BEYOND FUNCTIONAL CORRECTNESS: OPTIMIZING MULTI-TURN TOOL-AUGMENTED AGENT TRAJECTORIES TO REDUCE TOOL USAGE

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents are increasingly evaluated on multi-turn tool use, where success depends not only on calling the correct APIs but also on keeping interaction trajectories compact. Existing alignment methods largely optimize for correctness, while efficiency-oriented approaches mainly target retrieval or single-query settings, leaving open how to reduce excess tool calls in sustained, stateful dialogues without sacrificing task success. We propose an iterative Direct Preference Optimization (DPO) framework that improves both accuracy and efficiency for multi-turn tool agents. For each query, we sample multiple successful trajectories, rank them by tool-call overhead, and convert best–worst discrepancies into step-level preference pairs via two complementary rules: one detecting redundant or corrective detours, and another recovering divergences under parallel tool calling. A greedy decoding filter retains only pairs the current policy has not yet mastered, and the sample–construct–optimize loop is repeated to expose deeper inefficiencies inside turns. When this cross-trajectory loop plateaus, we further mine preferences from the greedy successful trajectory via validated pruning—ablating removable tool calls under the BFCL multi-turn checker—and apply an additional DPO stage with cleaned prefixes. On BFCL v3 multi-turn subsets, LoRA-based DPO followed by this prune stage raises an 8B BitAgent model to 79.5% average accuracy, surpassing the previous overall state of the art (xLAM-2-70b-fc-r, 77.38%), while reducing excess tool calls on the Base subset from 2082 to 25. Trained only on Base, the same policy generalizes to Miss-Param, Missing-Func, and Long-Context subsets with fewer excess tool calls (and often higher accuracy), and also lowers total tokens and inference steps, indicating that compact trajectories often coincide with correct multi-turn behavior.

1 INTRODUCTION

Recent years have witnessed a significant shift in the deployment of Large Language Model (LLM)-based agents, moving from static API calling (Yang et al., 2023)(Vemprala et al., 2024) towards complex, real-world multi-turn interactions. This evolution is underpinned by the development of comprehensive evaluation benchmarks, such as the **Berkeley Function Calling Leaderboard (BFCL) v3** (Patil et al., 2024) and **τ-bench** (Yao et al., 2025). To address the scarcity of high-fidelity multi-turn data, frameworks like **APIGen-MT** (Prabhakar et al., 2025) utilize agentic interplay and iterative feedback to generate diverse, verifiable training trajectories. These resources have substantially advanced *task success*—helping agents select correct tools with well-formed arguments across sustained dialogues—yet they leave open a distinct challenge in *multi-turn* tool agents: how to reduce excess API calls and inflated dialogue trajectories while preserving task success under stateful highly interactive settings such as booking tickets or sending messages.

To narrow the gap between human-level reasoning and model performance, prior work has explored complementary inference-time strategies beyond single-path greedy decoding. **Self-Consistency** (Wang et al., 2023) improves robustness by sampling multiple reasoning traces and aggregating their answers, while **ReAct** (Yao et al., 2023b) interleaves verbal reasoning with environment-grounded actions to enable step-wise decision making. Building on closed-loop control, **AdaPlanner** (Sun et al., 2023) adaptively revises code-formulated plans using environmental

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.

feedback. To more systematically explore the solution space, **Tree of Thoughts (ToT)** (Yao et al., 2023a) treats intermediate thoughts as searchable states, **Reasoning via Planning (RAP)** (Hao et al., 2023) employs **Monte Carlo Tree Search (MCTS)** to balance exploration and exploitation, and **ToolChain*** (Zhuang et al., 2024) applies **A*** **search** to prune high-cost branches in large tool-use action spaces. Complementarily, **Deductive Verification** (Ling et al., 2023) enforces logical coherence by checking intermediate reasoning steps. Moving beyond single-agent inference, multi-agent frameworks such as **DEI** (Zhang et al., 2025b) integrate specialized agent expertise through committee-style selection, surpassing the performance ceiling of any individual agent.

Building upon these decoding-time paradigms, recent research in agent training has shifted toward integrating such iterative rollouts with alignment objectives to select correct tools with well-formed arguments under diverse conditions. To refine model reasoning and improve decision-making precision, alignment algorithms—including PPO (Ouyang et al., 2022), DPO (Rafailov et al., 2023), and GRPO (Shao et al., 2024)—are increasingly employed. Specifically, the **Exploration-Based Trajectory Optimization (ETO)** framework (Song et al., 2024) employs DPO to refine policies by contrasting successful and erroneous trajectories derived from vanilla rollouts during iterative exploration. The FunRL (Hao et al., 2025) framework improves Large Language Model function calling performance by integrating Chain-of-Thought entropy into an enhanced Group Relative Policy Optimization (GRPO) algorithm to optimize multi-sample trajectory exploration and alignment. The **ATA**(Chen et al., 2024a) framework improves tool-augmented model performance and generalization by applying DPO to a preference dataset derived from tree-structured expert trajectories and previously neglected failed explorations. Furthermore, the **CIR** framework (Bai et al., 2026) and **ReTool** (Feng et al., 2026) enhance reasoning accuracy by by utilizing code-integrated paradigms that leverage executable code and iterative error feedback, tackling complex multi-step tasks through PPO based strategic tool optimization.

However, achieving mere correctness is insufficient for high-quality tool-augmented behavior. Even successful trajectories often suffer from efficiency problems that artificially inflate reasoning steps.In the field of retrieval-augmented agents, recent research has focused on improving tool efficiency by optimizing the decision-making boundary between leveraging internal parametric knowledge and external retrieval; for example, **SMART** (Qian et al., 2025) and **Search Wisely** (Wu et al., 2025b) enhance agentic self-awareness and uncertainty quantification to strategically mitigate tool overuse and redundant retrieval behaviors. In math and retrieval-augmented scenarios, **ARPO** (Dong et al., 2026) optimizes tool efficiency by leveraging an entropy-based adaptive rollout mechanism and GRPO-based step-level advantage estimation to prune redundant interactions while preserving critical reasoning paths(Dong et al., 2026). Similarly, **Tool-Light** (Chen et al., 2026) addresses sub-optimal reasoning behaviors—such as tool overuse and overthinking—by employing information entropy to guide sampling and incorporating a self-evolved preference learning paradigm to stabilize the tool-integrated reasoning process.

These efforts, however, primarily target single-query invocation or retrieval/code-integrated reasoning, rather than sustained multi-turn tool dialogues. To close this gap, we propose an iterative preference-optimization framework that improves both *correctness* and *efficiency* of multi-turn tool agents. Our pipeline consists of four steps:

- **Trajectory sampling.** We perform vanilla sampling to collect multiple candidate trajectories for each query, retain only the successful ones, and rank them by the number of tool calls.
- **Step-level preference construction.** By contrasting the most efficient successful trajectories against the least efficient ones and applying rule-based filtering, we construct step-level preference pairs that capture local decision differences along the dialogue.
- **Iterative DPO.** We fine-tune the policy with Direct Preference Optimization (DPO) on the constructed pairs, then re-run trajectory sampling and step-level preference construction with the updated model to obtain a new preference set, and repeat this sample–construct–optimize loop for multiple rounds.
- **Intra-trajectory pruning.** When cross-trajectory contrasts become sparse, we mine additional preferences from the greedy successful rollout by validated tool-call ablation under the BFCL multi-turn checker, and apply a final DPO stage with cleaned prefixes.

