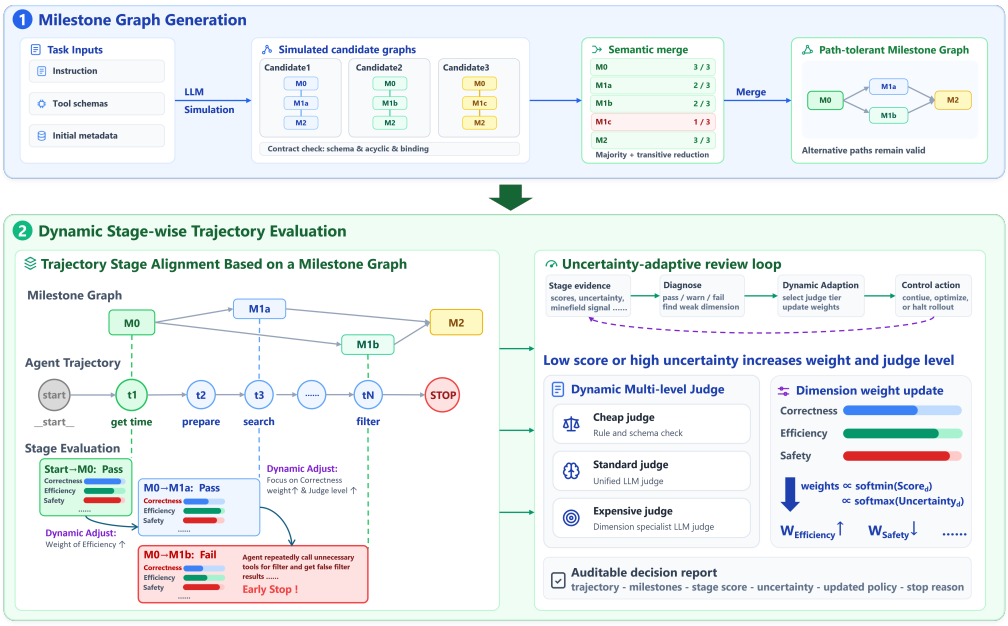

Figure 2: **Overview of the DynSTEER Workflow.** (1) Pre-execution milestone graph generation synthesizes candidate graphs from task inputs to compile a path-tolerant graph. (2) Dynamic stage-wise evaluation aligns trajectory and milestone graphs to evaluate weak links with trajectory context, adaptively scales judge levels, and records auditable early-stop decisions.

closes at $k = 3$, where t advances from 0 to 1 and becomes the unit used for milestone settlement. This dual-clock design couples safety responsiveness with semantic completeness: the raw event clock exposes fatal minefield operations as soon as the agent attempts them, whereas the closed-action clock prevents milestone settlement from missing information or state changes.

3.2 Milestone Graph Generation via Simulated Planning

To overcome single-reference bias, DynSTEER compiles V_{task} into a path-tolerant Milestone Graph $G = (M, E, \mathcal{K}, \mathcal{M}_{\text{fatal}})$ prior to execution, where M is the milestone set, E encodes precedence constraints, $\mathcal{K}$ specifies parameter bindings, and $\mathcal{M}_{\text{fatal}}$ defines hazardous operations.

Multi-Candidate Synthesis and Contract Validation. Given V_{task}, a blueprint generator synthesizes C candidate plans $\{G_1, \ldots, G_C\}$ via LLM prompting across diverse exploration heuristics, such as direct keyword queries versus hierarchical filtering. Each candidate $G_c = (M_c, E_c)$ undergoes deterministic contract verification:

$$\text{Valid}(G_c) = \text{SchemaCheck}(M_c, \mathcal{A}) \wedge \text{Acyclic}(E_c) \wedge \text{BindingConsistency}(\mathcal{K}_c) \tag{2}$$

Here, SchemaCheck requires every milestone and minefield to name an available API and to match its argument and return schema; Acyclic rejects circular precedence constraints; BindingConsistency requires dynamic bindings to reference declared upstream outputs with compatible parameter types. A candidate failing any check is discarded. Appendix C.2 details these deterministic filters.

Majority Consensus and Transitive Reduction. To retain essential invariants while preserving exploratory flexibility, a milestone m is retained only if it appears in a majority of valid candidates:

$$M = \left\{ m \;\middle|\; \sum_{c=1}^{C_{\text{valid}}} \mathbb{I}[m \in M_c] \geq \left\lceil \frac{C_{\text{valid}} + 1}{2} \right\rceil \right\} \tag{3}$$

Precedence edges are aggregated analogously. Finally, we compute the transitive reduction $E = \text{TR}(E_{\text{majority}})$ (Tarjan, 1972) to eliminate redundant shortcut dependencies, producing a minimal directed acyclic graph that accommodates diverse legitimate execution strategies.

3.3 STAGE-WISE DYNAMIC TRAJECTORY EVALUATION

During execution, DynSTEER dynamically aligns closed actions against the milestone graph. Algorithm 1 in Appendix C presents the complete algorithmic flow.

Ready Frontier Matching. Let $M_{<t}$ denote milestones settled prior to closed action t. The active search space is strictly bounded by the **ready frontier**:

$$\mathcal{F}_t = \{m \in M \setminus M_{<t} \mid \text{Pred}(m) \subseteq M_{<t}\} \tag{4}$$

An action a_t settles milestone $m^* \in \mathcal{F}_t$ if its tool call and state delta satisfy semantic predicates $\phi_{m^*}(a_t, s_t) = 1$. Restricting alignment to $\mathcal{F}_t$ prevents out-of-order settlement and reward gaming.

Dominator-Anchored Stage Isolation. To construct a semantically complete stage evidence window for m^*, DynSTEER computes the immediate dominator $\text{idom}(m^*)$ in G (Lengauer & Tarjan, 1979), defined as the unique node that must be traversed on every directed path from start node s_0 to m^*. The **stage evidence window** is bounded by:

$$\mathcal{W}(m^*) = [t_{\text{idom}(m^*)}, t] \tag{5}$$

where $t_{\text{idom}(m^*)}$ is the execution step at which the dominator was settled. Anchoring evidence to $\mathcal{W}(m^*)$ preserves all intermediate reasoning, trial-and-error attempts, and parameter adjustments that lead directly to m^*, ensuring that relevant actions are not omitted merely because the agent executes them in parallel branches. Unlike whole-trajectory evaluation, which obscures mistake locations, and single-step evaluation, which lacks context, dominator-anchored stages provide enough trajectory context to evaluate intermediate quality and pinpoint the exact first-error step.

Multidimensional Scoring. DynSTEER scores the entire evidence window $\mathcal{W}(m^*)$. Each stage uses a task-relevant subset $\mathcal{D}_j \subseteq \mathcal{D}$ of seven dimensions: progress, state consistency, tool quality, efficiency, safety, interaction quality, and recovery. Let j denote the evaluated-stage clock, independent of the closed-action clock t; stage j ends at the closed action at which a milestone candidate is evaluated. The composite stage score is:

$$S_j = \sum_{d \in \mathcal{D}_j} w_d^{(j)} s_d^{(j)}(m^*) \tag{6}$$

parameterized by dynamically normalized weights $\mathbf{w}^{(j)}$. Uncertainty-adaptive multi-level review then treats these stage scores and uncertainty estimates as control signals, allocating stronger review capacity to ambiguous or weak dimensions.

