PROGRA: Progress-Aware Reinforcement Learning for Multi-Turn Function Calling

Huacan Chai[1], **Zijie Cao**[1], **Maolin Ran**[1], **Yingxuan Yang**[1], **Jianghao Lin**[1], **Peng Xin**[2], **Hairui Wang**[2], **Renjie Ding**[2], **Ziyu Wan**[1], **Muning Wen**[1], **Weiwen Liu**[1], **Fei Huang**[2,*], **Weinan Zhang**[1,3,*], **Ying Wen**[1,3,*]
[1]Shanghai Jiao Tong University, China;
[2]LongShine AI Research, China;
[3]Shanghai Innovation Institute, China
{fatcat, wnzhang, ying.wen}@sjtu.edu.cn, huangfei@longshine.com

Abstract

Real-world tasks with Large Language Models (LLMs) require multi-turn, multi-step conversations, often involving complex function calls and multiple user interactions. Existing methods either decompose multi-turn trajectories into independent single samples, neglecting task-level structure, or rely on end-to-end reinforcement learning (RL) without explicit progress modeling. To overcome these limitations, we propose PROGRA, a framework that explicitly incorporates progress awareness into LLM training for multi-turn function calling. PROGRA combines a Progress Awareness Generation pipeline to construct training data linking conversation summaries with future plans, and Progress Awareness-Guided RL to condition decisions on this awareness, reducing redundancy and aligning local actions with global task completion. Experiments on two public benchmarks show that PROGRA substantially outperforms prior methods for robust, efficient multi-turn function calling. Our code is available at https://github.com/FatCatCHC/Progra.

1 Introduction

Large language models (LLMs) have made substantial advances in tool use, where invoking external APIs extends both their reasoning and execution capabilities (Lin et al., 2025b; Feng et al., 2025; Acikgoz et al., 2025; Zhou et al., 2026; Yang et al., 2025b; Ge et al., 2025). While recent work has improved the reliability of function calling in *single-turn conversations*, real-world applications rarely operate in isolation (Alkhouli et al., 2025; Lu et al., 2025; GLM-5-Team et al., 2026; Yang et al., 2025a). Instead, tasks such as travel planning and enterprise workflows naturally unfold over *multi-turn conversations*, where each turn influences future interactions and collectively shapes the overall task trajectory; a single inaccurate step may propagate downstream, leading to compounding errors or even task failure (Liu et al., 2025a; Lin et al., 2025a). This temporal interdependence makes it insufficient to optimize only turn-level accuracy; instead, it necessitates *progress awareness*, namely a coherent representation of task state that supports long-horizon execution (Sanders et al., 2022; Yin et al., 2025). Concretely, progress awareness entails (i) an accurate **summary** of interaction history, which mitigates contextual redundancy and informs decision-making, and (ii) explicit **planning** of future actions, which aligns intermediate steps with overall task objectives (Rastogi et al., 2020). Limited capacity for progress awareness causes LLMs to struggle with managing long-horizon dependencies in conversations. This leads to behaviors such as repeated function calls or omitted parameters, creating a bottleneck for improvements in multi-turn function calling.

Despite the significant impact, current methods for multi-turn function calling largely fail to endow LLMs with such progress awareness, creating a bottleneck for performance. Existing approaches to enhancing multi-turn function calling mainly rely on fine-tuning LLM with a carefully curated dataset (Prabhakar et al., 2025a; Yin et al., 2025). In practice, these approaches often reconstruct multi-turn conversation datasets into samples consisting solely of single-turn function calls, training the model to improve its single-turn accuracy. This paradigm not only lacks the diversity and dynamism of real-world scenarios but also, by degrading multi-turn task execution to single-turn question-answer predictions, neglects the awareness of the overall task progress during training (Sanders et al., 2022). By excluding future turns, these approaches hinder the development of task planning capabilities. Moreover, a focus on single-turn accuracy can lead models to overlook historical context that, while not immediately

*Corresponding authors.