Empirically, our method achieves state-of-the-art performance on the BFCL v3 multi-turn subsets, attaining an average accuracy of 79.5% and surpassing the previous best open model, xLAM-2-70b-fc-r (FC). Beyond accuracy gains, optimization on the same model substantially reduces tool-call overhead: on the multi-turn base subset, the number of *excess* tool calls—defined as the total tool calls over all completed samples minus those required by the ground-truth trajectories—drops from 2082 to 25.

2 PRELIMINARIES

2.1 AGENT-USER INTERACTION FORMULATION

We consider agent–user interaction as a process in which, under a shared conversational context, a user sequentially provides multiple intents, and the agent responds to each intent by invoking one or more available tools, acquiring necessary information, and completing the specified task, before finally producing a textual summary as its response to each intent. This process naturally exhibits partial observability from the agent's perspective. We formalize this interaction as a Partially Observable Markov Process (POMP) and characterize it from the agent's viewpoint by the tuple

$$(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}).$$

State Space The state space is defined as $\mathcal{S} = \{\mathcal{S}_{env}, \mathcal{S}_{conv}\}$, where $\mathcal{S}_{env}$ denotes environment states that are not directly observable by the agent (e.g., external world states or tool backends), and $\mathcal{S}_{conv}$ represents the observable conversational history, including all previous user messages, agent responses and tool-returned observations.

Observation Space The observation space is given by $\mathcal{O} = \{\mathcal{O}_{tool}, \mathcal{O}_{user}\}$, where $\mathcal{O}_{user}$ corresponds to messages provided by the user, and $\mathcal{O}_{tool}$ consists of outputs returned by external tools following tool invocations made by the agent.

Action Space The agent's action space is defined as $\mathcal{A} = \{\mathcal{A}_{toolcall}, \mathcal{A}_{resp}\}$, where $\mathcal{A}_{toolcall}$ represents structured tool function calls that conform to predefined schemas, and $\mathcal{A}_{\text{resp}}$ denotes natural-language responses generated by the agent. For each user intent, the agent may invoke one or more tools to obtain relevant information, and a single round of interaction terminates when the agent produces an action from $\mathcal{A}_{\text{resp}}$.

Transition Dynamics The transition function $\mathcal{T}$ consists of two complementary components:

$$\mathcal{T} : \begin{cases} \mathcal{S} \times \mathcal{A} \to \mathcal{S} \times \mathcal{O}, \\ \mathcal{S} \times \mathcal{O} \to \mathcal{S} \times \mathcal{A}. \end{cases}$$

The first corresponds to the agent executing an action and receiving external feedback, while the second models the agent processing an observation and selecting the next action.

Interaction Process At the beginning of a conversation, in the first step, the agent receives an initial user request $o^1 \in \mathcal{O}_{\text{user}}$, and selects an action $a^1 \in \mathcal{A}$, leading to the transition $(s^0, o^1) \to (s^1, a^1)$. At an intermediate step t, the interaction may proceed via one of the following transitions: $(s^t, a^t) \to (s^{t+1}, o^{t+1})$, or $(s^t, o^t) \to (s^{t+1}, a^{t+1})$, depending on whether the agent is executing an action or receiving an external observation at step t. If $a^t \in \mathcal{A}_{\text{toolcall}}$, the agent receives a tool output $o^{t+1} \in \mathcal{O}_{\text{tool}}$. If $a^t \in \mathcal{A}_{\text{resp}}$ and the interaction has not yet terminated, the agent receives the next user intent $o^{t+1} \in \mathcal{O}_{\text{user}}$. For transitions of the form $\mathcal{S} \times \mathcal{O} \to \mathcal{S} \times \mathcal{A}$, there are no explicit restrictions on which subsets o^t or a^{t+1} belong to, as long as they are valid elements of $\mathcal{O}$ and $\mathcal{A}$, respectively. The final transition terminates the interaction when the agent outputs $a^{\text{final}} \in \mathcal{A}_{\text{resp}}$.

2.2 STEP-LEVEL PREFERENCES AND THE DPO OBJECTIVE

To unlock the model's inherent potential for self-optimization, we focus on mining latent response sequences from the model's own distribution. Specifically, for a given user requirement, we employ high-temperature decoding to sample multiple successful action sequences. While these trajectories

all converge to the correct final state, they exhibit varying degrees of efficiency. By comparing these samples, we identify pairs consisting of a concise trajectory (preferred) and a redundant trajectory (rejected). This allows the model to learn from its own successful but unequally efficient rollouts, refining the policy toward compact tool use without relying on external expert demonstrations.

We define a trajectory as an interleaved sequence of external observations $o \in \mathcal{O}$ and agent actions $a \in \mathcal{A}$, denoted as $\tau = (o^1, a^1, o^2, a^2, \ldots, o^{final}, a^{final})$. A key observation is that, once two trajectories share the same prefix, their divergence is often determined by the next local action: the preferred action a_p preserves a compact tool-use trajectory, while the rejected action a_r frequently leads to downstream redundant tool calls or unnecessary error-recovery loops. Specifically, given a shared prefix context $c = (o^1, a^1, \ldots, a^k, o^k)$, where k denotes the index of the final observation before the trajectories diverge, the preferred trajectory directly prescribes the efficient action a_p, whereas the rejected trajectory selects a suboptimal next action a_r, which propagates into subsequent redundant or corrective transitions. Two representative cases are illustrated in Appendix A: an extra retrieval call despite already-available context, and an avoidable error that triggers a later recovery step. Motivated by this observation, we construct a step-level preference dataset $\mathcal{D} = \{(c, a_p, a_r)\}$ and use it to refine the policy.

Direct Preference Optimization(DPO) is particularly efficacious in this context due to its ability to perform precise policy alignment while maintaining the model's foundational capabilities. The optimization objective is formulated as:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(c,a_p,a_r)\sim\mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(a_p|c)}{\pi_{\text{ref}}(a_p|c)} - \beta \log \frac{\pi_\theta(a_r|c)}{\pi_{\text{ref}}(a_r|c)} \right) \right],$$

where $c = (o^1, a^1, \ldots, a^k, o^k)$ is the prefix context, π_{ref} denotes the reference model prior to fine-tuning, θ represents the trainable parameters, and β is a critical hyperparameter that governs the KL-divergence penalty between the policy π_θ and the reference model π_{ref}. The primary advantage of employing DPO here is twofold. First, the β parameter provides a controllable constraint that prevents the model from collapsing or drifting too far from its original distribution, thereby preserving overall task accuracy. Second, instead of a broad update, DPO enables the targeted recalibration of action probabilities—specifically penalizing the tokens that trigger redundant loops or unnecessary error-recovery steps, while reinforcing the direct, efficient paths.