3.4 UNCERTAINTY-ADAPTIVE REVIEW AND EXECUTION-TIME EARLY TERMINATION

Uncertainty-adaptive review is the control layer for stage-wise evaluation, adjusting dimension weights and judge levels after each stage. Execution-time early termination is a separate downstream safeguard that reuses the same stage evidence to control resource waste.

Multi-Level Judge Structure. Rather than assigning a uniform evaluator across all stages, DynSTEER deploys a three-level judge hierarchy whose prompt templates are detailed in Appendix F:

- **Level 1: Cheap Judge** (J_{cheap}): A deterministic structural evaluator inspecting tool status codes, schemas, and evidence rules without LLMs, incurring zero model inference overhead.
- **Level 2: Standard Judge** (J_{std}): A multi-field unified LLM evaluator scoring all target dimensions simultaneously within a single pass, aggregating confidence across multiple samples.
- **Level 3: Expensive Judge** (J_{exp}): A single-dimension specialist LLM evaluator executing dedicated expert rubrics and deep counterfactual analysis across multiple deliberation passes to capture subtle behavioral defects.

Uncertainty-Adaptive Review Adjustment. DynSTEER dynamically shifts evaluation weights and escalates judge levels toward an agent's vulnerable links. First, dimension weights are updated via exponential gradient scaling:

$$w_d^{(j+1)} = \frac{w_d^{(j)} \cdot \exp\left(\alpha(1 - s_d^{(j)}) + \beta u_d^{(j)}\right)}{\sum_{d' \in \mathcal{D}} w_{d'}^{(j)} \cdot \exp\left(\alpha(1 - s_{d'}^{(j)}) + \beta u_{d'}^{(j)}\right)} \tag{7}$$

where $s_d^{(j)} \in [0,1]$ is the dimension score, $u_d^{(j)} \in [0,1]$ is dimension uncertainty, F_j^{fatal} indicates fatal minefield evidence in stage j, and α, β scale sensitivity to quality deficits and uncertainty. Appendix D gives the design rationale and mathematical derivation of this update. Second, dimensions with low scores or high uncertainty receive increased attention in the next-stage evaluation, which is formalized as follows: the baseline judge level is:

$$\text{Level}_{\text{base}}^{(j+1)} = \begin{cases} J_{\text{cheap}}, & \text{if } S_j \geq \theta_{\text{pass}} \ \wedge \ \max_d u_d^{(j)} \leq \theta_{\text{low_u}} \ \wedge \ F_j^{\text{fatal}} = 0 \\ J_{\text{std}}, & \text{otherwise} \end{cases} \tag{8}$$

For each individual dimension d, its level $\text{Level}_d^{(j+1)}$ adaptively escalates along $J_{\text{cheap}} < J_{\text{std}} < J_{\text{exp}}$:

$$\text{Level}_d^{(j+1)} = \begin{cases} J_{\text{exp}}, & \text{if } s_d^{(j)} < \theta_{\text{fail}} \\ \min\left(J_{\text{exp}}, \text{Level}_d^{(j)} + 1\right), & \text{if } s_d^{(j)} < \theta_{\text{warn}} \vee u_d^{(j)} \geq \theta_{\text{high_u}} \\ \text{Level}_{\text{base}}^{(j+1)}, & \text{otherwise} \end{cases} \tag{9}$$

This focuses expensive LLM deliberation selectively on weak or ambiguous execution links while relaxing confident stages to rule-based verification to minimize unnecessary LLM invocations. Together with stage segmentation and local evidence isolation, this evaluation strategy resolves the evidence-granularity problem.

Execution-Time Early Termination. To prevent runaway executions on unrecoverable trajectories, DynSTEER monitors three policy stopping criteria. First, **Fatal Minefield Interception**, where $\phi^{\text{fatal}}(e_k) = 1$, signals immediately on the raw event clock upon hazardous or irreversible actions. Second, **Persistent Low Quality**, where $S_j < \theta_{\text{fail}}$, signals when accumulated stage deficits indicate unrecoverable goal drift. Third, **Frontier Stagnation** signals when an agent takes N_{stag} consecutive actions without advancing the ready frontier $\mathcal{F}_t$, detecting pathological stagnation. A stop emits an auditable report recording the step, dominator anchor, and failure reason.

4 Experiments

We evaluate DynSTEER on ToolSandbox and SWE-bench Pro using the same set of instruction-tuned agent backbones. The experiments address four central research questions. **RQ1**: whether stage-wise dynamic evaluation provides superior error focus and model discriminability over whole-trajectory evaluation. **RQ2**: how much dynamic judge levels, dynamic dimension weighting, and execution-time early stopping contribute to discriminability and efficiency. **RQ3**: whether milestone graphs compiled from task inputs reliably capture necessary goals while accommodating diverse solution paths. **RQ4**: how effectively execution-time early stopping identifies and removes wasteful execution prefixes.

4.1 Experimental Setup

Benchmark Environments and Evaluated Models. We evaluate on two benchmarks. **ToolSandbox** (Gu et al., 2024) is a stateful tool-use benchmark that scores complete trajectories through milestone and minefield checks. **SWE-bench Pro** (Deng et al., 2025) is a repository-level software-engineering benchmark that requires multi-file edits and test-based resolution. Appendix A reports the dataset statistics, and Appendix B details task filtering, baseline implementation, runtime parameters, and ablation variants. We evaluate **DeepSeek-V4-Pro**, **DeepSeek-V4-Flash** (DeepSeek-AI et al., 2024)[1], **Qwen3-Max-2026-01-23**, and **Qwen-Plus-2025-12-01** (Yang et al., 2024) under standard ReAct prompting (Yao et al., 2023). We set the evaluator LLM to Qwen3-Max-2026-01-23.

Comparative Paradigms. For controlled comparison, **Whole-Traj** and **DynSTEER** are evaluated on the same source trajectory. Whole-Traj treats the complete trajectory as one evaluation unit, whereas DynSTEER applies dynamic stage-wise evaluation to the same recorded trajectory. The details are specified in Appendix B.

Evaluation Metrics. We assess performance across three dimensions:

[1]The experiments use the post-2026-07-31 version of DeepSeek-V4-Flash.

Table 1: Main Evaluation Results on ToolSandbox and SWE-bench Pro.

Benchmark	Evaluator	Model	Mean Score ↑	Stop Rate	DS (ϵ=1.00) ↑	Saved Steps% ↑	Judge LLM Calls%
SWE-bench Pro	WHOLE-TRAJ.	DeepSeek-V4-Flash	69.54 ± 8.60	0.00%	0.0324	0.00%	–
		DeepSeek-V4-Pro	63.64 ± 8.93	0.00%		0.00%	–
		Qwen3-Max-2026-01	66.10 ± 4.15	0.00%		0.00%	–
		Qwen-Plus-2025-12	64.03 ± 3.96	0.00%		0.00%	–
	DYNSTEER	DeepSeek-V4-Flash	77.40 ± 5.50	6.12%	**0.0857**	12.19%	$9.60 \pm 1.12\%$
		DeepSeek-V4-Pro	69.90 ± 0.62	9.00%		10.78%	$6.80 \pm 0.37\%$
		Qwen3-Max-2026-01	64.93 ± 3.98	11.00%		6.64%	$9.64 \pm 0.47\%$
		Qwen-Plus-2025-12	61.87 ± 2.87	7.07%		2.38%	$10.14 \pm 1.06\%$
ToolSandbox	WHOLE-TRAJ.	DeepSeek-V4-Flash	78.80 ± 0.43	0.00%	0.0270	0.00%	–
		DeepSeek-V4-Pro	77.18 ± 0.14	0.00%		0.00%	–
		Qwen3-Max-2026-01	73.68 ± 1.00	0.00%		0.00%	–
		Qwen-Plus-2025-12	73.62 ± 1.09	0.00%		0.00%	–
	DYNSTEER	DeepSeek-V4-Flash	73.38 ± 0.26	16.63%	**0.0500**	11.44%	$17.97 \pm 0.66\%$
		DeepSeek-V4-Pro	71.11 ± 0.21	17.68%		16.06%	$20.69 \pm 0.41\%$
		Qwen3-Max-2026-01	65.03 ± 0.94	27.24%		21.54%	$20.03 \pm 0.20\%$
		Qwen-Plus-2025-12	66.15 ± 0.96	18.53%		22.14%	$21.33 \pm 0.18\%$

