TRACE: Why LLM Agents Fail in Multi-Step and Multi-Turn Environments?

Anonymous Authors[1]

Abstract

LLM-based agents have made remarkable progress in performing complex tasks across multi-turn interactions and in environments that require multi-step reasoning. Although recent advances have enabled more dynamic interactions with users and environments, these models still fail in ways that are not well defined and understood. In this paper, we discuss what agent abilities are lacking when LLMs fail in multi-turn settings. We perform a comprehensive analysis of around 10,000 trajectories spanning four popular agentic benchmarks: **τ^2-bench, BFCL v4, ALFWorld, and ACEBench** and seven LLMs covering four closed-source models: **GPT-5.4 Mini, DeepSeek-R1, Kimi-K2.5, and Llama-3.3-70B-Instruct-Turbo** and three open-source models at different scales: **Qwen3 (1.7B, 4B, 8B)**. We introduce TRACE, a capability-based taxonomy that organizes failures into three categories: A action–plan failures, B monitoring failures, and C outcome evaluation failures. Each failure type is mapped to specific agent skills, such as self-awareness, self-correction, proactivity, state-tracking, and policy adherence. We show that certain failure patterns recur across benchmarks and model families and can be attributed to poor agent capabilities. Our analysis introduces a principled taxonomy of the failure patterns and highlights the need for training methods that provide fine-grained, step-level feedback to LLM agents to improve these capabilities in multi-turn and multi-step reasoning environments.

1. Introduction

The emergence of methods such as Reinforcement Learning (RL) and Chain-of-Thought (CoT) reasoning has allowed LLM agents to make rapid progress in the performance of complex tasks, which has led them to gain considerable attention in the AI community (Cemri et al., 2025; Acikgoz et al., 2025b; Guo et al., 2025). Despite the advances, LLM agents still exhibit certain failure patterns that are not well understood. We present an empirical analysis of around 10,000 trajectories covering multiple benchmarks and model families, and show that these failures recur in consistent, identifiable patterns.

We study agent behavior in two different settings: multi-step and multi-turn. *Multi-step reasoning* refers to the execution of multiple, dependent actions within a single turn. These actions may include tool invocations or intermediate reasoning steps. *Multi-turn interaction*, on the other hand, refers to multiple turns of interaction with a user or the environment. Both settings come with their own set of challenges: multi-step reasoning requires correct tool invocation and planning, whereas multi-turn interaction requires self-correction and state-tracking. We conduct our analysis on benchmarks that cover all three cases: only multi-step tasks, only multi-turn tasks, and tasks that cover both.

We argue that our identified failure patterns are a reflection of poor agent capabilities and outcome evaluation, and we provide evidence that these capabilities are not consistently acquired using existing training techniques (Xu et al., 2025; Qian et al., 2025; Rawat et al., 2025). Building on this, we introduce a principled taxonomy of failure modes to get a better understanding of agent behavior in multi-step and multi-turn environments. To structure this analysis, we ask three key questions:

- Do multi-step and multi-turn agent failures form stable, identifiable patterns across models and benchmarks?
- Do these patterns persist across different training paradigms, including both RL-trained and non-RL agents?
- Can differences in training dynamics explain why these patterns emerge?

Although prior work has analyzed failure patterns in LLM-based systems (Cemri et al., 2025; Various, 2025a; Wang et al., 2025; Song et al., 2026), these efforts primarily focus on either multi-agent coordination or reasoning failures in non-interactive settings. As a result, they are unable to provide a detailed analysis of the failures that emerge in

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

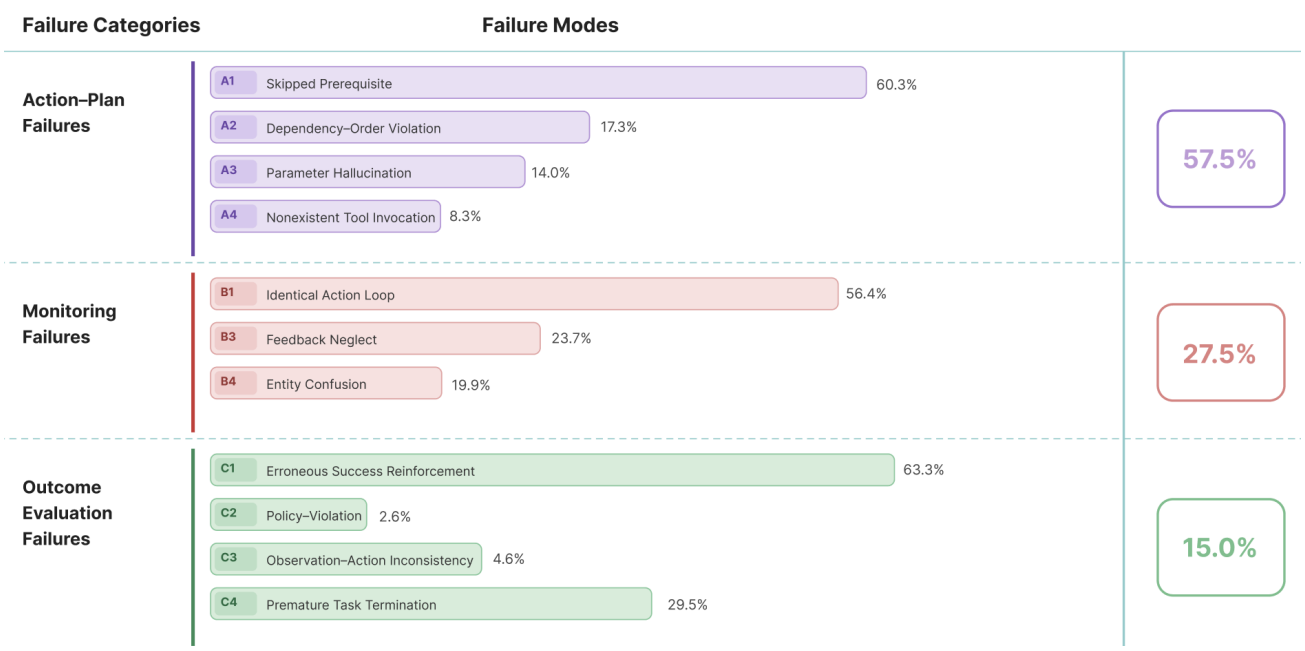

Figure 1. **A Taxonomy of Failure Modes**

single-agent settings that involve both multi-step reasoning and multi-turn interactions, which remain a major bottleneck for advancing long-horizon agentic tasks and user-oriented conversational domains (Acikgoz et al., 2025b).