Findings of the Association for Computational Linguistics: ACL 2026, pages 6519–6535
July 2-7, 2026

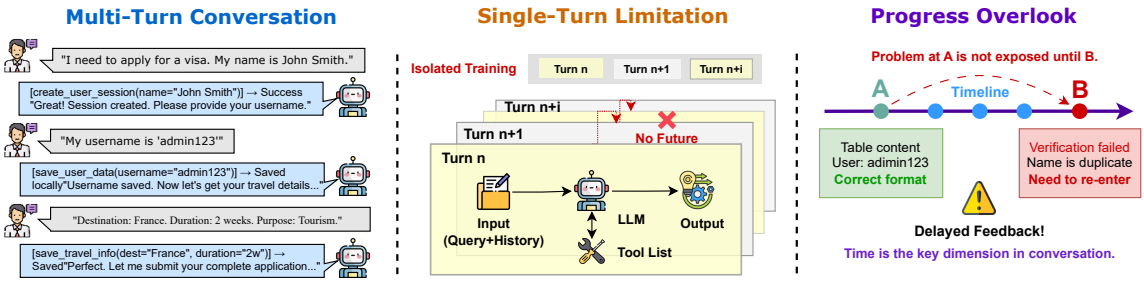

Figure 1: Fragmenting multi-turn conversations into single-turn samples breaks temporal continuity and prevents models from tracking task progress, leading to the ignorance of historical state and poor planning in later turns.

relevant, could be essential in future interactions, thereby limiting the model's ability to summarize past conversations effectively.

Another approach involves leveraging end-to-end reinforcement learning (RL) to optimize long-horizon returns in multi-turn interactions (Singh et al., 2025; Chen et al., 2025b; Zhang et al., 2025). Recent RL-based methods have enhanced function calling by modeling it as sequential decision-making. However, the end-to-end RL paradigm remains inefficient: as conversations expand, the growing redundancy in input contexts leads to increasingly convoluted decision-making and slower optimization. More critically, existing RL methods still lack explicit integration of progress awareness, limiting their ability to align local actions with global task goals effectively (Wang et al., 2025a).

To overcome the challenge of neglecting progress awareness in multi-turn conversations, we introduce PROGRA, a framework designed to explicitly integrate progress awareness into model training for enhanced multi-turn function calling. PROGRA comprises two key components: (i) the **P**rogress **A**wareness **G**eneration (PAG) pipeline, and (ii) the **P**rogress **A**wareness **G**uided **R**einforcement **L**earning (PAG-RL) approach. PAG automatically constructs a high-quality dataset that combines conversation history summaries with future task planning through a synthetic pipeline, thereby strengthening the ability of LLMs to generate progress awareness. Subsequently, PAG-RL leverages this awareness to guide end-to-end reinforcement learning, alleviating contextual redundancy and optimizing decision-making efficiency in real-world settings. Overall, our contributions focus on three key aspects:

1. To the best of our knowledge, PROGRA is the first work that identifies, formulates, and explicitly incorporates task progress awareness into training of LLMs for multi-turn function calling.
2. We design a novel progress awareness generation pipeline for automatically providing high-quality datasets to improve the capability of LLMs in progress awareness.
3. We design a progress awareness guided reinforcement learning algorithm, which enhances the model's training performance by explicitly incorporating progress awareness into end-to-end RL training, outperforming existing strategies on two public benchmarks.

2 Related Work

2.1 Function Calling

Recent studies on the function-calling capabilities of large language models have increasingly transitioned from focusing on *single-turn* invocations (Liu et al., 2024) to exploring *multi-turn* scenarios (Chen et al., 2025a). While ToolLLM (Qin et al., 2023) constructs a large-scale dataset of massive real-world APIs. APIGen-MT (Prabhakar et al., 2025b) further develops a two-phase agentic pipeline that synthesizes verifiable multi-turn trajectories from blueprint tasks. These methods address the *data bottleneck* for training, yet they remain essentially *data-driven*, lacking explicit modeling of global task progress. Another direction of enhancing the function calling ability is at prompt-engineering level, including reasoning–acting interleaving (Yao et al., 2023b), structured branching (Yao et al., 2023a), and self-reflection (Shinn et al., 2023) improve local robustness and error correction. However, these approaches emphasize local reasoning persistence rather than an explicit, evolving *progress progress* that connects intermediate calls with final task completion. In contrast, PROGRA aligns function call accuracy with overall task execution by explicitly incorporating progress