- **Metric Discriminability:** Following Qian et al. (2026), we measure Discriminability Score (**DS**). DS quantifies the ability to separate models by combining score dispersion with the fraction of model pairs whose mean scores differ by more than a margin:

$$\mathrm{DS}(\epsilon) = \left(\frac{\sigma_{\text{pop}}}{\mu}\right) \cdot \sqrt{\frac{N_{\text{sig}}(\epsilon)}{\binom{M}{2}}}, \quad N_{\text{sig}}(\epsilon) = \sum_{1 \le i < j \le M} \mathbb{I}\left(|\bar{s}_{m_i} - \bar{s}_{m_j}| > \epsilon\right) \tag{10}$$

  where μ and σ_{pop} are the mean score and standard deviation across models, and $N_{\text{sig}}(\epsilon)$ is the number of statistically distinguishable model pairs at margin ϵ. For readability, we normalize agent scores to $[0, 100]$ and report all DS results with $\epsilon = 1.00$.

- **Milestone Graph Quality:** On ToolSandbox, we compute Tool Operation Micro-F1 and Macro-F1 to measure semantic tool coverage, Fatal Minefield F1 for safety-critical action detection, and Graph Edit Distance (GED)-based similarity (Sanfeliu & Fu, 1983) for structural topology alignment.
- **Efficiency and Early Termination:** For each model, **Saved Steps%** is the percentage of matched Whole-Traj full-trajectory steps avoided by DynSTEER policy stops. **Stop Rate** divides policy stops by the matched evaluation population. For DynSTEER, **Judge LLM Calls%** is the ratio of stage-evaluation LLM calls to agent execution steps.

4.2 RQ1: STAGE-WISE DYNAMIC EVALUATION, ERROR FOCUS, AND MODEL DISCRIMINABILITY

Discrimination and efficiency. DynSTEER substantially enlarges the measurable capability margin: DS increases from 0.0270 to **0.0500** on ToolSandbox and from 0.0324 to **0.0857** on SWE-bench Pro, yielding relative gains of 85.2% and 164.0%. This change separates all six model pairs and raises the standard deviation of model means from 2.24 to 3.45 and from 2.34 to 5.87, respectively.

The pattern is consistent with stage-wise review: DynSTEER traces frontier progress and assigns lower stage-level evidence where an agent repeatedly fails tool-state transitions, violates safety constraints, or stalls without progress. Consequently, weak process behavior is no longer hidden behind an ultimately resolved outcome, and stronger models recover visibly from weaker ones in the score distribution. Because comparisons are paired on identical source trajectories, these gains reflect evaluator sensitivity to process quality rather than differences in the evaluated rollouts.

DynSTEER also removes 17.74% and 8.40% of Whole-Traj execution steps, achieving sharper discrimination while reducing evaluation execution cost. Additionally, compared with atomic-step evaluation, DynSTEER's judge LLM calls account for roughly 10–20% of total agent execution steps, indicating that it substantially reduces the cost of invoking LLM in evaluation.

Saved Steps vary substantially across backbones. Across benchmarks, model-level savings range from 11.44–22.14% on ToolSandbox to 2.38–12.19% on SWE-bench Pro. The narrower and lower SWE-bench Pro range reflects fewer policy stops in the smaller repository-level population, although individual stopped prefixes can still consume substantial execution budget. These results demonstrate that DynSTEER removes prefixes judged unrecoverable without collapsing model separation.

Table 2: Component Ablation on a ToolSandbox Subset.

Configuration	DS (ϵ=1.00) ↑	Stop Rate	Saved Steps%
WHOLE-TRAJ.	0.0228	0.00%	—
DYNSTEER	**0.0528**	**18.92%**	**12.90%**
-*w/o* dynamic weighting	0.0414	17.92%	12.75%
-*w/o* dynamic judge level	0.0439	15.83%	12.58%
-*w/o* dynamic	0.0311	15.83%	12.49%
-*w/o* policy stop	0.0286	0.00%	—

Table 3: Evaluation of simulated milestone graphs compiled by DynSTEER against the expert reference graphs on ToolSandbox.

Evaluation Dimension	Precision	Recall	F1 Score
Tool Operation (Micro)	84.0%	81.9%	82.9%
Tool Operation (Macro)	90.2%	88.4%	87.3%
Fatal Minefield Detection	91.8%	75.3%	82.7%
Structural Topology Metric		**Score**	
Topological GED Similarity		0.789	
Valid DAG Compilation Rate		100.0%	

Stability and rank consistency. The mean repeat-level standard deviation decreases from 0.66 to 0.59 on ToolSandbox and from 6.41 to 3.24 on SWE-bench Pro. Within each benchmark, DynSTEER preserves five of six pairwise model orders on ToolSandbox (Kendall $\tau = 0.67$) and four of six on SWE-bench Pro ($\tau = 0.33$); DeepSeek-V4-Flash remains first in both cases. In summary, DynSTEER offers greater model discriminability and more stable results across repeated evaluations.

4.3 RQ2: COMPONENT CONTRIBUTIONS TO DISCRIMINABILITY AND EARLY STOPPING

To answer RQ2, we conduct a component ablation on a ToolSandbox subset, and the sampling protocol and ablated variants are detailed in Appendix B. As shown in Table 2, removing any dynamic component or the early-stopping mechanism hurts the performance. Nevertheless, every ablated DynSTEER configuration remains more discriminative than WHOLE-TRAJ. This uniform improvement over whole-trajectory scoring indicates that stage-wise organization itself helps separate models and expose deficiency patterns, while the dynamic components further enlarge that margin.

Early stopping has the largest single-component impact. Disabling policy stopping reduces DS from 0.0528 to 0.0286. A plausible reason is that early stopping converts an unrecoverable execution prefix into decisive failure evidence. This binary signal complements graded stage scores: without it, a long routine suffix can dilute the score distribution and make systematically weak executions resemble recoverable noise.

Dynamic weighting and judge levels contribute complementary separation signals. Dynamically adjusting dimension weights makes models easier to distinguish: it reallocates evidence toward low-scoring or high-uncertainty dimensions and prevents uniform averaging from canceling complementary weaknesses. Dynamic judge levels nevertheless provide strong discriminability: relative to stage-only evaluation (-*w/o* dynamic), retaining judge-level escalation under -*w/o* dynamic weighting raises DS from 0.0311 to 0.0414. The early-stop statistics further separate the two mechanisms: -*w/o* dynamic weighting attains a higher Stop Rate and Saved Steps than -*w/o* dynamic judge level. This suggests that stronger review of poorly executed dimensions is more likely to expose problems and assign low scores promptly, allowing the agent to stop before additional wasted steps accumulate.