In this work, we address this gap by introducing a capability-based taxonomy that organizes failure modes into three categories: (A) action–plan failures, (B) monitoring failures, and (C) outcome evaluation failures. We show that certain failure patterns recur across benchmarks and model families and can be attributed to poor agent capabilities. Our analysis introduces a principled taxonomy of the failure patterns and highlights the need for training methods that provide fine-grained, step-level feedback to LLM agents to improve these capabilities in multi-turn and multi-step environments. The contributions of this paper are as follows:

- We perform an empirical analysis of approximately 10,000 trajectories across multiple benchmarks and models, identifying recurring failure patterns in LLM agents.
- We show that these failures can be explained as poor agent capabilities, and introduce a capability-grounded taxonomy that organizes failure modes and maps them to underlying agent skills.
- We demonstrate that these failure patterns persist across models and training techniques, highlighting the limitations of current techniques for training LLM agents.

2. Related Work

Error Analysis and Taxonomies for LLM Agents: Prior work has analyzed failure patterns in multi-agent coordination systems, in RL-trained agent behavior and in reasoning tasks. Cemri et al. (2025) propose a taxonomy of failures in multi-agent systems, identifying coordination and communication breakdowns. RAGEN (Wang et al., 2025) identifies phenomena known as *echo trap*, where similar trajectories reinforce each other during multi-turn self-improvement.

However, these taxonomies are descriptive and do not connect these failures to the underlying training techniques. Thus, it is not possible to understand whether these errors arise due to model or benchmark limitations or fundamental properties of the underlying training methods.

However, our taxonomy links failure patterns to training mechanics. This helps us identify failures that are consistent across models and benchmarks, and helps us understand how we can improve existing training techniques.

Reward-Based RL for LLM Agents: Recent work has shown that Reinforcement Learning (RL) can boost LLM performance in tool-calling and reasoning. (Shao et al., 2024) introduced Group Relative Policy Optimization (GRPO) that demonstrated reduced memory overhead during training as it replaced the critic model in PPO with group relative advantage estimation. DeepSeek-R1 (Guo et al., 2025) scaled GRPO to train large reasoning models using Reinforcement Learning with Verifiable Rewards

ID	Error Type	Implicated Skill(s)	Dimension
A1	Skipped prerequisite	Planning, Policy following	Reasoning Control
A2	Dependency-order violation	Planning, Agentic reasoning	Reasoning
A3	Parameter hallucination	Tool execution, Self-awareness	Control Monitor
A4	Nonexistent tool invocation	Tool selection, Self-awareness	Control Monitor
B1	Identical-action loop	Self-correction	Monitor
B3	Feedback neglect	Self-correction, State tracking	Monitor
B4	Entity confusion	State tracking	Monitor
C1	Erroneous success reinforcement	Self-awareness, Self-correction	Monitor
C2	Policy-violation masking	Policy following, Self-awareness	Control Monitor
C3	Observation–Action inconsistency	State tracking, Self-awareness	Monitor
C4	Premature task closure	Proactivity, Self-awareness	Control Monitor

Table 1. **Mapping from error types to implicated agent skills and the desideratum dimension each skill belongs to**

(RLVR), showing that RL can be used in place of supervised fine-tuning for reasoning chains.

In the context of conversational agents, many works have introduced structured and specific reward designs. ToolRL (Qian et al., 2025) showed that granular rewards improved tool invocation compared to coarse outcome rewards. They proposed a framework with fine-grained rewards that included correct function names, accurate parameter names and parameter values. SpeakRL (Acikgoz et al., 2025a) showed how carefully designed, granular format and outcome rewards can train conversational agents in asking clarifying questions in multi-turn conversations. AgentRL (Xu et al., 2025) trains agents across different tasks asynchronously and normalizes the rewards for each task so that no task contributes more than the other. Rawat et al. (Rawat et al., 2025) showed that combining rewards for correct tool use and correct final answers improves the co-ordination between reasoning and actions in multi-turn conversational agents.

Although these methods improve the empirical performance of agents, they require task-specific reward engineering. Thus, they are unable to explain why agents fail in these environments and cannot identify recurring failure patterns across models and benchmarks.

Turn-Level Credit Assignment: A central limitation in RL for LLMs is their scalar reward signal and credit assignment across long trajectories. Zeng et al. (Zeng et al., 2025) proposed rewards at the turn level in multi-turn environments. The method proposed using verifiable and LLM-as-a-judge rewards for the reasoning tasks. A concurrent practitioner's guide (Various, 2025c) decomposed RL for multi-turn environments into reward and policy pillars, and evaluated token-level credit assignment on TextWorld and ALFWorld. VinePPO (Kazemnejad et al., 2025) proposed doing away with learned value functions. Instead, it simulated Monte Carlo rollouts from the intermediate reasoning states and estimated their value using the average reward of the sampled continuations. OPRL (Various, 2025b) proposed learning a process reward model from agent trajectories during training.

Since these methods do not rely on scalar reward signals, they are able to mitigate the sparsity of trajectory level rewards. However, even these works require task-specific reward engineering and are reliant on learned reward functions. They construct denser reward signals, but do not identify when and why certain failures occur during training. Thus, these methods propose fixes without a principled taxonomy of the underlying failures that they aim to address.

3. Error Taxonomy for Agents

We present a taxonomy of recurring failure modes observed in agent trajectories across three benchmarks: **τ^2-bench** (retail and airline domains), ALFWorld, AceBench and BFCL. Rather than attributing errors to a single training paradigm, we characterize each failure in terms of the *agent-level skill* it implicates. We organize skills along three dimensions following the Reasoning–Monitor–Control desideratum Acikgoz et al. (2025b): Reasoning encompasses agentic reasoning and planning; Monitor covers self-awareness, self-correction, and state tracking; and Control spans tool selection, tool execution, policy following, and proactivity.

Table 1 provides a summary mapping between error types and implicated skills.

3.1. Family A: Action–Plan Failures

A These errors arise when the order of the agent's executed actions does not align with the task requirements. In other words, the agent's actions may be valid in isolation but their order of execution is not. The primary skill deficits are in *planning*, *tool selection* and *tool execution*, and *policy*