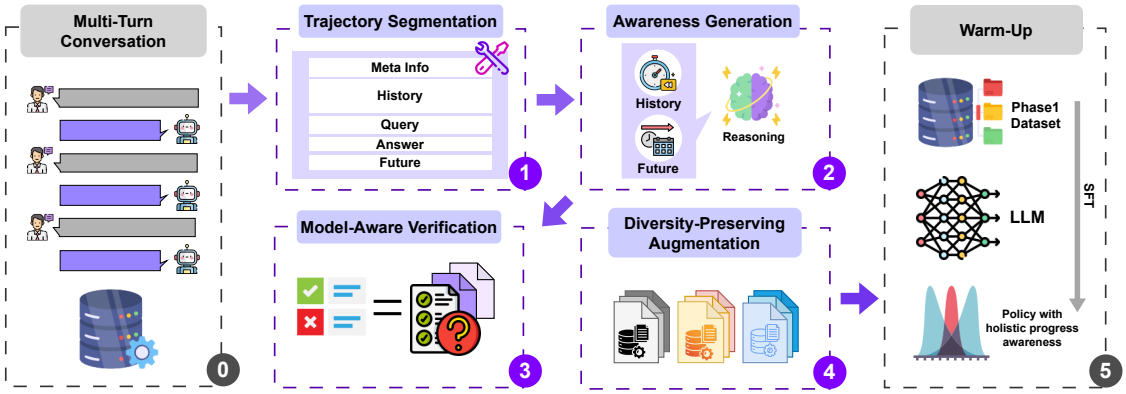

Figure 2: Overview of PAG. By summarizing history and planning future in Step 2, PAG generates progress awareness for each conversation turn and constructs a high-quality awareness dataset.

awareness into model training.

2.2 Multi-Turn Reinforcement Learning

With the rapid development of reinforcement learning, several works introduce RL for multi-turn interaction. ARTIST (Singh et al., 2025) integrates outcome-based RL with dynamic tool routing, while RLFactory (Chai et al., 2025) offers a modular post-training pipeline for multi-turn orchestration. More recently, turn-level credit assignment (Zeng et al., 2025) and conversation-level preference optimization (Shi et al., 2024) explicitly frame multi-turn tool use as a sequential decision process, addressing delayed reward signals. On the algorithmic side, Group Relative Policy Optimization (GRPO) (Shao et al., 2024a) eliminates the critic through within-group normalization, enabling more stable and efficient updates. Many recent multi-turn RL studies have adopted GRPO to stabilize training in sparse-reward (Mroueh et al., 2025) or process reward (Zheng et al., 2025a,b; Zhu et al., 2025). Despite these advances, existing RL agents rarely maintain an explicit *task progress awareness*. Without such awareness, agents often repeat calls or omit critical steps in long-horizon workflows. Our approach differs by introducing *progress aware guidance* into the multi-turn RL, thereby reducing context redundancy and aligning local action choices with global task execution.

3 Methodology

In this section, we will provide a detailed introduction to problem formulation in multi-turn function calling, following by two phases of PROGRA, namely **PAG** and **PAG-RL**.

3.1 Problem Formulation and Notation

We cast multi-turn function calling as a Markov Decision Process (MDP):

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P_E, r, \gamma, \rho_0, H), \tag{1}$$

where $\mathcal{S}$ is the space of conversation prefixes (states), $\mathcal{A}$ is the textual action space (function invocations or user-facing messages), P_E is the transition kernel in environment E, r is the step reward, $\gamma \in (0, 1]$ is the discount factor, ρ_0 is the initial state distribution, and H is the horizon. A trainable LLM π_θ acts as the stochastic policy. The conversation is indexed by turns i (delimited by user queries Q_i) and intra-turn steps $j \in \{1, \ldots, T_i\}$. The state at (i, j) is the entire conversation prefix:

$$S_{i,j} = \{(Q_k, \{(A_{k,\ell}, O_{k,\ell})\}_{\ell=1}^{T_k}, A_k^{\text{msg}})\}_{k=1}^{i-1} \cup (Q_i, \{(A_{i,\ell}, O_{i,\ell})\}_{\ell=1}^{j-1}), \tag{2}$$