4.4 RQ3: MILESTONE GRAPH GENERATION RELIABILITY

Table 3 summarizes the semantic accuracy, structural distance, and safety alignment of generated milestone graphs against canonical expert annotations. DynSTEER's multi-candidate synthesis and majority consensus achieve an **82.9% Tool Operation Micro-F1** and an **87.3% Tool Operation Macro-F1**, extending reliable tool coverage beyond frequent operations to tail tools. Fatal Minefield

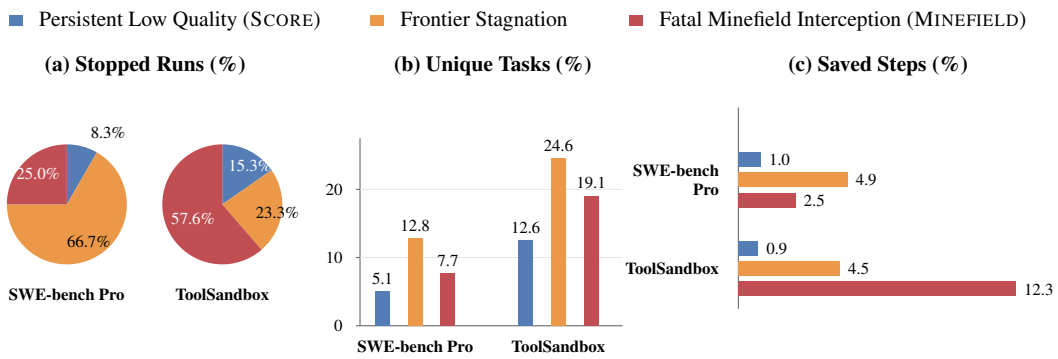

Figure 3: **Early-stop behavior and savings by termination criterion.** (a) Stopped-run shares, (b) unique-task coverage, and (c) Saved Steps contributions. Unique-task shares are not additive.

Detection reaches an **82.7% F1**, preferentially preserving irreversible operations from available task inputs. Transitive reduction retains alternative parallel branches without penalizing valid exploratory variations, such as searching by contact name rather than date, achieving a 100% valid DAG compilation rate with stronger topological alignment.

These expert-referenced results support using the compiled milestone graph as a principled basis for stage partitioning. The Macro-F1 (87.3%) and GED similarity (0.789) show that the compiled graphs capture tail tools and fine-grained topology. Nevertheless, the perfect valid-DAG rate and branch-preserving transitive reduction indicate that compilation yields stable checkpoints without collapsing alternative valid plans across diverse execution strategies.

4.5 RQ4: Early-Stop Behavior and Counterfactual Savings

Failure modes analysis. Figure 3(a) shows that Fatal Minefield Interception dominates ToolSandbox stopped runs at 57.56%, and Figure 3(c) shows its 12.34% contribution to the aggregate 17.74% Saved Steps. Its unique-task share is only 19.06%, whereas Frontier Stagnation covers the largest unique-task share (24.56%) but saves only 4.53%. Thus, ToolSandbox failures often occur early as unsafe or invalid calls on interactive tasks.

On SWE-bench Pro, Frontier Stagnation dominates stopped runs at 66.67% and unique-task coverage at 12.82%, whereas Fatal Minefield Interception accounts for 25.00% and 7.69%, respectively (Figure 3(a,b)). Frontier Stagnation contributes the largest saving, 4.86% of the aggregate 8.40%, reflecting prolonged repository exploration and repair loops before a terminal test result. Persistent Low Quality remains conservative, saving 1.00% on SWE-bench Pro and 0.87% on ToolSandbox.

Common task-level stagnation. Despite these distributional differences, Frontier Stagnation has the largest unique-task coverage on both benchmarks: 12.82% on SWE-bench Pro and 24.56% on ToolSandbox (Figure 3(b)). Its stopped-run share and contribution to savings still differ across datasets, but the pattern is consistent: stagnation arises in more distinct tasks than in either Persistent Low Quality or Fatal Minefield Interception. This indicates that the most prevalent risk is not confined to immediate safety violations or uniformly poor execution. Instead, agents on both benchmarks frequently enter extended progress vacuums in which local actions continue without advancing the task frontier, making frontier-aware execution-time review valuable in both tool-use and repository-level settings. In addition, Appendix E presents cases illustrating auditable stop decisions.

5 Conclusion

In this work, we propose **DynSTEER**, a dynamic stage-wise framework that advances LLM agent trajectory evaluation from passive, post-hoc terminal verification toward proactive execution review. DynSTEER segments rollouts at key execution nodes, adapts its multi-level review strategy to stage-level results, compiles a path-tolerant milestone graph without reference leakage, and terminates unrecoverable executions. Experimental results show that DynSTEER improves evaluation discriminability by over 85% compared with whole-trajectory evaluation and saves 17.74% of execution steps, providing a principled and cost-aware foundation for LLM-agent evaluation and optimization.

REFERENCES

Javed A Aslam, Virgiliu Pavlu, and Emine Yilmaz. A statistical method for system evaluation using incomplete judgments. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 541–548, 2006.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Deng, Chong Chen, et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, et al. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

Jian Gu, Shusen Jiang, Boyi Zhang, Zeqing Meng, Yifan Lu, Jiarui Gao, Yuchen Hu, Shengyu Zhang, and Fei Wu. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. *arXiv preprint arXiv:2408.04682*, 2024.

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 (ICLR)*, 2024.

Kevin Kuo, Ethan Anderson, and Dawn Song. Do agents dream of root shells? partial-credit evaluation of LLM agents in capture the flag challenges. In *IEEE/ACM International Conference on AI Engineering (AIWare)*, 2026.

Thomas Lengauer and Robert Endre Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1(1):121–141, 1979.

Meng Li, Yifan Sun, Dong Zhao, Yue Kang, and Xin Wang. TRAJECT-Bench: A trajectory-aware benchmark for evaluating agentic tool use. *arXiv preprint arXiv:2510.04550*, 2025.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hanghang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations (ICLR)*, 2024.

Shishir G Patil, Huanru Zhang, Zheyuan Yan, and Joseph E Gonzalez. The Berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *International Conference on Machine Learning (ICML)*, 2025.

Chen Qian, Yuxuan Dang, Jiaao Chen, Wei Liu, Yue Wang, Min Zhang, and Heng Ji. AgentProcess-Bench: Evaluating step-level effectiveness in tool-augmented agent trajectories. *arXiv preprint arXiv:2410.05284*, 2024.

Qi Qian, Chengsong Huang, Jingwen Xu, Changze Lv, Muling Wu, Wenhao Liu, Xiaohua Wang, Zhenghua Wang, Zisu Huang, Muzhao Tian, Jianhan Xu, Kun Hu, He-Da Wang, Yao Hu, Xuanjing Huang, and Xiaoqing Zheng. Benchmark^2: Systematic evaluation of LLM benchmarks. *arXiv preprint arXiv:2601.03986*, 2026.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations (ICLR)*, 2024.

Alberto Sanfeliu and King-Sun Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(3):353–362, 1983. doi: 10.1109/TSMC.1983.6313167.

Yuling Shi, Zhensu Sun, Junsen Dong, Chengcheng Wan, David Lo, and Xiaodong Gu. EarlyEval: Cheaper agent evaluation via early outcome prediction. In *Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE)*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 8634–8652, 2023.

Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2): 146–160, 1972.