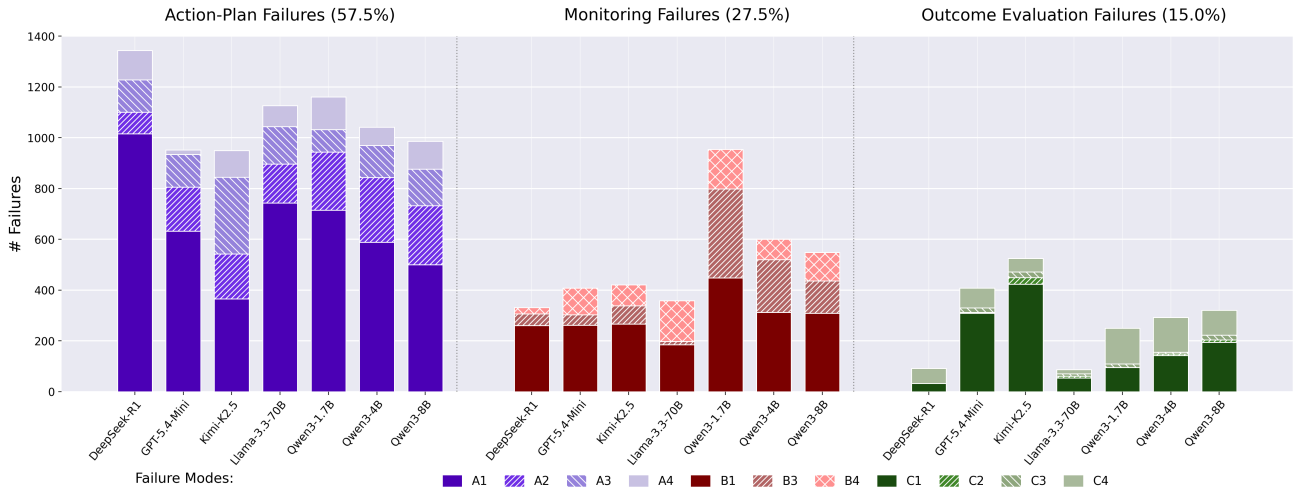

Figure 2. **Distribution of Failures**

adherence.

1 A1: Skipped Prerequisite:

The agent skips the prerequisite steps, causing errors in downstream actions. This is because one or more preconditions for the latter actions are not satisfied.

In **BFCL**, we observe such errors in the `miss_func` error category. A function such as `turn_on_wifi()` is required before subsequent network calls, but the agent omits it entirely and proceeds. This is also observed in the retail domain of **τ^2-bench**, wherein the agent has a policy to authenticate the user by invoking the `find_user_id_by_name_zip` tool call. However, it skips the authentication call and directly queries the order details. We observed this omission in over 60% of Qwen3-1.7B retail trajectories.

This error reflects a planning deficit: the agent fails to construct a complete action plan that includes all mandatory preconditions.

2 A2: Dependency-Order Violation:

All required actions are present, but they are executed in an order that violates a dependency. Due to this, a later step fails even though each individual action looks reasonable on its own.

A more representative example in the retail domain of **τ^2-bench** is when the agent selects an order workflow before verifying that the order state supports that workflow. For example, the agent sometimes calls `modify_pending_order_items` on an order that has already been delivered. This is invalid because delivered items can no longer be modified through the pending-order tool; instead, the agent should begin the exchange workflow for that item. This shows that the model identifies a relevant action, but applies it before the required state condition has been established.

This error is different from A1 in that no action is missing. Instead, the agent identifies relevant actions but executes them before the state of the task has been properly established.

3 A3: Parameter Hallucination:

The agent selects the correct tool, but supplies one or more parameters that are unsupported or not grounded in the available context.

In **τ^2-bench**, this sometimes appears when the model adds a `user_id` argument to tools such as `get_order_details` or `get_reservation_details`, even though those tools do not accept it. In **BFCL**, the same broader behavior corresponds to the `miss_param` error category.

This failure reflects a self-awareness gap: the agent does not recognize that the required parameter is absent from its context and should be obtained via a prior tool call or user query, and instead confabulates a plausible-looking value.

4 A4: Nonexistent Tool Invocation:

The agent makes a tool call that does not exist in the environment's action space.

In **τ^2-bench**, this failure often appears when the agent invents helper tools for requesting missing information, even though those tools are not part of the benchmark's

action space. For example, in the airline domain, the model sometimes calls `ask_for_reservation_id` or `ask_for_user_id`, even though the environment does not provide such tools. Instead of using the normal dialogue channel to ask the user for the missing information, the agent invokes a tool that sounds plausible but is unavailable.

This error reveals a failure in self-awareness regarding the agent's own capabilities. The agent does not verify that a tool exists before invoking it. Instead, the agent hallucinates tool names.

3.2. Family B: Monitoring Failures

B These errors occur when the agent fails to monitor its own behavior and the interaction state. The core deficits lie in *self-correction*, *state tracking*, and *self-awareness*.

1 B1: Identical-Action Loop:

The agent repeats actions with identical parameter values despite receiving failure signals.

We observe this pattern across both **τ^2-bench** and **BFCL**. In the airline domain of **τ^2-bench**, one specific example is when the model repeatedly calls `get_reservation_details` with the same reservation ID even after the environment has already returned the same reservation information. In **BFCL**, the same broader pattern appears in force-terminated traces, where the model repeats non-progressing tool behavior without changing its strategy. In both cases, the repeated action does not move the task forward.

The agent lacks the self-corrective ability to identify failed actions and understand that repetition will not lead to a different outcome.

3 B3: Feedback Neglect:

The agent cannot condition its belief state and future actions on evidence of failure provided by the environment. The evidence may include error messages, or unexpected states, but the agent fails to use these to update its belief state.

This failure appears across both **τ^2-bench** and **BFCL**. In **τ^2-bench** retail Task 37, the model receives the explicit error message "The new item id should be different from the old item id," but still issues `modify_pending_order_items` again instead of changing its strategy immediately. In **BFCL** `multi_turn_long_context_1`, the agent shows the same tendency: after moving `log.txt` into the `archive/` directory, it receives feedback showing that its grep-based approach is failing, but continues with the broken strategy instead of first changing into the correct directory and updating the workflow.

This failure highlights the agent's poor ability to self-correct. It does not update its belief state on the basis of environmental feedback.

4 B4: Entity Confusion:

The agent cannot track which entity is referred to in an action, resulting in incorrect and inaccurate actions being taken.

In **ACEBench** `agent_multi_turn_8`, the error begins when the agent maps the request onto the wrong person: it treats `Frank` as the relevant user and asks for Frank's credentials, even though the workflow should involve someone else. The same underlying problem appears in **τ^2-bench** retail Task 59, but at the order level. There, the agent loses track of which user is associated with which order, leading to a 27-turn loop of actions grounded in the wrong entity.

This failure demonstrates a failure in state tracking. The agent's belief state regarding which entities are being referenced, modified, or discussed gets corrupted over the course of the interaction.

3.3. Family C: Outcome Evaluation Failures

C These errors arise because the agent is unable to critically evaluate its own outputs and actions. An important feature of this family of errors is that the agent reaches a terminal state without recognizing what has gone wrong in its intermediate steps. These failures primarily highlight drawbacks in an agent's *self-awareness*, *policy adherence*, and *proactivity* skills.

1 C1: Erroneous Success Reinforcement:

The agent takes one or more incorrect or redundant intermediate actions, but the trajectory still reaches a correct final outcome. As a result, those earlier mistakes are never flagged or corrected at the trajectory level.

A brief example appears in **τ^2-bench** retail Task 48, where the agent successfully processes the return of one item, but then also attempts to return a second item that is not yet eligible for return. The extra return attempt fails, but the overall trajectory still reaches a successful outcome, so the incorrect intermediate action is not penalized at the trajectory level.

This failure highlights a gap in self-awareness. The agent cannot retrospectively evaluate whether its intermediate steps were correct, but can only determine whether the trajectory succeeded or failed.