3 METHODS

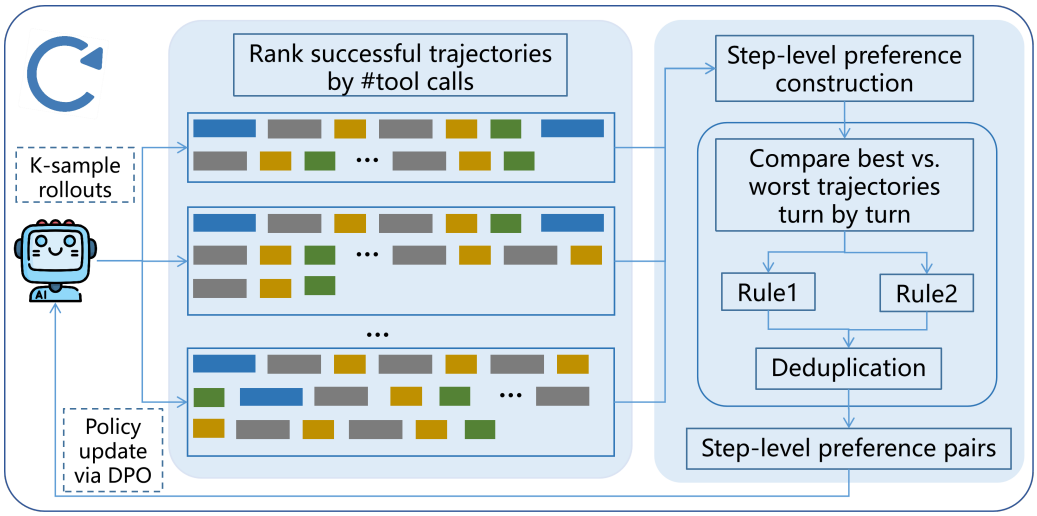
3.1 ITERATIVE DIRECT PREFERENCE OPTIMIZATION

We adopt an iterative DPO procedure that repeatedly re-samples trajectories from the updated policy, rebuilds step-level preference pairs, and re-optimizes the model (Fig. 1a). Each iteration consists of three stages: sampling and ranking, step-level preference construction, and a DPO update. We detail these stages below.

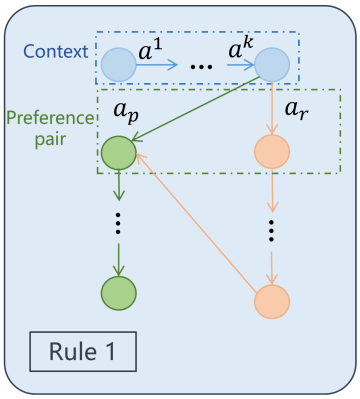
Sampling and selection. For each query, we collect K trajectories via high-temperature vanilla sampling together with one greedy-decoded trajectory. We retain only successful trajectories and measure each by tool-call overhead (equivalently, the number of tool calls relative to a compact reference). Among these successful rollouts we select the most efficient as *best* and the least efficient as *worst*.

Turn-wise comparison. We then contrast the best and worst trajectories turn by turn under the shared conversational context. At each turn, we seek local decision differences that explain why one path remains compact while the other accumulates excess tool calls, and convert these differences into step-level preference pairs (c, a_p, a_r). To make this conversion robust under realistic tool-calling patterns, we apply two complementary filtering rules (Figs. 1b and 1c).

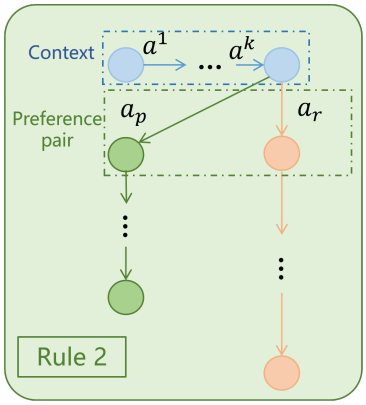
Rule 1: multi-locus detour detection. Rule 1 targets elongated subpaths caused by redundant tool invocations or avoidable error-recovery loops (Fig. 1b). Given a shared prefix, if the rejected branch takes a longer detour before rejoining an equivalent progress point already reachable by the preferred branch, we treat the first diverging actions as a preference pair. A key advantage of this rule



(a) Iterative DPO pipeline.



(b) Rule 1: multi-locus detour detection.



(c) Rule 2: first-divergence under parallel tool calls.

Figure 1: Overview of our iterative preference-optimization framework. (a) We sample K high-temperature trajectories plus one greedy trajectory, rank successful rollouts by tool-call overhead, construct step-level preference pairs, and update the policy via DPO in a closed loop. (b) Rule 1 extracts preference pairs from redundant tool calls or avoidable error-recovery detours, and can fire multiple times within the same turn. (c) Rule 2 recovers missed pairs under parallel tool calling by selecting the first divergence point on turns with unequal step counts.

is that it can fire *multiple times within the same turn*: whenever a local detour appears, an additional (c, a_p, a_r) pair can be extracted, providing denser supervision than trajectory-level preference alone.

Rule 2: first-divergence under parallel tool calls. In practice, an agent may emit multiple tool calls in a single action. Such parallel invocations break the sequential detour pattern assumed by Rule 1 and can cause otherwise informative efficiency gaps to be missed. Rule 2 addresses this failure mode (Fig. 1c): for turns whose best and worst trajectories differ in the number of steps, we identify the *first* divergence point and add the corresponding action pair to the preference set. This complements Rule 1 by recovering samples that would otherwise be filtered out under parallel tool-call schemas.

Deduplication and iterative updates. All extracted pairs are first deduplicated to avoid over-representing repeated local decisions. We then apply a greedy decoding filter to keep only informative supervision: a pair (c, a_p, a_r) is retained only if the current policy's greedy decoding under c fails to produce the preferred action a_p. The resulting set $\mathcal{D} = \{(c, a_p, a_r)\}$ is therefore both

compact and targeted—it excludes cases where the model already behaves efficiently under greedy decoding—and is used for the DPO update of the current round. We then update the policy with the DPO objective in Sec. 2.2 and close the loop by re-sampling trajectories from the refined model. This iteration is essential for exposing deeper efficiency gaps. Rule 2 corrects the first diverging action on turns with unequal tool budgets; after DPO, the updated policy tends to choose the efficient action already at the start of those turns. As a result, later sampling no longer collapses into early redundant paths, and previously hidden best–worst divergences deeper in the dialogue become available for preference construction in subsequent rounds.