Dingmin Wu, Ce Zhang, Chen Lin, Min Chen, and Rui Song. Agent-testing agent: A meta-agent for automated testing and evaluation of conversational AI agents. In *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*, 2026.

An Yang, Baosong Yang, Binyuan Zhang, Bofei Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

John Yang, Syeda Nahida Akter, Chou Han, Noah Shinn, Ofir Press, Karthik Narasimhan, and Shunyu Yao. InterCode: Standardizing and benchmarking interactive coding with execution feedback. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:15886–15915, 2023.

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

Yuxuan Zhang, Jiaqi Wang, Zihan Liu, Hao Chen, Bo Li, and Jing He. Claw-Eval: Towards trustworthy evaluation of LLM agents. *arXiv preprint arXiv:2604.06132*, 2026.

Daniel Zhou, Vikram Sreedhar, and Graham Neubig. AgentRewardBench: Evaluating automatic evaluations of web agent trajectories. *arXiv preprint arXiv:2504.08942*, 2025.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2024.

Table 4: **ToolSandbox Benchmark Dataset Statistics.** Distribution of evaluated scenarios across functional domains, interaction modalities, adversarial perturbation variants, and safety minefield configurations.

Scenario Domain / Sub-category	Count	Avg. User Turns	Available Tools	Minefield Invariants
Calendar & Scheduling Management	112	3.4	8 – 18	Data integrity, date bounds
Messaging & Notification Services	136	2.9	6 – 15	Privacy, unverified sends
Contact & Personal CRM	98	2.6	5 – 12	Irreversible deletion
Financial & Transaction Processing	84	4.1	10 – 24	Unauthorized transfer
Information Retrieval & Weather/Geocoding	79	3.2	7 – 16	Parameter type constraints
Perturbation & Difficulty Variants	**Count**	**Properties & Challenge Characteristics**		
Standard Canonical	127	Nominal parameters, standard system prompt and API schemas		
Distraction Tools (3–10 extra APIs)	158	Noise tools added to prompt to test agent tool retrieval		
Scrambled Arguments / Descriptions	142	Shuffled arg orders and adversarial field descriptors		
Insufficient Information (Rejection Tasks)	82	Ambiguous queries requiring clarification or safe refusal		
Total Evaluated Benchmark Pool	**509**	**3.2 turns**	**12.4 tools/task**	**704 total minefields**

Table 5: **SWE-bench Pro Public-Split Sampling Statistics.** The evaluated subset is stratified by repository to preserve the public test split's project diversity.

Repository	Public Split	Stratified Sample
NodeBB/NodeBB	44	6
ansible/ansible	96	14
element-hq/element-web	56	7
flipt-io/flipt	85	12
future-architect/vuls	62	8
gravitational/teleport	76	11
internetarchive/openlibrary	91	13
navidrome/navidrome	57	7
protonmail/webclients	65	9
qutebrowser/qutebrowser	79	11
tutao/tutanota	20	2
Total	**731**	**100**

A Benchmark Dataset Details

ToolSandbox. ToolSandbox (Gu et al., 2024) is an interactive, stateful benchmark engineered to evaluate complex multi-turn decision-making in tool-using LLM agents. Unlike static benchmarks that rely solely on mock API dictionaries, ToolSandbox maintains a stateful world environment with persistent entities, temporal relationships, and strict data invariants. Table 4 summarizes the evaluated scenario distribution.

SWE-bench Pro. SWE-bench Pro (Deng et al., 2025) supplies repository-level software-engineering tasks with Docker execution environments, gold patches, and test-based resolution. Its public test split contains 731 tasks from 11 repositories. Table 5 reports the repository-stratified sample used in our experiments.

B Detailed Experimental Configuration

This appendix documents task selection, the benchmark-specific WHOLE-TRAJ implementation, runtime parameters, and component ablations. All experiments use the four agent backbones in Section 4.1 and three repeats per task. DynSTEER evaluations replay the same complete source trajectories as WHOLE-TRAJ; a virtual stop marks a counterfactual prefix boundary and does not launch a separate rollout.

B.1 Task Selection and Filtering

ToolSandbox. The main experiment uses all 509 ToolSandbox scenarios backed by fully local, sandboxed execution environments rather than live RapidAPI endpoints. This selection prioritizes:

(1) reproducibility, because external endpoints introduce downtime, schema drift, network latency, and rate limits; (2) inspectable state snapshots and invariant audits, which are required for stage settlement and safety analysis; and (3) hermetic execution, which avoids external data leakage and uncontrolled API cost.

SWE-bench Pro. We start from the public test split with 731 tasks from 11 repositories and draw a repository-stratified sample of 100 tasks (Table 5). Task containers use only local Docker images or case-level cached images. When a required image cannot be loaded, the run is an infrastructure failure rather than evidence of agent capability; we remove the affected WHOLE-TRAJ/DynSTEER pair before computing score, stop-rate, DS, and early-stop statistics. This leaves 39 matched tasks. The retained set preserves repository stratification at sampling time but is smaller than the complete public split, which we report as a limitation.

Component-ablation subset. The ablation samples 100 ToolSandbox scenarios from the 509 main-experiment scenarios, stratified by task type and safety category. It uses the same four backbones, three repeats, source trajectories, and paired-comparison protocol as the main experiment.

B.2 WHOLE-TRAJ IMPLEMENTATION

WHOLE-TRAJ is the non-stage-wise comparison paradigm: it receives the complete trajectory as one unit and does not partition evidence by dominator milestones.

On ToolSandbox, it calls the benchmark-native trajectory evaluator over the recorded execution context. The evaluator compares milestone and minefield evidence exposed by the trajectory and returns a single similarity score, normalized to the common $[0, 100]$ reporting scale.

On SWE-bench Pro, the official benchmark protocol resolves a patch with tests but does not provide a native whole-trajectory evaluator. We therefore use an outcome-informed whole-trajectory audit. An evaluator LLM receives the task, repository context, complete command/result trajectory, final patch, and terminal patch/test outcome. It returns one holistic score on the common reporting scale that audits execution quality conditioned on the terminal outcome. This construction provides baseline access to both the full trajectory and the benchmark's terminal evidence while preserving single-unit evaluation granularity.

B.3 RUNTIME PARAMETERS

ToolSandbox agents use standard ReAct prompting, a maximum of 100 messages, and a fixed Qwen-Plus simulated user. SWE-bench Pro agents use a shell tool, at most one command per turn, at most 60 tool calls, a 600-second command timeout, and model temperature 0.0. Agent credentials are injected only into clients and are excluded from trajectories and released artifacts.

Milestone graphs are generated with Qwen3-Max-2026-01-23 at temperature 0.7, targeting 6 candidate graphs with at most 4 generation batches. LLM judges use the same fixed judge model, Qwen3-Max-2026-01-23, at temperature 0.1. Unless otherwise specified, stage thresholds are pass 0.80, warn 0.60, fail 0.40, low dimension uncertainty 0.20, high dimension uncertainty 0.45, fatal-minefield threshold 0.95, and threshold margin 0.05. Dynamic-weight sensitivities are $\alpha = \beta = 1.0$, and frontier stagnation fires after 16 observations without ready-frontier progress.

B.4 COMPONENT-ABLATION DEFINITIONS

All ablations retain the same source trajectories, milestone graphs, minefield checks, and models.