2 C2: Policy Violation:

The agent violates an explicit environment rule, but still reaches a correct final outcome. As a result, the violation is

never penalized.

In **τ^2-bench** retail Task 16, the violation occurs during a multi-step order update. The model recognizes that each order change should receive separate confirmation, but still uses one bundled confirmation to cancel two pending orders and return one delivered item. Because the requested final outcome is achieved, the evaluation rewards the trajectory even though the agent bypasses the required confirmation policy.

This failure demonstrates that agents can learn to complete tasks successfully through shortcuts and policy violations that would not be acceptable in real-world deployment.

3 C3: Observation–Action Inconsistency:

The agent's assumptions and current belief state contravene the evidence returned by its own tool calls.

A clear example appears in **τ^2-bench** Airline Task 5. The user repeatedly claims to be a Gold member, but `get_user_details` returns `membership: regular`. The model then calls `transfer_to_human_agents` with a summary that describes the user as a Gold member. The next action therefore follows the user's unsupported claim rather than the tool evidence.

This failure highlights state tracking and self-awareness: the agent does not detect that its planned action contradicts the environment observation it just received.

4 C4: Premature Task Termination:

The agent concludes interactions before all sub-tasks have been addressed and without verifying task completeness.

A clear example appears in **BFCL** `multi_turn_base_46`, where the user asks the agent to delete all files in the `Drafts` directory and then remove the directory itself. The model responds that the task cannot be completed, even though the available tools are sufficient to finish it step by step. A similar supporting pattern appears in **ACEBench**, where some multi-turn trajectories produce no actionable progress even though the remaining task can still be completed through the available process steps. In both cases, the interaction is closed too early even though a valid plan still exists.

A proactive agent would verify whether all user requirements were completed before concluding the interaction.

4. Experiments

4.1. Benchmarks and Setup

We analyze agent behavior in four benchmarks: τ^2-bench, BFCL v4, ALFWorld, and ACEBench. The τ^2-bench consists of multi-step reasoning and multi-turn interactions, where the LLM agent makes tool calls under policy constraints in the retail and airline domains. BFCL v4 evaluates multi-step reasoning using function calling tasks with explicit dependencies between actions. ALFWorld tests the LLM agents on long-horizon embodied tasks that require sequential decision-making and interaction with a dynamic environment. ACEBench evaluates agents in multi-turn settings, where the focus is on conversational and interactive settings.

We evaluated seven LLMs spanning different model families and scales, including closed-source models (GPT-5.4 Mini, DeepSeek-R1, Kimi-K2.5, Llama-3.3-70B-Instruct-Turbo) and open-source models (Qwen3-1.7B, 4B, 8B). We analyze around 10,000 trajectories generated in these benchmarks, and each trajectory is labeled with the 11 failure sub-pillars of TRACE using an LLM-based judge, allowing consistent identification of failure patterns. This checks whether the identified failure patterns are consistent across different environments, model families, and model scales.

4.2. Which failures dominate agent behaviour?

TRACE organizes failures into three categories: A action–plan failures, B monitoring failures, and C outcome evaluation failures.

What we find: Across all annotated trajectories, A Action–Plan failures comprise 57.5% of all observed failures, more than twice the rate of B Monitoring failures (27.5%) and nearly four times that of C Outcome Evaluation failures (15.0%). Within A, one pattern dominates: A1 (Skipped Prerequisite) alone accounts for 60.3% of all Action–Plan failures (Figure 1).

Interpretation: One of the most important observations we make is that agents fail because they omit steps that should have been part of the plan. This concentration of errors into a small number of recurring patterns suggests that failures are driven by systematic weaknesses rather than noise. In particular, the prevalence of skipped prerequisites points to a breakdown in planning and dependency reasoning: agents fail to construct complete action sequences that satisfy required preconditions.

4.3. The dominant failures recur across model families

Figure 2 breaks down the 11 sub-pillars across the seven models we evaluate. The relative ordering of sub-pillars

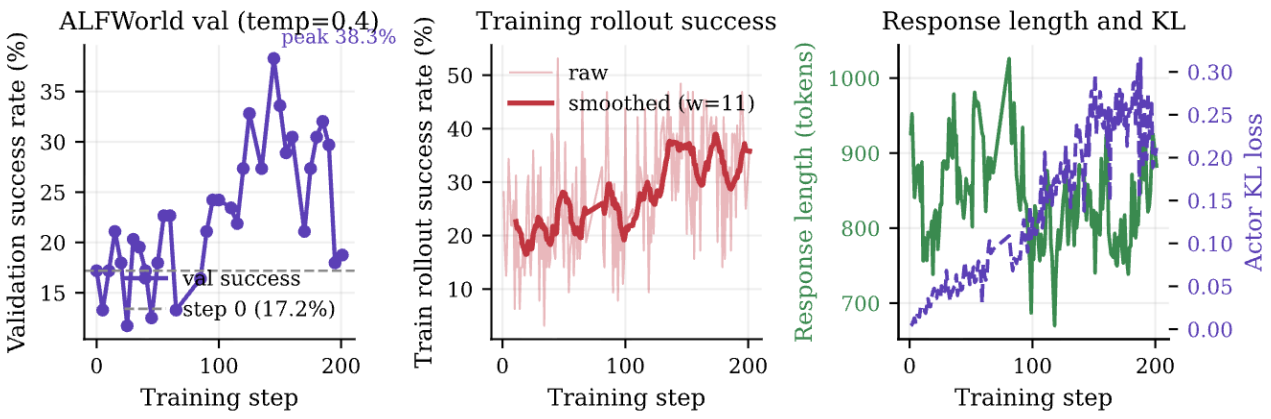

Figure 3. **GRPO training of Qwen3-1.7B on ALFWorld**

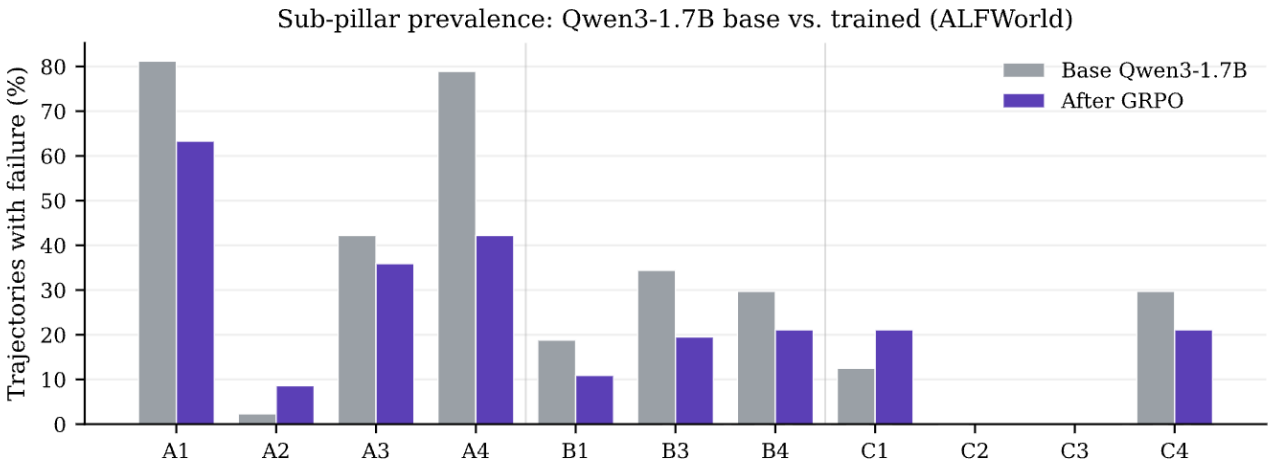

Figure 4. **Sub-pillar prevalence (% of trajectories) for Qwen3-1.7B on ALFWorld, before and after GRPO training**