where $A_{i,j} \in \mathcal{A}$ is either a structured function call with arguments or a user-facing message A_i^{msg} that ends turn i. $O_{i,j}$ is the observation from E after applying action $A_{i,j}$. In step j, the policy will sample an action and receive feedback from E:

$$A_{i,j} \sim \pi_\theta(\cdot \mid S_{i,j}), \tag{3}$$

$$(O_{i,j}, r_{i,j}) \sim P_E(\cdot \mid S_{i,j}, A_{i,j}), \tag{4}$$

and the conversation appends $(A_{i,j}, O_{i,j})$ to form $S_{i,j+1}$. A trajectory τ concatenates turns until solving K user queries (or reach H steps):

$$\tau = \{(Q_i, \{(A_{i,j}, O_{i,j}, r_{i,j})\}_j^{T_i}, A_i^{\text{msg}})\}_i^K, \tag{5}$$

$$R(\tau) = \sum_{i=1}^{K} \sum_{j=1}^{T_i} \gamma^{t(i,j)} r_{i,j}, \tag{6}$$

where $t(i, j)$ is the global step index. The learning objective is to maximize $\mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$.

3.2 Phase 1: Progress Awareness Generation

Progress Awareness Generation (PAG) is designed to automatically synthesize a high-quality progress awareness dataset and warm-up π_θ, containing trajectory segmentation, awareness generation, model-aware verification, diversity-preserving augmentation and model warm-up.

Trajectory Segmentation. Given a multi-turn dataset $\mathcal{D}$ containing trajectories formatted as in Eq. (5), we segment each trajectory τ at each assistant response and only those where the assistant's response is a function call will be retained. Each segmentation becomes an instance:

$$\tau' = (\textit{info}^{\text{meta}},\, c^{\text{his}},\, q,\, a^{\text{fc}},\, c^{\text{fut}}), \tag{7}$$

where $\textit{info}^{\text{meta}}$ contains tool descriptions/schema and the scenario description, c^{his}/c^{fut} are the conversation contexts before/after the current step, q is the current user query, and a^{fc} is the ground-truth function call for this step.

Awareness Generation. Given these segmented instance, an off-the-shelf generator LLM_{gen} is employed to generate a compact textual *awareness document* S^a for each τ':

$$S^a = LLM_{\text{gen}}(\tau';\, \text{prompt}), \tag{8}$$

where the prompt elicits three components: (i) a concise *history summary* capturing user intent, function calling history, and important arguments; (ii) a short *future plan*, including anticipated function sequence and decision points; (iii) minimal *rationale* that links history to plan. We denote the raw corpus as $\mathcal{D}^{\text{raw}} = \{S^a\}$.

Model-Aware Verification. Although LLM_{gen} is typically a highly capable LLM, the quality of the generated awareness still requires validation. At this stage, based on the principle that *an ideal progress awareness should contain the necessary information to reconstruct the answer*, we introduce a Model-Aware Verification operation. In this stage, a frozen copy of the target policy is employed as LLM_{ver}. It attempts to recover the function call solely from S^a in the absence of the original conversation:

$$\hat{a}^{\text{fc}} = LLM_{\text{ver}}(S^a, \textit{info}^{\text{meta}}). \tag{9}$$

Subsequently, a normalized equivalence predicate $\text{Eq}(\cdot, \cdot)$ checks schema-level equality (argument order invariance, whitespace-insensitive strings, commutative sets/lists):

$$\text{Eq}(\hat{a}^{\text{fc}}, a^{\text{fc}}) = \mathbb{I}[\text{schema\_equal}(\hat{a}^{\text{fc}}, a^{\text{fc}})], \tag{10}$$

only instances with $\text{Eq} = 1$ are retained, yielding $\mathcal{D}^{\text{ver}} = \{S^a \in \mathcal{D}^{\text{raw}} \mid \text{Eq}(\hat{a}^{\text{fc}}, a^{\text{fc}}) = 1\}$.

Diversity-Preserving Augmentation. To mitigate lexical overfitting and improve robustness, an augmenter LLM_{aug} is applied to perform semantic-level transformations over each verified S^a with a randomly sampled operation type $\in$ {paraphrase, schema-perturb, word-mask}:

$$\tilde{S}^a = LLM_{\text{aug}}(S^a; \text{type}), \tag{11}$$

yielding $\mathcal{D}^{\text{aug}} = \{\tilde{S}^a\}$. Specifically, the instructions for each augmentation operation are as follows: (1) paraphrase: paraphrasing user intents to introduce lexical and syntactic variation; (2) schema-perturb: modifying function names and parameter values within permissible ranges to simulate realistic perturbations; (3) word-mask: applying random masking to function-related words to enhance robustness to incomplete or noisy inputs.