3.2 Intra-Trajectory Preference Mining via Validated Pruning

When iterative DPO approaches a performance plateau, the best–worst gap in Sec. 3.1 often shrinks: successful rollouts become similarly compact, so cross-trajectory contrasts yield too few informative (c, a_p, a_r) pairs. In this regime, we switch to *intra-trajectory* preference mining on the model's *greedy* successful trajectory. Concretely, we ablate tool calls one by one (and, when needed, in small combinations) and re-validate each candidate with the BFCL multi-turn checker—requiring the pruned rollout to preserve the final environment state and the required execution-response coverage. Any call whose removal remains valid is treated as evidence of local inefficiency, from which we construct step-level preferences (c, a_p, a_r) with cleaned prefixes.

Validated pruning of a successful trajectory. We treat the official multi-turn checker as a validity oracle and search for removable actions under three complementary cut types, applied in order.

Autonomy.(Chen et al., 2024b) Within each turn, we attempt to delete entire assistant tool actions from the *end* of the turn toward the beginning. A action is marked as an Autonomy cut if the trajectory remains valid after its removal. Intuitively, these are trailing tool calls that do not affect task completion; the preferred local action is to terminate the turn with a natural-language response.

Redundancy and Error. After Autonomy cuts are fixed, we scan the remaining tool calls from back to front and try deleting one call at a time. If removal preserves validity, the call is recorded as a cut: if the corresponding tool observation is an explicit failure, we label it **Error**; otherwise we label it **Redundancy**. Thus Error captures avoidable failed attempts, while Redundancy captures successful but unnecessary invocations (e.g., re-fetching information already available).

Repair. Error cuts alone do not always expose compensating detours that are jointly deletable (e.g., entering a directory that does not contain the target file, then returning to the previous working directory). Therefore, on every turn that contains at least one Error cut, we take the residual tool calls after removing Redundancy and Error, say m calls, and enumerate deletions of size $k = 2, \ldots, m - 1$. A subset whose removal still yields a valid trajectory is recorded as a **Repair**. The corresponding pseudocode is given in Alg. 1 (Appendix B).

Step-level pairs with cleaned prefixes. Cuts identify *which* actions—or individual tool calls within an action—are removable; we next convert them into stepwise preferences whose contexts are causally consistent with the preferred policy. For each trajectory, we form a per-turn deletion set Δ from the Autonomy, Redundancy/Error, and Repair cuts obtained above, and walk the turn from front to back while maintaining a cleaned prefix c. At each assistant action whose content changes under Δ:

- if only a subset of parallel calls is deleted, we emit (c, a_p, a_r) with a_r the original call list and a_p the retained calls, then append a_p and the corresponding tool returns;
- if the entire action is deleted, we emit a pair against the next retained action (or the turn-final text under Autonomy) and *jump forward*, appending that retained action and its returns so subsequent pairs do not reuse a inefficient history;
- if the action is an Autonomy cut, we set a_p to the turn-final natural-language reply and skip remaining tool steps in the turn.

Cross-turn history is accumulated in cleaned form, so later pairs never condition on tool observations that the preferred policy would not have invoked. Alg. 2 (Appendix B) formalizes this cleaned-prefix construction. We finally apply one an additional DPO stage on the resulting pairs. Unlike

Table 1: Comparison of accuracy (%) and excess tool calls (Exc.) on BFCL v3 multi-turn subsets. Exc. denotes the total number of tool calls consumed by the model over completed samples minus those required by the ground-truth trajectories. the ratio in parentheses (e.g., $10/14$) denotes the alignment ratio on dataset: after DPO, the number of samples whose greedy decoding matches a_p, divided by $|\mathcal{D}|$. Best Acc. (higher) and best Exc. (lower) in each column are in bold.

Model	Base		Miss-Param		Missing-Func		Long-Context	
	Acc (%)	Exc.	Acc (%)	Exc.	Acc (%)	Exc.	Acc (%)	Exc.
BitAgent-Bounty-8B	78.0	2082	67.0	2495	48.5	4995	57.5	3399
+ DPO-iter1 (10/14)	84.5	682	75.0	883	63.5	3188	73.0	1594
+ DPO-iter2 (6/10)	85.0	181	77.5	579	71.5	2006	77.5	944
+ DPO-iter3 (3/6)	85.0	150	78.5	499	72.0	1895	78.0	957
+ DPO-iter4 (2/4)	**85.5**	71	78.5	362	71.5	2023	78.5	903
+ DPO-prune (4/9)	85.0	**25**	**79.0**	368	**75.0**	1714	**79.0**	778
Llama-xLAM-2-8b-fc-r	77.5	157	64.5	298	69.5	473	67.5	74
+ DPO-iter1 (11/13)	77.5	101	63.5	268	69.5	401	69.5	35
+ DPO-iter2 (7/11)	78.0	85	63.0	254	70.0	359	69.0	29
+ DPO-iter3 (3/5)	78.0	83	64.5	218	70.0	366	68.5	22
+ DPO-prune (14/71)	77.5	65	62.5	**216**	72.0	**322**	71.0	**11**

the iterative best–worst pairs, these preferred actions are counterfactual compressions of a greedy rollout rather than modes of the model's own sampling distribution; we therefore use fewer training epochs to cut redundant tool use while limiting accuracy regression.

4 Experiments

To optimize the agent's action space and minimize excessive tool invocations, we perform self-optimization using Direct Preference Optimization (DPO) on the **Base Multi-Turn** subset of BFCL v3 (Patil et al., 2024)[1]. We focus on two strong 8B open-source backbones, **BitAgent-Bounty-8B** and **Llama-xLAM-2-8b-fc-r** (Zhang et al., 2025a)—hereafter BitAgent and xLAM for brevity—which currently achieve the highest accuracies among open models at this scale. Notably, xLAM already exceeds the strongest proprietary baseline on this subset, Claude-Opus-4-5-20251101 (FC). Detailed sampling settings, LoRA hyperparameters, and the additional prune-stage schedule are provided in Appendix C. Under this training pipeline (Secs. 3.1 and 3.2), the 8B BitAgent reaches an average accuracy of 79.5% across the four multi-turn subsets, surpassing the previous state of the art, xLAM-2-70b-fc-r (FC) (77.38%), despite using a much smaller backbone.