within each family is preserved across closed-source frontier models, open-weight frontier models, and small open-source models. A1 is the modal sub-pillar in every model. B1 is the dominant monitoring failure in every model. C1 is the dominant outcome-evaluation failure in every model. The shape of the failure distribution is largely invariant to the model scale and family.

The absolute rates do shift with scale. The smaller Qwen3 models exhibit higher rates of A4 and B1, behaviors that reflect basic capability gaps rather than structural ones. The frontier models reduce these substantially, but A1 remains within a narrow band across all seven models. This is the cleanest evidence that A1 is not a capability gap that scaling alone closes.

4.4. Successful trajectories often contain incorrect intermediate actions

C Outcome Evaluation failures comprise a smaller percentage of failures, but their internal composition is informative. Erroneous Success Reinforcement accounts for 63.3% of all outcome-evaluation failures. Thus, there are numerous instances in which despite accomplishing the task, the agent frequently takes at least one redundant or invalid intermediate action. Although this run is rewarded successfully, at the step level this behaviour would not be acceptable when deployed in the real world. This is the gap that any process-level training method would need to fix, which no technique currently does.

C4 Premature Task Termination is the second largest sub-pillar in C. In this case, agents terminate interactions while

a valid plan still exists for the remaining sub-tasks. This reflects poor self-evaluation, where the agent does not verify whether its work is complete before terminating.

4.5. Failure patterns persist after RL training

To test whether the identified failure patterns exist despite RL post-training, we train Qwen3-1.7B with GRPO on ALFWorld for 145 steps and re-annotate the resulting trajectories using TRACE. We then compare the 128 trained-model trajectories against 128 base-model trajectories on the same ALFWorld validation set. The training hyperparameters are detailed in Appendix A.3.

RL training reduces but does not eliminate TRACE failures: Qwen3-1.7B with GRPO in ALFWorld significantly improves the task success rate. The success rate increases to 37.5% (trained) from 20.3% (base) and the average trajectory length drops to 22.1 steps from 26.1. Failures such as B1 (Identical-Action Loop) and C4 (Premature Task Termination) drop from 18.8% to 10.9% and 29.7% to 21.1% respectively. These are the failures most directly related to the basic skills that RL can refine. Despite the improved performance, A1 (Skipped Prerequisite) remains the dominant failure mode after training (63.3%), followed by A4 (Nonexistent Tool Invocation) that affects 42.2%, and then B3 (Feedback Neglect) that is observed in 19.5% of the trajectories.

Erroneous success not reduced by RL: An interesting observation from the trained model's run on the validation set was that 27 of the 48 successful trajectories (56.2%) contain at least one invalid intermediate action that received positive reward at the trajectory level.

These trajectories contribute the same positive gradient as clean successful trajectories. This is the outcome evaluation failure that TRACE identifies and discusses as Erroneous Success Reinforcement: trajectory-level rewards cannot discern a clean, successful task completion from one which has incorrect intermediate steps or behavior. As mentioned earlier, existing training techniques such as RL cannot correct this behaviour without step level, fine grained feedback.

4.6. Summary

We make three main observations from our experiments. First, the failures of LLM-based agents are heavily skewed towards A Action–Plan failures with A1 (Skipped Prerequisite) being the dominant sub-pillar for every model. The closed-source models exhibit similar failure patterns as the open-sourced ones, differing primarily in their frequency. Second, the failure patterns and distribution are independent of model size, family and benchmark. Third, RL training using GRPO shifts the absolute rates of each failure but does not change the shape of the distribution across models and benchmarks. Using trajectory-level rewards, RL is unable to rectify error C1 (Erroneous Success Reinforcement) that continues to affect more than half of all successful trajectories after training. These observations and findings signal the need for more granular, step level training signals that current training methods are unable to provide.

Discussion and Conclusion

The main goal of this paper is to improve the reliability of LLM-based agents. TRACE aims to provide a principled understanding and framework for understanding and analyzing failures in single agent, multi-step and multi-turn environments. By identifying and categorizing these failures, we aim to help the community develop more robust single agent systems in different environments. TRACE gives us a principled way to understand how agents fail, maps those failures to agentic capabilities, and highlights how and why current training methods do not alleviate these failures entirely. We connect these failures back to the training techniques and show not just what is broken, but also what to fix. Thus, our findings highlight the need for more fine-grained, step-level training signals to improve long-horizon reasoning and reliability in agents.

References

Acikgoz, E. C., Oh, J., Hao, J., Jeon, J. H., Ji, H., Hakkani-Tür, D., Tur, G., Li, X., Ma, C., and Fan, X. Speakrl: Synergizing reasoning, speaking, and acting in language models with reinforcement learning, 2025a. URL https://arxiv.org/abs/2512.13159.

Acikgoz, E. C., Qian, C., Wang, H., Dongre, V., Chen, X., Ji, H., Hakkani-Tür, D., and Tur, G. A desideratum for conversational agents: Capabilities, challenges, and future directions, 2025b. URL https://arxiv.org/abs/2504.16939.

Cemri, M. et al. Why do multi-agent llm systems fail?, 2025. URL https://arxiv.org/abs/2503.13657.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Kazemnejad, A. et al. Vineppo: Refining credit assignment in rl training of llms, 2025. URL https://arxiv.org/abs/2410.01679.

Qian, C., Acikgoz, E. C., He, Q., Wang, H., Chen, X., Hakkani-Tür, D., Tur, G., and Ji, H. Toolrl: Reward is all tool learning needs, 2025. URL https://arxiv.org/abs/2504.13958.

Rawat, M., Chakraborty, A., et al. When actions teach you to think: Reasoning-action synergy via reinforcement learning in conversational agents, 2025. URL https://arxiv.org/abs/2512.11277.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Song, P., Han, P., and Goodman, N. Large language model reasoning failures, 2026. URL https://arxiv.org/abs/2602.06176.

Various. Understanding failure modes in large language model agents, 2025a. URL https://arxiv.org/abs/2510.10150.