- ***-w/o* dynamic weighting:** dimension weights remain uniform, and adaptive judge-level routing and policy stopping remain enabled.
- ***-w/o* dynamic judge level:** adaptive routing is disabled, and every dimension uses the fixed cheap structural judge. Dynamic weighting and policy stopping remain enabled.
- ***-w/o* dynamic:** both adaptive routing and dynamic weighting are disabled. Evaluation uses uniform weights and the fixed cheap judge, while policy stopping remains enabled.
- ***-w/o* policy stop:** review and scoring are unchanged, but all stopping criteria are suppressed. Replay covers the full source trajectory, so Saved Steps is not applicable.

Algorithm 1 DynSTEER Execution-Time Stage Evaluation and Control

Require: Task inputs V_{task}, rollout events $\langle e_1, \ldots, e_K \rangle$, candidate count C, stop thresholds Θ
Ensure: Final evaluation report $\mathcal{R}$, termination status

$\{G_1, \ldots, G_C\} \leftarrow \text{SynthesizeCandidates}(V_{\text{task}}, C)$
$G \leftarrow \text{ValidateAndAggregate}(\{G_1, \ldots, G_C\})$ {Milestone DAG $G = (M, E, \mathcal{K}, \mathcal{M}_{\text{fatal}})$}
$M_{\text{settled}} \leftarrow \emptyset,\ T_D(s_0) \leftarrow 0,\ t \leftarrow 0,\ j \leftarrow 0,\ n_{\text{stag}} \leftarrow 0$
$\mathcal{R} \leftarrow \emptyset,\ \mathbf{w}_0 \leftarrow \text{InitWeights}(),\ \mathbf{L}_0 \leftarrow \{J_{\text{cheap}}\}_{d \in \mathcal{D}}$
for each raw event e_k in the rollout **do**
  if $\text{MatchMinefield}(e_k, \mathcal{M}_{\text{fatal}})$ **then**
    return $\text{Halt}(\mathcal{R}, \text{reason} = \text{"Fatal Minefield"}, k)$
  end if
  if e_k closes agent action a_t with observed state s_t **then**
    $t \leftarrow t + 1$
    $\mathcal{F}_t \leftarrow \{m \in M \setminus M_{\text{settled}} \mid \text{Pred}(m) \subseteq M_{\text{settled}}\}$
    $m^*, \mathcal{A}_t \leftarrow \text{MatchMilestone}(\mathcal{F}_t, a_t, s_t)$ {$\mathcal{A}_t$ stores ready-candidate attempt scores}
    $S_j, \mathbf{s}_j, \mathbf{u}_j, \mathbf{d}_j \leftarrow \text{None}$
    if $m^* \neq \text{None}$ **then**
      $\mathcal{W} \leftarrow [T_D(\text{idom}(m^*)), t]$ {Dominator-anchored evidence window}
      $S_j, \mathbf{s}_j, \mathbf{u}_j, \mathbf{d}_j \leftarrow \text{EvaluateStage}(\mathcal{W}, m^*, \mathbf{w}_j, \mathbf{L}_j)$
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{(m^*, S_j, \mathbf{d}_j)\}$
      $\mathbf{w}_{j+1}, \mathbf{L}_{j+1} \leftarrow \text{UpdateEvaluationPolicy}(\mathbf{w}_j, \mathbf{s}_j, \mathbf{u}_j, S_j, \Theta)$
      $j \leftarrow j + 1$
      if $\text{AcceptStage}(S_j, \mathbf{d}_j, \Theta)$ **then**
        $M_{\text{settled}} \leftarrow M_{\text{settled}} \cup \{m^*\},\ T_D(m^*) \leftarrow t,\ n_{\text{stag}} \leftarrow 0$
        $\mathcal{F}_t \leftarrow \text{AdvanceFrontier}(\mathcal{F}_t, m^*)$
      else
        $n_{\text{stag}} \leftarrow \text{UpdateStagnationWatch}(\mathcal{F}_t, \mathcal{A}_t, n_{\text{stag}}, \Theta)$
      end if
    else
      $n_{\text{stag}} \leftarrow \text{UpdateStagnationWatch}(\mathcal{F}_t, \mathcal{A}_t, n_{\text{stag}}, \Theta)$
    end if
    $z_t \leftarrow \text{CheckEarlyStopCriteria}(\mathcal{R}, \mathcal{F}_t, S_j, \mathbf{d}_j, n_{\text{stag}}, t, \Theta)$
    if $z_t \neq \text{None}$ **then**
      return $\text{Halt}(\mathcal{R}, \text{reason} = z_t.\text{reason}, t)$
    end if
  end if
end for
return $\text{FinalizeReport}(\mathcal{R}, M_{\text{settled}}, \text{status} = \text{"Completed"})$

C Algorithmic Formulations and Proof Details

C.1 DynSTEER Execution-Time Stage Evaluation and Control Algorithm

Algorithm 1 formalizes execution-time stage evaluation, dominator evidence-window extraction, uncertainty-adaptive judge-level adaptation, and early termination. The pseudocode separates the closed-action clock t from the evaluated-stage clock j: unmatched actions can still trigger frontier-stagnation monitoring without advancing the weight or judge-policy index.

The sentinel $S_j = \perp$ marks a closed action that produces no stage score; CheckEarlyStopCriteria then skips the low-quality criterion but can still fire frontier stagnation from $\mathcal{A}_t$ and n_{stag}. Thus every argument passed to the stop checker is defined. The map T_D records dominator settlement times, with the augmented entry s_0 anchored at zero. An accepted stage requires a ready-milestone match and successful hard-constraint/pass-threshold checks; only accepted stages advance M_{settled} and reset stagnation. Failed and unmatched attempts remain in the audit trail through $\mathcal{R}$ or $\mathcal{A}_t$, while UpdateStagnationWatch counts consecutive attempts without meaningful frontier-progress improvement.

C.2 Candidate Contract Validation

Before consensus, we deterministically check every candidate; we consult no LLM during this filtering step. **Schema validation** rejects malformed graphs and unknown operations: each milestone must have a unique identifier, a well-formed success condition, and a canonical tool name present in $\mathcal{A}$; minefield descriptors must likewise reference available destructive operations. It also checks that edge endpoints, dependency declarations, and output-field references exist.

Structural validation verifies that the directed precedence relation is finite, free of self-loops and cycles, and consistent with milestone reachability. Edges that duplicate an existing dependency are collapsed before aggregation, while genuinely distinct precedence constraints are retained.

Binding validation requires every dynamic parameter binding to name a milestone that can execute earlier in the same candidate, an output field declared by that milestone, and a target parameter whose schema accepts the referenced type. Literal arguments are checked against required keys and declared types. Thus, the system removes hallucinated APIs, malformed payloads, cyclic plans, and dangling or type-incompatible bindings before majority consensus.

C.3 Blueprint Consensus Aggregation Details

The blueprint aggregation pipeline synthesizes multiple candidate graphs $\{G_1, \ldots, G_C\}$ into a canonical Milestone DAG $G = (M, E, \mathcal{K}, \mathcal{M}_{\text{fatal}})$.