Consequently, a high-quality dataset containing strong progress awareness $\mathcal{D}^{\text{aug}}$ is obtained.

Model Warm-Up. To facilitate high-quality progress awareness generation during the next phase, we employ the aforementioned $\mathcal{D}^{\text{aug}}$ for model warming up. In order to enhance the familiarity with function call structure, a lightweight cold-start dataset $\mathcal{D}^{\text{cs}}$ is additionally curated, extracted from the original training set $\mathcal{D}$ and adhering well-formed function-call exemplars (no awareness text). The final SFT dataset is given by $\mathcal{D}^{\text{sft}} = \mathcal{D}^{\text{aug}} \cup \mathcal{D}^{\text{cs}}$.

As a result of this stage, $\pi_{\theta'}$ is obtained with a learned ability to summarize history and outline a plan for the future, namely aforementioned *progress awareness*.

3.3 Phase 2: Progress Awareness-Guided RL

After strengthening the LLM's progress awareness via the PAG stage, we introduce Progress Awareness Guided RL (PAG-RL), which explicitly incorporates progress awareness into end-to-end reinforcement learning, aiming to improve the model's effectiveness in realistic scenarios. This section

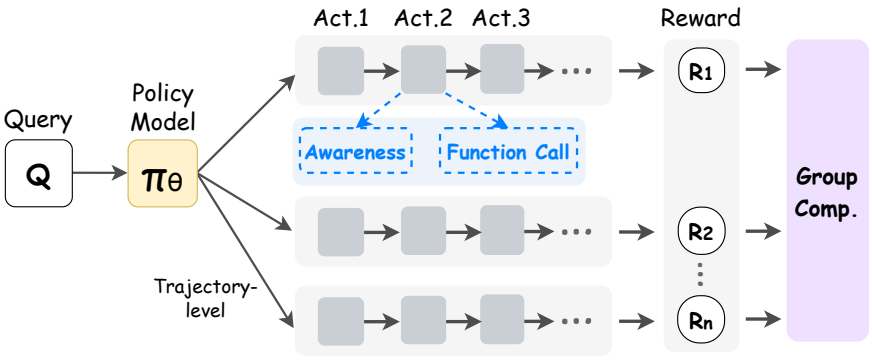

Figure 3: Overview of PAG-RL. By incorporating progress awareness, the policy model generates multi-step action trajectories in response to a query.

will be organized with *awareness-guided rollout* and *composite reward*, and *optimization procedure*.

3.3.1 Awareness-Guided Rollout

Following the formulation in Sec. 3.1, the state $S_{i,j}$ includes all prior interactions, including the user query Q_i, the sequence of all past actions $\{A_{\leq j}\}$, and corresponding observations $\{O_{\leq j}\}$ in rollout. At each intra-turn step (i, j), instead of conditioning on the entire raw prefix, $\pi_{\theta'}$ first emits a compact progress awareness as:

$$S^a_{i,j} \sim \pi_{\theta'}(\cdot \mid \mathit{info}^{\text{meta}}, Q_i, \{(A_{\leq j-1}, O_{\leq j-1})\}).$$

Conditioned on $S^a_{i,j}$, $\pi_{\theta'}$ then generates the action $A_{i,j+1}$ in a chain-of-thought (CoT) style:

$$A_{i,j+1} \sim \pi_{\theta'}(\cdot \mid \mathit{info}^{\text{meta}}, Q_i, S^a_{i,j}), \quad (12)$$
$$= a^{\text{think}}_{i,j+1} \oplus a^{\text{fc}}_{i,j+1}, \quad (13)$$

where textual action $A_{i,j+1}$ includes intermediate reasoning $a^{\text{think}}_{i,j+1}$ and the function call $a^{\text{fc}}_{i,j+1}$, wrapped by `<think>` and `<answer>`. After $a^{\text{fc}}_{i,j+1}$ executed in E, $(O_{i,j}, r_{i,j})$ will be returned and the rollout proceeds. The rollout will terminate either upon all queries completed or reaching the maximum number of interactions, obtaining a trajectory τ, after which the reward $R(\tau)$ of the trajectory is computed as defined by Equ. 5.