Various. Online process reward learning for agentic reinforcement learning, 2025b. URL https://arxiv.org/abs/2509.19199.

Various. A practitioner's guide to multi-turn agentic reinforcement learning, 2025c. URL https://arxiv.org/abs/2510.01132.

Wang, Z., Wang, K., Wang, Q., Zhang, P., Li, L., Yang, Z., Yu, K., Nguyen, M. N., Liu, L., Gottlieb, E., Lam, M., Lu, Y., Cho, K., Wu, J., Fei-Fei, L., Wang, L., Choi, Y., and Li, M. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning, 2025. URL https://arxiv.org/abs/2504.20073.

Xu, Z. et al. Agentrl: Scaling agentic reinforcement learning, 2025. URL https://arxiv.org/abs/2510.04206.

Zeng, S., Wei, Q., Brown, W., Frunza, O., Nevmyvaka, Y., and Hong, M. Reinforcing multi-turn reasoning in llm agents via turn-level credit assignment, 2025. URL https://arxiv.org/abs/2505.11821.

A. Appendix

A.1. Judge Prompt

We label every trajectory with the eleven sub-pillars using a single LLM judge (GPT-5.4 Mini, temperature 0.0). The full prompt sent to the judge is given below. The judge is given the full trajectory, which includes the benchmark configuration, the available tools, and it applies the taxonomy uniformly across benchmarks.

```
# TASK
You are conducting a quantitative error analysis of LLM agent failure
trajectories across multiple benchmarks. You will classify failures using
the MART taxonomy (defined below), then produce per-benchmark and
per-model percentage breakdowns suitable for a stacked bar chart
visualization.

You will be given trajectory data spanning thousands of trajectories
across multiple benchmarks and models. Process them programmatically
(Python). Do NOT attempt to read every trajectory by hand.

# FILE SCHEMAS

## tau-bench
Files named like <model>_<domain>_all.json where domain in {retail,
airline}. Each file is a SINGLE JSON object with:
- info: run config
- tasks: list of task definitions, each with id, description,
  user_scenario, evaluation_criteria.actions (the gold action sequence)
- simulations: list of trajectories, one per task, each with:
  - id, task_id, termination_reason, reward_info (contains reward,
    db_check, action_checks)
  - messages: list of turns. Each message has role in {assistant, user,
    tool}, content, tool_calls (list of {id, name, arguments}), turn_idx
  - Success signal: reward_info.reward == 1.0 is success; 0.0 is failure.
    Also check db_check.db_match and per-action action_match.
  - Tau benchmark = retail OR airline; treat them as separate
    sub-benchmarks in your aggregation.

## BFCL v4
Files in <model>/multi_turn/BFCL_v4_multi_turn_<category>_<result|score>.json
where category in {base, miss_func, miss_param, long_context}. Use the
*_score.json files; they contain both the trajectory and the validation
outcome.
- Format: JSON Lines. First line is a header ({"accuracy": ...,
  "correct_count": ..., "total_count": ...}); skip it. Each subsequent
  line is one task.
- Per-task fields: id, model_name, test_category, valid (bool; false =
  failed), error (with error_type and error_message), inference_log
  (the trajectory).
- inference_log[0] is initial state info; subsequent entries are dicts
  with begin_of_turn_query and step_0, step_1, ... each step containing
  assistant messages with content (often containing <tool_call> blocks)
  and reasoning_content, plus tool responses.
- Known error_type values include: multi_turn:instance_state_mismatch,
  multi_turn:execution_response_mismatch, multi_turn:force_terminated,
  multi_turn:empty_turn_model_response. Use these as STRONG PRIORS but
  do not stop there; read the trajectory to identify the actual MART
  label.

## ACEBench
Each model directory contains results/ and scores/ subdirs. Categories:
data_agent_multi_step, data_agent_multi_turn,
data_normal_multi_turn_user_adjust, data_normal_multi_turn_user_switch.
- *_result.json: JSONL. Each line: {"id": "...",
  "result": [end-state APIs], "process": [model action sequence]}
- *_score.json: JSONL. First line is summary {"end_to_end_accuracy",
  "process_accuracy", "correct_count", "total_count"}. Skip it.
- *_process.json: JSONL. Each line is {"<task_id>":
  {"process_accuracy": ..., "model_output": [...], "call_process":
  [gold sequence]}}. This is the most useful file; it gives you both
  the model's actions and the gold sequence side by side.
- Compare model_output vs call_process to detect skipped prerequisites
  (A1), wrong order (A2), wrong tool / nonexistent calls (A4), bad
  params (A3).

# THE MART TAXONOMY (USE EXACTLY THESE 11 LABELS)

## Family A -- Action-Plan Failures
- A1 Skipped Prerequisite: Agent omits a required precondition (e.g.,
  skips turn_on_wifi, skips find_user_id_by_name_zip authentication).
```

```
  Plan is incomplete.
- A2 Dependency-Order Violation: All required actions present, but
  executed in wrong order (e.g., modify_pending_order_items on
  already-delivered order).
- A3 Parameter Hallucination: Correct tool, but parameter is fabricated,
  ungrounded, or unsupported (e.g., passing user_id to a tool that does
  not accept it).
- A4 Nonexistent Tool Invocation: Agent invokes a tool that does not
  exist (e.g., inventing ask_for_reservation_id).

## Family B -- Monitoring Failures
- B1 Identical-Action Loop: Same action with identical (or
  near-identical) parameters repeated despite no progress.
- B3 Feedback Neglect: Explicit error message or contradicting
  observation given, agent ignores it and persists with broken strategy.
- B4 Entity Confusion: Agent loses track of which entity (user, order,
  product, file) is referenced.

## Family C -- Outcome Evaluation Failures
- C1 Erroneous Success Reinforcement: Trajectory succeeds despite
  incorrect/redundant intermediate steps.
- C2 Policy Violation: Agent breaks an explicit rule but reaches correct
  outcome (e.g., bundles confirmations when policy requires separate
  ones).
- C3 Observation-Action Inconsistency: Agent's next action contradicts
  evidence from its own tool observations (e.g., user claims Gold, tool
  returns regular, agent acts on the claim).
- C4 Premature Task Closure: Agent terminates before all subtasks done
  despite available tools.

# CLASSIFICATION RULES

1. Prefer one most-specific label per failure. Use multiple only when
   mechanisms are genuinely distinct.
2. Disambiguation:
   - A1 vs A2: A1 = action *missing*. A2 = action *present but
     mis-ordered*.
   - B1 vs B3: B1 = repetition without progress. B3 requires an
     explicit error/feedback signal that was ignored. Both can co-occur.
   - B1 vs B4: B1 = same action repeated. B4 = wrong entity targeted.
   - C1 vs C2: C1 = mistake that is not a rule violation. C2 = explicit