Let each candidate graph be represented by its set of nodes M_c, directed precedence edges E_c, and tool argument bindings $\mathcal{K}_c$. We define node semantic equivalence $\sim$ using canonical tool names and parameter role bindings:

$$m_i \sim m_j \iff \text{ToolName}(m_i) = \text{ToolName}(m_j) \wedge \text{Role}(m_i) = \text{Role}(m_j) \tag{11}$$

For each equivalence class $[m]$, its consensus support is:

$$\text{Support}([m]) = \sum_{c=1}^{C_{\text{valid}}} \mathbb{I}\left(\exists m' \in M_c \text{ s.t. } m' \in [m]\right) \tag{12}$$

Milestones with $\text{Support}([m]) \geq \lceil (C_{\text{valid}} + 1)/2 \rceil$ are admitted into the canonical milestone set M. Precedence edges are aggregated analogously, and transitive reduction $\text{TR}(E)$ is computed using Tarjan's linear-time topological ordering algorithm (Tarjan, 1972) to eliminate redundant shortcut edges.

C.4 Immediate Dominator Stage Anchoring Proof

Let $G = (M, E)$ be the DAG with augmented entry node s_0. A node d dominates node n (d dom n) if every directed path from s_0 to n contains d. The **immediate dominator** $\text{idom}(n)$ is the unique node that strictly dominates n without dominating any other strict dominator of n. We compute the dominator tree using the Lengauer-Tarjan algorithm (Lengauer & Tarjan, 1979) in $\mathcal{O}(|E|\alpha(|E|, |M|))$ time.

When milestone m^* is settled at step t, its stage evidence window $\mathcal{W}(m^*) = [t_{\text{idom}(m^*)}, t]$ encompasses precisely the intermediate steps executed between its immediate dominator and m^*. Because m^* could not have been reached without traversing $\text{idom}(m^*)$, all causal dependencies are preserved while concurrent, independent branches are cleanly partitioned.

D Theoretical Design of the Dynamic Weighting Mechanism

The evaluator must repeatedly direct attention toward dimensions whose recent evidence reveals either low quality or high uncertainty, while preserving positivity and comparability across stages. A normalized exponential tilt is a natural mechanism for this purpose: it turns each observed signal into a multiplicative potential rather than an ad hoc reassignment, so persistent evidence accumulates rather than gets overwritten. Starting from Eq. 7, the common factor $\exp(\alpha)$ cancels after normalization and gives

$$w_d^{(t+1)} \propto w_d^{(t)} \exp\Big(-\alpha s_d^{(t)} + \beta u_d^{(t)}\Big). \tag{13}$$

Thus, each stage multiplies a dimension's weight by the exponential of the dimension-specific potential $-\alpha s_d^{(t)} + \beta u_d^{(t)}$. Lower scores increase the potential's contribution to future attention, and higher uncertainty does the same.

Iterating this multiplicative rule makes the mechanism's cumulative nature explicit. Because normalization introduces only a dimension-independent constant, the ratio between any two dimensions after T stages is

$$\frac{w_d^{(T+1)}}{w_{d'}^{(T+1)}} = \frac{w_d^{(1)}}{w_{d'}^{(1)}} \exp\left[\alpha \sum_{t=1}^{T} \left(s_{d'}^{(t)} - s_d^{(t)}\right) + \beta \sum_{t=1}^{T} \left(u_d^{(t)} - u_{d'}^{(t)}\right)\right]. \tag{14}$$

Consequently, a linearly growing cumulative score deficit produces an exponentially growing weight ratio. This is the soft-min form of attention: the mechanism does not merely remember the latest weak dimension, but progressively reallocates review capacity toward persistent weaknesses.

To understand what such repeated tilting does to a model's aggregate score, first isolate the score component by freezing a profile $s = (s_1, \ldots, s_D)$. Define the partition function, tilted distribution, and tilted score by

$$Z(\lambda) = \sum_{d \in \mathcal{D}} e^{-\lambda s_d}, \qquad p_d(\lambda) = \frac{e^{-\lambda s_d}}{Z(\lambda)}, \qquad S(\lambda) = \sum_{d \in \mathcal{D}} p_d(\lambda) s_d. \tag{15}$$

The parameter λ represents the effective score sensitivity accumulated over repeated updates. Differentiating $\log Z$ gives

$$\frac{\partial}{\partial \lambda} \log Z(\lambda) = \frac{\sum_d (-s_d) e^{-\lambda s_d}}{Z(\lambda)} = -S(\lambda), \tag{16}$$

and differentiating once more yields

$$\frac{dS}{d\lambda} = -\frac{d^2}{d\lambda^2} \log Z(\lambda) = -\operatorname{Var}_{p(\lambda)}(s) \leq 0. \tag{17}$$

This equation contains both the direction and the rate of the score change. Every model's tilted score decreases as focus sharpens, but the decrease is proportional to the weighted variance of its own dimension profile. A flat profile changes little, whereas a profile with a distinct weakness changes sharply.

The next question is what information this tilt exposes. Decompose model i's profile into an average level and a zero-sum deviation:

$$s_i = m_i \mathbf{1} + v_i, \qquad \sum_{d \in \mathcal{D}} v_{i,d} = 0. \tag{18}$$

Uniform weighting retains only the first component:

$$S_i^{\text{avg}} = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} (m_i + v_{i,d}) = m_i. \tag{19}$$

It therefore cancels v_i, even though v_i records where a model succeeds and fails. In the sharp-tilt limit, only the weakest dimension receives weight, so

$$\lim_{\lambda \to \infty} S_i(\lambda) = \min_{d \in \mathcal{D}} (m_i + v_{i,d}) = m_i + \min_{d \in \mathcal{D}} v_{i,d}. \tag{20}$$

The second term is the depth of the model's worst weakness relative to its own average. For two models A and B, the limiting gap becomes

$$S_A(\infty) - S_B(\infty) = \underbrace{(m_A - m_B)}_{\text{average-level difference}} + \underbrace{\left(\min_d v_{A,d} - \min_d v_{B,d}\right)}_{\text{weakness-depth difference}}. \tag{21}$$

Hence the mechanism changes the comparison axis: uniform weighting compares average levels, whereas exponential tilting progressively adds the information carried by complementary failure patterns. For example, $s_A = (0.9, 0.3)$ and $s_B = (0.5, 0.9)$ give a uniform gap of 0.100, but

the limiting gap becomes 0.200 even though both tilted scores decrease. Both models are pulled downward, while the model with the deeper weakness is pulled down faster.

The same reasoning also clarifies when separation is guaranteed. Let $\mathcal{I}$ denote the model set and define the cross-model dispersion

$$\mathrm{Dis}(\lambda) = \sum_{i \in \mathcal{I}} \big(S_i(\lambda) - \bar{S}(\lambda)\big)^2, \qquad \bar{S}(\lambda) = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} S_i(\lambda), \tag{22}$$

Using Eq. 17,

$$\frac{d\,\mathrm{Dis}}{d\lambda} = 2 \sum_{i \in \mathcal{I}} \big(S_i - \bar{S}\big)\, \frac{dS_i}{d\lambda} = -2 \sum_{i \in \mathcal{I}} \big(S_i - \bar{S}\big)\, \mathrm{Var}_{p_i}(s_i). \tag{23}$$

where p_i denotes the tilted distribution induced by Eq. 15 for model i's profile s_i.

Dispersion increases when models below the current tilted mean have larger internal variances, that is, when their disadvantages are concentrated in deeper weak links. If instead a high-average model also contains the deepest weakness, this derivative can become negative, and the gap can compress. The mechanism therefore guarantees a principled shift of evidence toward weak links, not unconditional divergence between all model pairs.

The uncertainty term now completes the design. In Eq. 14, $\beta u_d^{(t)}$ contributes another dimension-specific potential: uncertain evidence receives attention before it is discarded, allowing the next review level to test an apparent weakness with stronger evidence. Once that evidence is resolved, subsequent updates respond to the refined score and uncertainty estimates. Taken together, the score tilt supplies the comparative objective—moving from average performance to weakness depth—while uncertainty controls the epistemic allocation of review capacity during that transition.