4.1 Performance on Base Multi-Turn

As shown in Table 1, iterative DPO consistently improves tool-use efficiency on the training distribution for both backbones, while the accuracy trajectory depends on the starting policy. For xLAM, which is not good at parallel tool calling, constructing preferences with **Rule 1 alone** is more effective than combining Rule 1 and Rule 2; all xLAM results are therefore obtained under Rule 1-only iteration. The first round (`DPO-iter1`, $11/13$) preserves the baseline accuracy of 77.5% while reducing excess tool calls by 35.7% (157→101), showing that preference learning can refine trajectories without hurting core competence. A second round ($7/11$) further compresses excess calls to 85 and slightly raises accuracy to 78.0%. The third round ($3/5$) yields only marginal additional compression (85→83) with accuracy saturating at 78.0%, indicating that Rule 1-only optimization on xLAM has largely reached a plateau. We then apply an additional `DPO-prune` stage ($14/71$) with intra-trajectory pairs mined from greedy successful Base rollouts. This further cuts excess calls to 65 while restoring Base accuracy to the baseline 77.5%, suggesting that counterfactual prune preferences are effective for compression.In contrast, BitAgent improves accuracy and efficiency jointly across rounds. `DPO-iter1` ($10/14$) already lifts Base accuracy from 78.0% to 84.5% and cuts excess calls from 2082 to 682. Subsequent iterations continue this trend: `iter2–iter4` push accuracy to 85.0%/85.0%/85.5% while shrinking excess calls to 181/150/71. The final `DPO-prune`

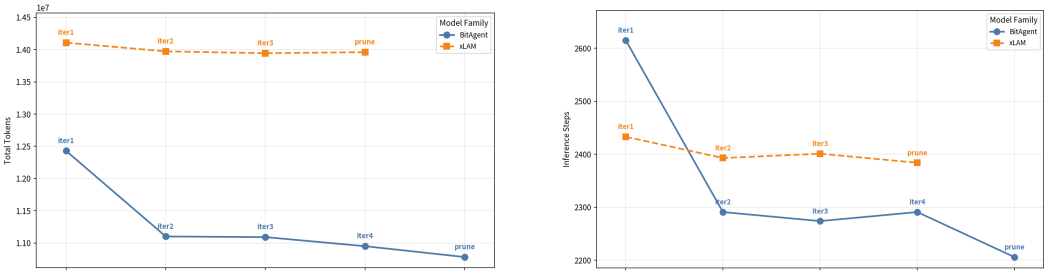
[1] `https://gorilla.cs.berkeley.edu/blogs/13_bfcl_v3_multi_turn.html`

stage (4/9) preserves near-peak accuracy (85.0%) and further reduces excess calls to 25—the lowest Base Exc. in Table 1—showing that validated pruning can still recover compression after best–worst contrasts become sparse. This supports our earlier observation that, for already capable agents, preferring compact successful trajectories often coincides with choosing the correct solution path.

4.2 Generalization to Augmented Scenarios

Although preference pairs are constructed only from the Base subset, the resulting policies transfer to the Augmented categories (Miss-Param, Missing-Func, and Long-Context). This suggests that the model internalizes a preference for minimum-viable tool use that remains useful under harder interaction conditions.

- **Miss-Param.** This subset requires recognizing incomplete arguments and seeking clarification instead of guessing. On xLAM, excess calls decrease steadily across iterative rounds and the prune stage (298→268→254→218→216), while accuracy stays near the baseline (64.5%→63.5%→63.0%→64.5%→62.5%). On BitAgent, the transfer is stronger: accuracy rises from 67.0% to 78.5% by `iter3`/`iter4` and reaches 79.0% after `DPO-prune`, while excess calls drop from 2495 to 362/368, indicating fewer speculative executions under missing parameters.
- **Missing-Func.** Here the agent should detect that no available function can fulfill the request, rather than exhaustively trying irrelevant tools. xLAM reduces excess calls from 473 to 359 by `iter2` (with a slight rebound to 366 at `iter3`) while retaining or slightly improving accuracy (69.5%→70.0%); `DPO-prune` then raises accuracy to 72.0% and lowers Exc. to 322. BitAgent shows large early gains (Acc. 48.5%→72.0% at `iter3`; Exc. 4995→1895), after which `iter4` trades a small accuracy drop (71.5%) for no further Exc. improvement (2023). The subsequent prune stage recovers both metrics (Acc. 75.0%; Exc. 1714), suggesting that intra-trajectory supervision corrects mild over-compression from the last iterative round.
- **Long-Context.** Amid distractor information, efficient policies should invoke only task-critical tools. xLAM improves both metrics in early rounds (Acc. 67.5%→69.5%; Exc. 74→35), then continues compressing excess calls to 22 by `iter3` with a mild accuracy regression to 68.5%; `DPO-prune` lifts accuracy to 71.0%—the best Long-Context Acc. for xLAM—while further reducing Exc. to 11. BitAgent again exhibits large joint gains through `iter4` (57.5%→78.5%; 3399→903), and prune pushes Acc. to 79.0% with Exc. 778, reinforcing that Base-trained compactness preferences—including the final prune stage—transfer to information-dense multi-turn settings.



(a) Total tokens on Base.

(b) Inference steps on Missing-Func.

Figure 2: Efficiency side effects of iterative DPO and the final prune stage. Reducing excess tool calls also lowers token consumption and inference depth. Solid lines denote BitAgent; dashed lines denote xLAM. xLAM has one fewer iterative round before prune.

4.3 Effect on Token Cost and Inference Steps

Reducing excess tool calls should also lower end-to-end interaction cost. Figs. 2a and 2b track *total tokens* on Base and *inference steps* on Missing-Func across DPO rounds, including the final

DPO-prune stage. Both metrics decline as Exc. shrinks, but with different slopes: BitAgent drops sharply after the first iteration and keeps compressing through prune, whereas xLAM improves more gradually from an already compact baseline. Full per-subset token and step statistics, together with a discussion of why Exc. and inference steps are complementary under parallel tool calling, are provided in Appendix D.

An apparent discrepancy, however, is that BitAgent often reports *higher* excess tool calls than xLAM, yet requires *fewer* inference steps after DPO (Fig. 2b). This is explained by parallel tool calling: BitAgent frequently emits multiple tool invocations within a single assistant step, so one inference step can amortize several tool calls. In contrast, xLAM is weak at parallel calling and tends to serialize tool use, yielding fewer tools per step but a longer step sequence.

5 Related Work

Efficiency issues in successful tool-use trajectories. A growing body of work shows that task success alone is insufficient for assessing tool-augmented agents. **TTUA** (Chen et al., 2024b) highlights an autonomy-related failure mode in which an agent has already obtained enough information to complete the task, yet continues invoking tools unnecessarily. **TRACE** (Kim et al., 2026) likewise argues that evaluation should look beyond the final answer: it prompts general-purpose LLMs, typically in a few-shot setting, to inspect reasoning trajectories and identify inefficient tool-calling steps along otherwise successful paths. These studies establish that redundant or prolonged tool use is a first-class problem even when the eventual outcome is correct.

Efficient retrieval and single-query tool use. Motivated by tool overuse, several methods optimize *when* and *how often* to call external resources, primarily in retrieval-augmented or single-query settings. **SMART** (Qian et al., 2025) and **Search Wisely** (Wu et al., 2025b) improve agentic self-awareness and uncertainty estimation to decide between parametric knowledge and external retrieval, thereby mitigating unnecessary search. In related search/math scenarios, **ARPO** (Dong et al., 2026) and **Tool-Light** (Chen et al., 2026) encourage compact tool-integrated reasoning via entropy-aware rollouts or preference learning, while **StepSearch** (Zheng et al., 2025) introduces step-wise RL rewards with redundancy penalties for multi-hop search, and **WebDancer** (Wu et al., 2025a) trains autonomous information-seeking agents with staged SFT and on-policy RL over search/click actions. Collectively, these approaches reduce retrieval or tool frequency for a given query, but they mainly operate within a single user request—or agent–environment search loops for that request—rather than sustained dialogues with evolving user intents.