     policy violation.
   - C3 vs B4: B4 = entity-identity drift across many turns. C3 =
     single-step contradiction with a fresh observation.
3. Successful trajectories CAN have C-family failures. Do not skip them.
4. BFCL error_type is a hint, not a label. Map it:
   instance_state_mismatch often -> A1/A2/B3 depending on cause;
   force_terminated often -> B1/B3; empty_turn_model_response often ->
   C4; investigate the trajectory to confirm.
5. Flag separately (do not force into taxonomy):
   - Environment/benchmark bugs
   - Ambiguous task specs
   - Loops that look semantic-equivalent rather than identical (B2 is
     out of scope this round)

# WORKFLOW

## Phase 1 -- Setup
1. Build a master inventory: (benchmark, model, sub_benchmark,
   file_path, num_tasks). Print this inventory before doing anything
   else.
2. Strip __MACOSX/ paths and .DS_Store files.
3. For each (benchmark, model) cell, count: total trajectories, total
   successful, total failed.

## Phase 2 -- Detection (write Python heuristics, do not classify by hand)
Build per-trajectory feature extractors:
- Tool-call sequence: extract ordered list of (tool_name, arguments) for
  the model's actions.
- Gold sequence: from evaluation_criteria.actions (tau), call_process
  (ACEBench), or known error category (BFCL).
- Error/feedback messages: scan tool responses for "error", "invalid",
  "does not", "not found", etc.
- Heuristic flags (each maps to one or more MART labels):
  - missing_required_action -> candidate A1
  - out_of_order_action -> candidate A2
  - unsupported_param (param name not in declared schema) -> candidate A3
  - unknown_tool_name (tool not in available_tools list) -> candidate A4
  - repeated_identical_call (>=2 calls with identical name+args) ->
    candidate B1
  - error_then_same_strategy (error returned, next call uses same broken
```

```
    approach) -> candidate B3
  - entity_id_switch (user_id/order_id/etc. changes mid-trajectory
    inconsistently) -> candidate B4
  - succeeded_with_extra_failed_calls -> candidate C1
  - policy_keyword_violated (e.g., bundled confirm in tau retail) ->
    candidate C2
  - assistant_summary_contradicts_tool_obs -> candidate C3
  - terminated_with_open_subtasks (assistant says "cannot" or stops, but
    task list incomplete) -> candidate C4
For each trajectory, run all detectors; record every flag with:
turn_idx, evidence snippet, raw match.

## Phase 3 -- Classification
For each flagged trajectory, emit one record per failure:
{
  "benchmark": "BFCL",
  "sub_benchmark": "multi_turn_base",
  "model": "Qwen3-4B",
  "task_id": "multi_turn_base_0",
  "success": false,
  "failures": [
    {"label": "A2", "turn_index": 3,
     "evidence": "called mv before mkdir for temp/",
     "rationale": "..."}
  ]
}
Write all records to annotations.jsonl.

## Phase 4 -- Aggregation
Produce:
- failure_counts.csv with columns: benchmark, sub_benchmark, model,
  A1, A2, A3, A4, B1, B3, B4, C1, C2, C3, C4, total_failures,
  total_trajectories, total_failed_trajectories.
- failure_percentages.csv: same columns, but A1..C4 are % of
  total_failures within that row.
- family_rollup.csv: benchmark, model, family_A_count, family_B_count,
  family_C_count, family_A_pct, family_B_pct, family_C_pct.

## Phase 5 -- Plot-Ready JSON
{
  "by_benchmark": {
    "BFCL": {
      "models": ["GPT-5.4-Mini", "DeepSeek-R1", "Kimi-K2.5",
               "Llama-3.3-70B", "Qwen3-1.7B", "Qwen3-4B", "Qwen3-8B"],
      "family_A": {"A1": [per-model counts], "A2": [...],
                 "A3": [...], "A4": [...]},
      "family_B": {"B1": [...], "B3": [...], "B4": [...]},
      "family_C": {"C1": [...], "C2": [...], "C3": [...], "C4": [...]},
      "family_totals_pct": {"A": <%>, "B": <%>, "C": <%>}
    },
    "tau2_retail": { ... },
    "tau2_airline": { ... },
    "ACEBench": { ... }
  }
}
Family-level percentages should match the layout of the reference figure
(one panel per family, x-axis = model, stacked bars = sub-modes).

## Phase 6 -- Validation
1. Sample 20 random failure classifications across benchmarks. For each,
   paste the trajectory excerpt and re-justify the label. Report any
   reclassifications.
2. Sanity-check totals: sum(A1..C4) == total_failures per row of
   failure_counts.csv.
3. Report any (benchmark, model) cell where >30% of failed trajectories
   produced no MART label; this means heuristics are missing something.

# DELIVERABLES (in this order)

1. Inventory printout: what you found in each input file.
2. Methodology summary (<=300 words): how heuristics map to MART labels
   and your confidence per family.
3. annotations.jsonl: per-trajectory failure records.
4. failure_counts.csv, failure_percentages.csv, family_rollup.csv.
5. plot_data.json: the structured payload above.
6. A rendered stacked bar chart (matplotlib, faceted by family) using
   the percentages, matching the reference figure's layout.
7. Analysis (<=600 words):
   - Dominant failure modes per benchmark
   - Cross-model patterns (does smaller Qwen3 show more A1, as the
     paper suggests?)
```

```
  - Anomalies and limitations of the heuristic classifier
  - Where manual review would most improve precision

# IMPORTANT
- Process programmatically. Do not try to read every trajectory by eye.
- Cite evidence (turn_idx + short snippet) for every label.
- Keep heuristic code transparent and inspectable; emit a
  heuristics_log.txt summarizing what each detector matched on.
- Do not invent labels outside the 11 listed. Flag uncertain cases
  separately.

# BEGIN
First, restate the disambiguation rules in your own words and outline
your detection plan per benchmark (tau, BFCL, ACEBench). Then process
the input files and print the inventory.
```

A.2. Benchmarks and Models

Benchmarks: We evaluate on four benchmarks, covering multi-step and multi-turn environments:

- τ^2-bench; (retail and airline domains): multi-turn tool-using customer-service agent operating under a written policy, with a separate user-simulator LLM. We use the public test split of 114 retail and 50 airline tasks.
- BFCL v4: function-calling benchmark covering four categories used in our analysis — `multi_turn_base`, `multi_turn_long_context`, `multi_turn_miss_func`, and `multi_turn_miss_param`.
- ALFWorld: long-horizon embodied benchmark over 6 task families (`pick_and_place`, `look_at_obj_in_light`, `pick_clean_then_place_in_recep`, `pick_cool_then_place_in_recep`, `pick_heat_then_place_in_recep`, `pick_two_obj_and_place`). We use the standard 134-task unseen validation split.
- ACEBench: multi-turn tool-using benchmark focused on conversational interaction; we use the `agent_multi_turn` split.