E Qualitative Case Studies and Error Analysis

E.1 Case 1: Early Termination of Infinite Search Frontier Stagnation

Scenario ID:
`search_message_with_recency_oldest_multiple_user_turn_alt_3_distraction_tools_arg_type_scrambled`

Agent Model: Qwen3-Max-2026-01 under ReAct prompting.

Execution Trace Summary:

- The user asks for the oldest matching text. The agent asks one clarification, learns that the oldest message is required, and then obtains the current timestamp; milestone m_0 settles with score 1.0.
- The next milestone, retrieving the target message, stays ready but never improves: its best stage score is 0 for eight consecutive observations.
- DynSTEER therefore emits a frontier-stagnation stop at replay observation 43 and maps that decision to the source execution boundary at batch 22.
- **Replay accounting:** the complete source trajectory records 29 execution batches and 41.04s of execution latency; the matched prefix records 22 batches and 24.68s. Stopping there would save 7 batches and 16.36s of execution latency. These are paired counterfactual savings, not a separately executed live interruption.

E.2 Case 2: First-Error Localization vs. Fluke Success

Scenario ID:
`add_reminder_content_and_date_and_time_10_distraction_tools`

Agent Model: DeepSeek-V4-Flash.

Execution Trace Summary:

- The agent was tasked with setting a reminder with a specific date and time.

- In intermediate steps, the agent called an invalid distraction API `set_calendar_alarm` with malformed argument types, which failed.
- The agent then stumbled upon `add_reminder` after multiple trial-and-error retries.
- **Whole-Traj Evaluation:** Scored the complete trajectory as 1.0000 because the required milestones were ultimately resolved, while ignoring the 4 erroneous intermediate API calls and high token overhead.
- **DynSTEER Stage Review:** In the DeepSeek-V4-Flash run, the stage reaching m_0 scores 0.897, whereas the subsequent target-message stage falls to 0.272 and triggers the low-score policy. DynSTEER therefore preserves the distinction between clean execution and an error-prone, inefficient execution.

F Prompt Templates for Milestone Generation and Stage Judges

This section documents excerpts from the primary English prompt templates used by DynSTEER. The excerpts follow the implementation files closely; for long rule lists, we show only representative head/tail rules, and we denote omitted rules with ....

Prompt Template: Standard Judge Unified Multi-Field Prompt

You are a strict task-trajectory evaluator. Evaluate only the supplied task, stage interval, constraint_checks, and trajectory steps. Do not invent external facts.

Context fields:
- `stage_goal` is the current stage success condition; `task.task_description` is only task background.
- `rubrics` contains only the dimensions to score in this pass; `dimension_scores` must contain only those dimensions.
- `steps` are the primary behavioral evidence. Cite step index, actor, event_type, tool_call, tool_result, or raw sender/recipient.
- `constraint_checks` are auxiliary structured scorer evidence and must not replace auditing steps.
- `interval` provides the evaluated step range, status, milestone_score, and milestone evidence.
- `required_output` defines the exact JSON output shape. Do not output `stage_score`; code computes it from `dimension_scores`.

Evaluation objective:
- Decide whether this stage satisfies `stage_goal` within the supplied interval.
- Score only the dimensions present in `rubrics` from 0 to 1.
- Ground every score in evidence and distinguish completed, partially completed, merely claimed, and unsupported work.

Evidence rules (excerpt):
- Every evidence item must reference a step index, `interval.evidence` item, or `constraint_checks` item.
- If a judgment cannot be grounded in Context, explain the evidence gap in diagnosis and lower the relevant dimension score.
- For `state_snapshot` evidence, `constraint_checks` may serve as state evidence; user-facing communication quality must still be audited from steps.
- ...
- Do not use external facts or assumptions outside Context.

Return exactly one JSON object matching `required_output`. Do not wrap it in prose.

Context: {`context_json`}

Prompt Template: Template of Expensive Specialist Judge Prompts

You are the specialist LLM-Judge for the {`dimension`} dimension.
Score only {`dimension_specific_focus`}.

Focus on:
• {`Dimension-specific forensic criteria`}.

Evaluation steps:
1. Dimension-specific evidence-audit steps.
2. ...
3. The `dimension_scores` field may contain only {`dimension`}.
Use `Context.rubrics.`{`dimension`} as the score anchor. Return exactly one JSON object matching `required_output`.

Context: {`context_json`}

Prompt Template: Milestone Graph Candidate Generation Prompt

System Prompt:
You generate candidate milestone goal graphs, not step-by-step trajectories. A node belongs in one candidate graph only when, under that candidate's complete and feasible interpretation, every valid completion must reach that operation or goal. One candidate is a hypothesis about necessity; necessity across candidates is decided later by code aggregation.

User Prompt Input:
This is candidate batch {`batch_index`}. Focus code: {`focus_code`}.
Batch focus: {`focus_instruction`}
Return exactly {`num_graphs`} complete, standalone candidate graphs for the current task.

Independence and diversity rules:
- Each graph must define its own dispositions, nodes, edges, and minefields. Local IDs have graph-local scope only.
- Seek genuine diversity between two complete judgments: a different visible tool, direct use of public information, a valid bulk state goal, a different necessary dependency, or a justified executability judgment.
- ...
- If no genuine alternative is supported, the two graphs may be equivalent. Completeness and correctness take priority over artificial difference.

Current public task JSON: {`task`}

Required semantics (excerpt):
- **Necessity, feasibility, and goals:** Keep only milestones that are necessary under this graph's complete judgment. ... When information is insufficient, mark only the real terminal tool or critical producer needed by the user request; never guess the target object.
- **Node schemas and provenance:** The only allowed node kinds are `tool_call`, `set_state`, and `emit_message`. A `tool_call` node has exactly `local_id`, `turn_id`, `kind`, `evidence_id`, and `arguments`. ... Never hard-code or copy a dynamic UUID, row ID, timestamp, coordinate, token, or tool-produced value as `public_literal` unless that exact value is explicitly visible in a permitted current source.
- **Edges:** Add an edge only for a true non-commutable data, prerequisite, or goal dependency. ... No self-loop, duplicate edge, missing endpoint, or cycle is allowed.
- **Minefields:** A minefield marks a fatal wrong tool call that must not occur for the stated turn. Use only severity `fatal` and the four listed reason codes. ...
- **Exact response shape:** Return one JSON object with exactly one top-level key, `graphs`; `graphs` must contain exactly {`num_graphs`} graph objects. Every graph object has exactly `dispositions`, `nodes`, `edges`, and `minefields`. Return raw valid JSON only.

Final response skeleton:
```
{"graphs":[{"dispositions":{},"nodes":[],
"edges":[],"minefields":[]}, {"dispositions":{},"nodes":[],
"edges":[],"minefields":[]}, ...]}
```

AI USE STATEMENT

In this work, we used generative AI tools to polish and refine the writing, check grammar, implement methods, assist with mathematical derivations, and assist with translation.

Additionally, we used generative AI tools to draft figures and parts of the paper, summarize and analyze existing literature, search for information, and suggest possible titles and keywords.

We did not use generative AI tools to generate research ideas or conduct data analysis, and the remaining required disclosure tasks do not apply to this work.

We reviewed all AI-assisted work and carefully checked and verified all LLM-polished content.

We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.