6 Conclusion

We study efficiency as a first-class objective for multi-turn tool agents: even successful trajectories can accumulate excess API calls that inflate dialogue cost. To address this, we introduced an iterative preference-optimization pipeline that mines step-level contrasts between compact and elongated successful rollouts, filters them with rule-based divergence detection and greedy decoding checks, and repeatedly updates the policy with DPO. When cross-trajectory contrasts become sparse near the performance plateau, an additional DPO-prune stage further mines intra-trajectory preferences by validating and removing Autonomy, Redundancy, Error, and Repair calls, yielding a final compactness boost. Empirically, the approach improves both correctness and compactness on BFCL v3. In particular, an 8B BitAgent backbone reaches 79.5% average multi-turn accuracy after four iterations plus prune, exceeding the prior overall SOTA held by a 70B model, while sharply reducing excess tool calls, token consumption, and inference steps. Although preference pairs are constructed from Base trajectories alone, the resulting policy transfers to Miss-Param, Missing-Func, and Long-Context augmented scenarios.

Looking ahead, a natural extension is to study tool-call efficiency under multi-agent collaboration, where multiple agents jointly solve tasks through multi-turn interaction and tool use. In such settings, redundant calls may arise not only within a single agent's trajectory but also from duplicated or poorly coordinated actions across agents; adapting step-level preference optimization to this regime is a promising direction for broader agent systems.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish writing, for retrieval and discovery of related work, and to assist in implementing experimental code. Additionally, we used generative AI tools as a writing assistant for drafting and revising paper text. We have reviewed all AI-assisted work: all retrieved references were manually screened and read by the authors; AI-suggested wording was edited for accuracy and clarity; and all AI-assisted code was read, modified, and verified by the authors before use in experiments. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies preference optimization for multi-turn tool-using LLM agents, with the goal of improving both task success and interaction efficiency by reducing excess API calls on public benchmarks (BFCL v3). Our experiments use existing open-source models and publicly available evaluation protocols; we do not collect new human-subject data, private user logs, or personally identifiable information. We view more compact tool use as beneficial for practical deployment: fewer unnecessary invocations can lower latency, compute, and API cost, which may improve the usability of agent systems in everyday interactive settings. At the same time, efficiency-oriented optimization could be misused to accelerate harmful automated workflows. Our method does not introduce new attack capabilities, and we encourage deploying such agents under existing safety filters, access controls, and usage policies. We have no conflicts of interest beyond standard academic research, and we believe this work is consistent with the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the iterative DPO pipeline and the intra-trajectory pruning procedure in Secs. 3.1 and 3.2, with pseudocode in Appendix B. Sampling settings, LoRA/DPO hyperparameters, and the prune-stage schedule are reported in Appendix C. All experiments are conducted on the public BFCL v3 multi-turn benchmarks under the official evaluation protocol. Trained model weights and a public code repository will be released after the conclusion of the double-blind review process.

REFERENCES

Fei Bai, Yingqian Min, Beichen Zhang, Zhipeng Chen, Xin Zhao, Lei Fang, Zheng Liu, Zhongyuan Wang, and Hongteng Xu. Towards effective code-integrated reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 30022–30030, 2026.

Sijia Chen, Yibo Wang, Yi-Feng Wu, Qing-Guo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. Advancing tool-augmented large language models: Integrating insights from errors in inference trees. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 106555–106581. Curran Associates, Inc., 2024a. doi: 10.52202/079017-3382. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/c0f7ee1901fef1da4dae2b88dfd43195-Paper-Conference.pdf.

Yifei Chen, Guanting Dong, and Zhicheng Dou. Toward effective tool-integrated reasoning via self-evolved preference learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=mNeitRAdWV.

Zhi-Yuan Chen, Shiqi Shen, Guangyao Shen, Gong Zhi, Xu Chen, and Yankai Lin. Towards tool use alignment of large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 1382–1400, Miami, Florida, USA, November 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.82. URL https://aclanthology.org/2024.emnlp-main.82/.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and

Zhicheng Dou. Agentic reinforced policy optimization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=TX4k7BF6aO.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tRk1nofSmz.

Bingguang Hao, Maolin Wang, Zengzhuang Xu, Yicheng Chen, Cunyin Peng, Jinjie Gu, and Chenyi Zhuang. Exploring superior function calls via reinforcement learning. *arXiv e-prints*, pp. arXiv–2508, 2025.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.507. URL https://aclanthology.org/2023.emnlp-main.507/.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*, 2022. URL https://mlanthology.org/iclr/2022/hu2022iclr-lora/.

Wonjoong Kim, Sangwu Park, Yeonjun In, Sein Kim, Dongha Lee, and Chanyoung Park. Beyond the final answer: Evaluating the reasoning trajectories of tool-augmented agents, 2026. URL https://arxiv.org/abs/2510.02837.

Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic, and Hao Su. Deductive verification of chain-of-thought reasoning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 36407–36433. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/72393bd47a35f5b3bee4c609e7bba733-Paper-Conference.pdf.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 126544–126565. Curran Associates, Inc., 2024. doi: 10.52202/079017-4020. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/e4c61f578ff07830f5c37378dd3ecb0d-Paper-Conference.pdf.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Manoj Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/5e3661f7fe4c8ac5652d62eb3d3c96ea-Paper-Datasets_and_Benchmarks_Track.pdf.

Cheng Qian, Emre Can Acikgoz, Hongru Wang, Xiusi Chen, Avirup Sil, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. SMART: Self-aware agent for tool overuse mitigation. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 4604–4621, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.239. URL https://aclanthology.org/2025.findings-acl.239/.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 53728–53741. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a85b405ed65c6477a4fe8302b5e06ce7-Paper-Conference.pdf.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Jun-Mei Song, Mingchuan Zhang, Y. K. Li, Yu Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *ArXiv*, abs/2402.03300, 2024. URL https://api.semanticscholar.org/CorpusID:267412607.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of LLM agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7584–7600, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.409. URL https://aclanthology.org/2024.acl-long.409/.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 58202–58245. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/b5c8c1c117618267944b2617add0a766-Paper-Conference.pdf.

Sai H. Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. Chatgpt for robotics: Design principles and model abilities. *IEEE Access*, 12:55682–55696, 2024. doi: 10.1109/ACCESS.2024.3387941.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *International Conference on Learning Representations*, 2023. URL https://mlanthology.org/iclr/2023/wang2023iclr-selfconsistency/.

Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhenglin Wang, Zhengwei Tao, Ding-Chu Zhang, Zekun Xi, Robert Tang, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. Webdancer: Towards autonomous information seeking agency. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 120957–120985. Curran Associates, Inc., 2025a. doi: 10.52202/085713-4035. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/af043aee9cb137785d93195c6cf4cd96-Paper-Conference.pdf.

Peilin Wu, Mian Zhang, Xinlu Zhang, Xinya Du, and Zhiyu Chen. Search wisely: Mitigating sub-optimal agentic searches by reducing uncertainty. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 19723–19734, Suzhou, China, November 2025b. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.998. URL https://aclanthology.org/2025.emnlp-main.998/.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. Gpt4tools: Teaching large language model to use tools via self-instruction. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 71995–72007. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/e393677793767624f2821cec8bdd02f1-Paper-Conference.pdf.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822. Curran Associates, Inc., 2023a. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=WE_vluYUL-X.

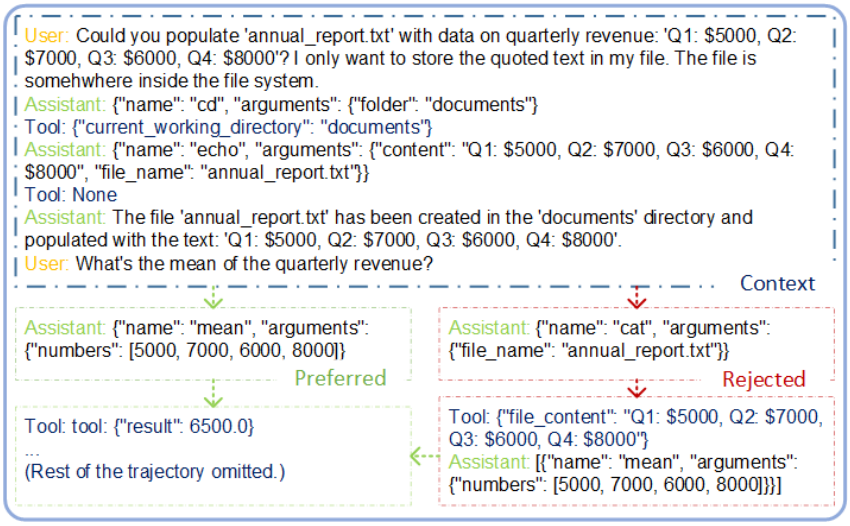
Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, volume 2025, pp. 9965–10017, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/1b126cc38b8638e07bef37e7b2bb72bf-Paper-Conference.pdf.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Zhiwei Liu, Yihao Feng, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. xLAM: A family of large action models to empower AI agent systems. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 11583–11597, Albuquerque, New Mexico, April 2025a. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.578. URL https://aclanthology.org/2025.naacl-long.578/.

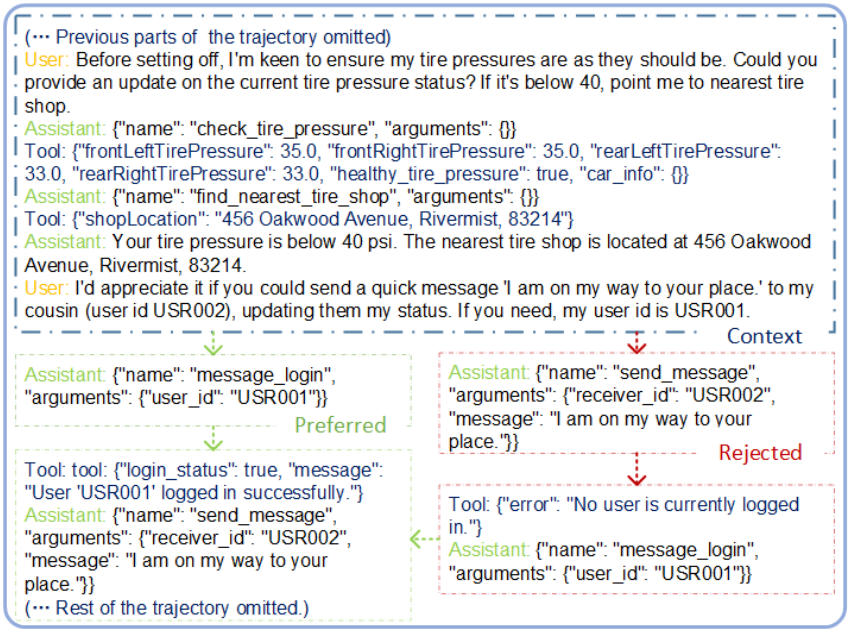
Kexun Zhang, Weiran Yao, Zuxin Liu, Yihao Feng, Zhiwei Liu, Rithesh Ramapura Narasimha Murthy, Tian Lan, Lei Li, Renze Lou, Jiacheng Xu, Bo Pang, Yingbo Zhou, Shelby Heinecke, Silvio Savarese, Huan Wang, and Caiming Xiong. Diversity empowers intelligence: Integrating expertise of software engineering agents. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 86691–86714, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/d7b50b8ac2c781a12f26155f48310d8d-Paper-Conference.pdf.

Xuhui Zheng, Kang An, Ziliang Wang, Yuhang Wang, and Yichao Wu. StepSearch: Igniting LLMs search ability via step-wise proximal policy optimization. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21805–21830, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1106. URL https://aclanthology.org/2025.emnlp-main.1106/.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan Rossi, Somdeb Sarkhel, and Chao Zhang. Toolchain* : Efficient action space navigation in large language models with a* search. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 4524–4549, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/13250eb13871b3c2c0a0667b54bad165-Paper-Conference.pdf.



(a) Redundant tool call. The rejected trajectory unnecessarily reads the file content before calling `mean`, although the required values are already available in the dialogue context.



(b) Unnecessary corrective step. The rejected trajectory attempts to send a message before logging in, triggers an error, and only then performs the missing login step.

Figure 3: Examples of inefficient tool-use behaviors used to derive step-level preference pairs.

A Examples of Inefficient but Successful Trajectories

Fig. 3 shows two typical inefficiencies used to derive step-level preference pairs. In Fig. 3a, the rejected trajectory invokes an extra retrieval tool despite already possessing the necessary information in the conversational context, thereby inflating the tool-call budget without improving task success. In Fig. 3b, an avoidable early mistake triggers an error observation and a subsequent recovery step, forming an unnecessary corrective loop that likewise lengthens the trajectory. Both rollouts eventually complete the user intent; the preferred action a_p is the compact next step under the shared prefix c, while a_r is the local choice that opens the detour.