Models We evaluate seven models including closed-source and small source models: GPT-5.4 Mini, DeepSeek-R1, Kimi-K2.5, Llama-3.3-70B-Instruct-Turbo, Qwen3-1.7B, Qwen3-4B, and Qwen3-8B. The closed-source models are queried using their APIs and the open-source models are served locally with vLLM 0.10.2 (`tool-call-parser=hermes`, `enable-auto-tool-choice`). We evaluate all models with a temperature 0.0 for deterministic comparison.

A.3. RL Training Setup

Goal: We conduct the RL experiments 4.5 to check whether the failure patterns identified by TRACE persist after post-training and to confirm our hypothesis that the identified failures highlight the lack of broader agentic capabilities that current training techniques are not able to fix.

Training framework: We train Qwen3-1.7B with GRPO on ALFWorld using `verl-agent` (built on verl 0.5.0), which provides a multi-turn rollout loop and a GRPO trainer compatible with agentic environments. We use the trainer's existing ALFWorld environment wrapper.

Hyperparameters: Hyperparameters are listed in Table 2. The choices follow the `run_alfworld.sh` configuration shipped with verl-agent, adapted to Qwen3-1.7B and our compute budget (4×A100-40GB on NERSC Perlmutter, single node, interactive QoS).

Reward: The reward is the binary ALFWorld task-success signal at the end of the trajectory. We get reward $r = 1$ if the environment concludes that the task was completed successfully and $r = 0$ otherwise. No intermediate rewards are used and the advantage for each rollout is computed as the group-relative reward following (Shao et al., 2024).

Evaluation: We evaluate on the same 128 ALFWorld validation tasks before training (base model) and after 145 GRPO steps (trained model), and then evaluate the base and trained model on the same validation set.

Table 2. GRPO hyperparameters for ALFWorld training.

Hyperparameter	**Value**
Base model	Qwen3-1.7B
Algorithm	GRPO
Group size G	4
Train batch size	16 prompts
Mini-batch size	32
Max prompt length	2,048 tokens
Max response length	2,048 tokens
Max steps per episode	30
Rollout temperature	1.0
Validation temperature	0.4
Validation tasks	128 ALFWorld unseen-val tasks
Validation frequency	every 5 steps
Optimizer	AdamW
Learning rate	1×10^{-6}
KL loss	enabled
KL coefficient	0.01
Tensor parallel size	2
GPU memory utilisation	0.6
Hardware	4×A100-40GB (1 node)
Total training steps	145

A.4. Extended Qualitative Examples

1 C1: Erroneous Success Reinforcement:

User Goal: Order total is $1,166.98 but card only has $1,150. Try to split payment. If not possible, ask for most expensive item + price and whether it can be cancelled individually. If not possible, ask if all items can be switched to cheapest options to bring total to $1,150. If so, confirm and do it. Must communicate: camera · $481.50.

Turn 1

User: Can you split the payment with another card? Card only has $1,150 and total is over $1,160.

Assistant: Most expensive item: Action Camera at $481.50. Individual items cannot be cancelled — only full orders or full modification

Turn 16

User: Switch all items to cheapest options, bring total to $1,150.

Tool_call(): **get_product_details() x 5 for all 5 products**

Turn 28

Error: Tool: modify_pending_order_items called with 5 pairs - but **Patio Umbrella and Backpack have identical old/new item IDs** (no change). API rejects: "new item id should be different from old item id".

TURN 28 – TURN 33 LOOP WITH 3 IDENTICAL FAILING CALLS

Tool: modify_pending_order_items with only 3 item pairs (camera, lamp, t-shirt) — drops the duplicate pairs and succeeds.

Assistant: "Total adjusted to $1,130.86 by replacing Camera, Lamp, T-Shirt with cheaper options. $36.13 refund processed."

Trajectory Level Evaluation (Ours)

P2 Spurious Success

Trajectory succeeds despite containing: 3 tool errors, 3-turn identical retry loop, and a semantically wrong mutation (camera replaced with T-Shirt *item_id*).

Example (τ^2-bench Retail Task 37, Qwen3-8B). The user's goal is to bring a pending order under the card limit by switching items to cheaper variants if needed. Qwen3-8B eventually reaches the correct final action, but the trajectory contains multiple intermediate errors. It first calls `find_user_id_by_email(...)` and receives "User not found," then recovers using `find_user_id_by_name_zip(...)`. Later, when modifying order #W9348897, the model issues the same invalid `modify_pending_order_items(...)` call three times, each failing with "The new item id should be different from the old item id," before finally repairing the arguments and submitting a correct reduced modification. This trajectory demonstrates C1 because the final database state is correct and the run therefore receives reward $= 1$, even though the path to success includes failed identification, repeated invalid modification attempts, and trial-and-error recovery. At the trajectory level the run is labeled successful, but the positive signal would also reinforce intermediate behaviors that should not be imitated.

❶ **B1: Identical-Action Loop:**

User Goal: Two pending orders #ORDER-A and #ORDER-B. Cancel #ORDER-A if not processed within 5 days, confirm refund [$XXX]. Change shipping address on #ORDER-B to [ADDR] and confirm order total [$YYY].

Turn 1

User: I have placed two orders #ORDER-A and #ORDER-B. Cancel #ORDER-A if not processed within 5 days, confirm refund [$XXX]. Change shipping address on #ORDER-B to [ADDR] and confirm order total [$YYY].

Turn 2

Assistant: To assist you, I need to authenticate your identity. Could you please provide your email address or your full name and zip code?

User: I am [USER-NAME] and my zip code is [ZIP]

Tool_call(): **modify_pending_order_address**({"order_id": "ORDER-A ","address": "[ADDR]"})

Tool_call(): **get_order_details**({"order_id "ORDER-B")

⋮

Turn 27

Tool_call(): **modify_pending_order_address**({"order_id": "ORDER-A ","address": "[ADDR]"})

Tool_call(): **get_order_details**({"order_id "ORDER-B")

Trajectory Level Evaluation (Ours)

P3 Circular Tool Invocation

Once loop breaks, agent cancels ORDER-A (the address-modified order) instead of ORDER-B (the older one the user explicitly wanted cancelled). Agent has lost track of which order was which because of the 27-turn loop confusion. DB now permanently wrong.

Example (τ^2-bench Retail Task 59, Qwen3-4B). After authentication, the agent correctly retrieves the two pending orders, but then fails to maintain the mapping between the user's two requests and the two order IDs. Instead of executing the cancellation and address-update branches cleanly, it falls into a repair loop that alternates between `modify_pending_order_address(...)` on ORDER-A and `get_order_details(...)` on ORDER-B. The problem is not repetition in the abstract; the same address-modification call is re-issued on the same order multiple times, while the second order is repeatedly reopened without progress. Over the course of the loop, the agent loses track of which order should be canceled and which one should be updated, and the final state reflects that confusion. This example demonstrates B1 because the repeated tool pattern is persistent, visible at the action level, and directly tied to the downstream order mix-up rather than being a harmless redundancy.