3.3.2 Reward Design

Each action in rollout will receive a composite reward including textual structure, function schema, task success, and execution efficiency, which can be formulated as:

$$r_{i,j} = \alpha_{\text{fmt}} \underbrace{\mathbb{I}[\textsc{Template}(A_{i,j})]}_{r^{\text{fmt}}} + \alpha_{\text{schema}} \underbrace{\mathbb{I}[\textsc{Schema}(A_{i,j})]}_{r^{\text{schema}}}$$
$$+ \alpha_{\text{acc}} \underbrace{\mathbb{I}[\textsc{Success}(S^E_{i,j} \Rightarrow S^E_{i,j+1})]}_{r^{\text{acc}}} - \lambda \underbrace{\mathbb{I}[A_{i,j} \neq \emptyset]}_{r^{\text{pen}}}, \quad (14)$$

with $\alpha_{\text{fmt}}, \alpha_{\text{schema}}, \alpha_{\text{acc}}, \lambda > 0$. Template enforces the output tags in Eq. 12, Schema checks the function name and argument types/values, and Success verifies task completion. If the executed call satisfies the user query and brings the environment to the target state, we grant a positive reward r^{acc}. The penalty term r^{pen} controls rollout length, encouraging efficient task completion.

3.3.3 Optimization Procedure

After obtaining multi-turn trajectories through rollout and computing the corresponding trajectory-level rewards based on the composite reward function, the Group Relative Policy Optimization (GRPO (Shao et al., 2024b)), a critic-free variant of PPO, is applied to optimize the policy.

Given a batch of L trajectories $\{\tau_\ell\}_{\ell=1}^{L}$ rolled out by π_{old}, we normalize the scalar return $R(\tau_\ell)$:

$$\hat{A}_\ell = \frac{R(\tau_\ell) - \mu_R}{\sigma_R},$$
$$\mu_R = \tfrac{1}{L}\sum_\ell R(\tau_\ell), \quad \sigma_R^2 = \tfrac{1}{L}\sum_\ell (R(\tau_\ell) - \mu_R)^2.$$

Following GRPO, the same normalized advantage $\hat{A}_\ell$ is evenly assigned to all tokens in τ_ℓ in an concated textual action representation:

$$A^{\text{con}}_{i,j} = S^a_{i,j} \oplus a^{\text{think}}_{i,j} \oplus a^{\text{fc}}_{i,j}. \quad (15)$$

Let $\tau_{\ell,(t)}$ be the t-th token and $\tau_{\ell,<t}$ its prefix (which includes `<sum>`, `<think>`, and `<answer>` regions). The objective is

$$J_{\text{GRPO}}(\theta) = \frac{1}{L}\sum_{\ell=1}^{L}\frac{1}{|\tau_\ell|}\sum_{t=1}^{|\tau_\ell|}$$
$$\min\left(\frac{\pi_\theta(\tau_{\ell,(t)} \mid \tau_{\ell,<t})}{\pi_{\text{old}}(\tau_{\ell,(t)} \mid \tau_{\ell,<t})}\hat{A}_\ell,\ \text{clip}\Big[\frac{\pi_\theta}{\pi_{\text{old}}}, 1-\epsilon, 1+\epsilon\Big]\hat{A}_\ell\right)$$
$$-\beta_{\text{KL}}\,\text{KL}\big[\pi_\theta \,\|\, \pi_{\theta'}\big],$$

where the optional KL term regularizes the policy towards the SFT reference $\pi_{\theta'}$ to prevent reward hacking; ϵ and β_{KL} are hyperparameters.

4 Experiment

4.1 Research Questions

To rigorously evaluate the effectiveness of PROGRA, we propose the following research questions:

- Does PROGRA consistently outperform alternative training strategies across diverse datasets and backbone LLMs?
- What is the relative contribution of each stage and component of PROGRA to the overall performance gains?
- To what extent can models trained with progress awareness data demonstrate enhanced capability in progress-guided decision-making?
- Does PROGRA perform better than other baselines on unseen domain?
- Does the performance gains of PROGRA come from the reliance on a supervisor of strong LLM?
- How well does PROGRA perform in real multi-turn conversation scenarios (case study)?