Algorithm 1 Validated cut mining on a successful trajectory

Require: Successful trajectory τ; validity oracle Valid
Ensure: Autonomy / Redundancy / Error cuts; Repair combinations

$\tau' \leftarrow \tau$
for each turn t in τ' **do**
  for assistant tool action a from last to first in turn t **do**
    if $\mathrm{Valid}(\tau' \setminus \{s\})$ **then**
      mark a as AUTONOMY; remove a from τ'
    else
      break ▷ stop tail deletion for this turn
    end if
  end for
end for
$\mathcal{R} \leftarrow \emptyset$, $\mathcal{E} \leftarrow \emptyset$, $D \leftarrow \emptyset$
for each remaining tool call c in τ' from last to first **do**
  if $\mathrm{Valid}(\tau' \setminus (D \cup \{c\}))$ **then**
    $D \leftarrow D \cup \{c\}$
    if tool return of c is an error observation **then**
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{c\}$
    else
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{c\}$
    end if
  end if
end for
$\mathcal{P} \leftarrow \emptyset$
for each turn t with $\mathcal{E}_t \neq \emptyset$ **do**
  let C_t be residual calls in turn t after removing $\mathcal{R} \cup \mathcal{E}$
  for $k = 2$ to $|C_t| - 1$ **do**
    for each subset $S \subseteq C_t$ with $|S| = k$ **do**
      if $\mathrm{Valid}(\tau' \setminus (D \cup S))$ **then**
        $\mathcal{P} \leftarrow \mathcal{P} \cup \{S\}$
      end if
    end for
  end for
end for
return Autonomy cuts on τ; $\mathcal{R}$, $\mathcal{E}$, $\mathcal{P}$

B Cut Mining and Preference Pair Construction

Algorithms 1 and 2 formalize the intra-trajectory procedure in Sec. 3.2. Alg. 1 discovers Autonomy, Redundancy, Error, and Repair cuts under the BFCL multi-turn validity oracle, applied in that order on a greedy successful trajectory. Alg. 2 then walks each turn with a cleaned prefix c and converts changed actions into step-level pairs (c, a_p, a_r): partial parallel-call deletions yield retained vs. original call lists; whole-step deletions (including Autonomy) pair against the next retained action or the turn-final text and jump forward so later contexts never condition on removed observations.

C Experimental Setup

Candidate trajectories are generated on the **Base Multi-Turn** subset of BFCL v3[2]. We focus on **BitAgent-Bounty-8B** and **Llama-xLAM-2-8b-fc-r** (Zhang et al., 2025a), which achieve the highest accuracies among compact open models at the 8B scale. Notably, despite its modest size, **Llama-xLAM-2-8b-fc-r** already exceeds the strongest proprietary baseline on this subset, Claude-Opus-4-5-20251101 (FC).

[2] https://gorilla.cs.berkeley.edu/blogs/13_bfcl_v3_multi_turn.html

Algorithm 2 Step-level preference construction with cleaned prefixes

Require: Successful trajectory τ; Autonomy cuts $\mathcal{A}$; Redundancy/Error cuts $\mathcal{R} \cup \mathcal{E}$; Repair combinations $\mathcal{P}$
Ensure: Preference set $\mathcal{D} = \{(c, a_p, a_r)\}$
form per-turn deletion set Δ from $\mathcal{A} \cup \mathcal{R} \cup \mathcal{E} \cup \mathcal{P}$
$\mathcal{D} \leftarrow \emptyset$; $c \leftarrow$ empty prefix
for each turn t in τ **do**
  $i \leftarrow$ index of first assistant action in turn t
  while i points to an action in turn t **do**
    let a be the assistant action at i; let $a' \leftarrow a \setminus \Delta$ ▷ retained content
    if a is unchanged under Δ **then**
      append a and its tool returns (if any) to c; advance i
    else if a is an Autonomy cut **or** $a' = \emptyset$ **then**
      let a_p be the next retained action in turn t (turn-final text if Autonomy / no later tool step remains)
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(c, a_p, a)\}$
      append a_p and its returns (if any) to c; jump i to the action after a_p
    else
      ▷ partial parallel-call deletion
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(c, a', a)\}$
      append a' and the returns of retained calls to c; advance i
    end if
  end while
  append any remaining turn-final text (if not already in c) to c
end for
return $\mathcal{D}$

For each model, we collect self-generated samples via high-temperature decoding (K=6, T=0.7) on Base, construct step-level preference pairs, and apply DPO with **LoRA** (Hu et al., 2022) adapters (r=16, α=32, dropout 0.05). Optimization uses a learning rate of 1×10^{-5}, DPO β=0.1, and 10 epochs per iterative round. After the cross-trajectory loop plateaus, we apply an additional DPO stage with intra-trajectory validated pruning on greedy successful Base rollouts (Sec. 3.2): BitAgent-Bounty-8B still uses 10 epochs, whereas Llama-xLAM-2-8b-fc-r uses only 4 epochs to further cut excess tool calls while limiting accuracy regression.

D Token Cost and Inference Steps

Table 2 reports total token consumption (in millions) and inference steps on all four BFCL v3 multi-turn subsets. Consistent with the Exc. reductions in Table 1, both metrics generally decrease over iterative DPO and the final prune stage. On BitAgent, the largest absolute savings appear on Long-Context and Missing-Func, where the baseline trajectories are longest; Base tokens fall from 15.2M to 10.8M and Missing-Func steps from 3168 to 2206 after prune. On xLAM, changes are smaller in magnitude but still favor leaner rollouts (e.g., Base tokens 14.5M→14.0M; Missing-Func steps 2495→2384).

The main-text analysis of Figs. 2a and 2b already notes that BitAgent and xLAM differ in how they pack tool calls into inference steps: BitAgent favors parallel invocation within a single assistant turn, whereas xLAM tends to serialize tool use. This distinction further motivates our preference-construction choice. Because xLAM rarely exhibits parallel tool-call patterns, Rule 2 provides limited benefit, and we construct xLAM preferences with **Rule 1 alone**; BitAgent benefits from the full Rule 1+Rule 2 pipeline in Sec. 3.1.

Table 2: Total tokens (Tok., in millions) and inference steps (Steps) on BFCL v3 multi-turn subsets across DPO stages. Lower is better for both metrics.

Model	Stage	Base		Miss-Param		Missing-Func		Long-Context	
		Tok.	Steps	Tok.	Steps	Tok.	Steps	Tok.	Steps
BitAgent	base	15.2	2016	18.7	2424	24.8	3168	48.9	2472
	iter1	12.4	1697	14.7	1996	20.1	2615	28.6	1902
	iter2	11.1	1547	14.0	1907	17.0	2291	26.9	1746
	iter3	11.1	1548	13.7	1873	16.8	2274	27.1	1756
	iter4	11.0	1528	13.4	1855	16.9	2291	25.2	1725
	prune	10.8	1509	13.5	1857	16.1	2206	24.9	1686
xLAM	base	14.5	2005	17.2	2360	17.8	2495	22.4	1976
	iter1	14.1	1953	16.8	2329	17.3	2433	22.1	1942
	iter2	14.0	1937	16.7	2315	16.9	2393	22.3	1941
	iter3	13.9	1934	16.5	2281	17.0	2401	22.3	1934
	prune	14.0	1937	16.6	2294	16.8	2384	22.3	1940