4.2 Experiment Setup

Benchmark & Evaluation. We evaluate the performance of PROGRA on two widely used multi-turn function calling benchmarks: BFCL-V3 Multi-Turn (BFCL) (Patil et al., 2024) and τ-Bench (Yao et al., 2024). For BFCL, we adopt 3 subsets: *Base*, *Miss Functions* (*Miss. F.*) and *Miss Parameters* (*Miss. P.*), which respectively provide a standard and an augmented evaluation setting. We follow the benchmark's official metric, *Executable Function Accuracy*. For τ-Bench, we evaluate on the *airline* and *retail* scenarios. GPT-4o is used as the user simulator. Each evaluation sample is tested across 3 trials and averaged results reported.

Backbone LLMs & Baselines. We adopt Qwen2.5-7B-Instruct (Team, 2024), xlam2-3B and xlam2-8B (Prabhakar et al., 2025b) as backbone LLMs. These models span different sizes and architectures while exhibiting strong performance in function calling. We compare PROGRA against representative inference and training strategies designed to improve multi-turn function calling: (1) **Reasoning:** Enhances function call accuracy by prompting the model to generate chain-of-thought (CoT) reasoning. (2) **SFT:** Trains the model via supervised fine-tuning on multi-turn conversation data. (3) **ST-GRPO/Dr.GRPO:** The segmented multi-turn conversation samples are directly used for training with GRPO and Dr.GRPO(Liu et al., 2025b). (4) **MT-GRPO/Dr.GRPO:** Vanilla multi-turn GRPO and Dr.GRPO algorithm, following the RAGEN (Wang et al., 2025b) framework, which computes trajectory-level token advantages and applies the same reward setting as PROGRA.

4.3 Experiment Details

During the PAG data synthesis phase, we use GPT-4o as both LLM_{gen} and LLM_{aug}. For each data entry, we allow a maximum of 4 iterations and employ 5 reviewers in APIGen-MT (Prabhakar et al., 2025b) to assess the executability and correctness of the generated data.

During the warm-up stage in PAG, we employed supervised fine-tuning (SFT) using the LoRA (Low-Rank Adaptation) approach. We conducted hyperparameter searches over the following ranges: the LoRA rank was varied from 8 to 16; the learning rate ranged from 1e-7 to 1e-6; the batch size was set between 8 and 16; and the LoRA α (alpha) parameter was searched within the range of 8 to 64. A held-out validation set was used to evaluate whether the large language model (LLM) had been successfully trained during this warm-up phase.

In the reinforcement learning stage, we adopted the GRPO algorithm to optimize the policy. A KL-divergence penalty with a coefficient of $\beta = 0.001$ was used. The training was performed with a batch size of 8 and 8 rollout trajectories per batch. The maximum sequence length was limited to 16,384 tokens, and the temperature for the rollout generation was set to 1. Each training experiment was limited at 200 iterations, with a maximum of 10 actions per rollout. We used the LoRA-based training strategy during this phase. Hyperparameter tuning was conducted over the following ranges: LoRA rank between 8 and 16, LoRA α between 32 and 64, and learning rate from 1e-9 to 1e-5.

Model	Method	BFCL				τ-Bench		
		Overall	*Base*	*Miss F.*	*Miss P.*	*Overall*	*Airline*	*Retail*
Qwen2.5-7B	Base Model	8.33	9.50	8.50	7.00	16.60	8.00	25.20
	Reasoning	10.00	12.00	**11.50**	6.50	18.00	10.00	26.00
	SFT	8.67	11.50	7.50	7.00	21.30	**20.00**	22.61
	ST-GRPO	8.33	12.50	7.00	5.50	18.19	10.00	26.38
	ST-Dr.GRPO	8.72	12.67	7.50	6.00	18.33	10.00	26.67
	MT-GRPO	8.67	11.17	7.67	7.17	19.00	11.33	26.67
	MT-Dr.GRPO	8.17	**14.50**	6.50	3.50	18.52	10.67	26.38
	PROGRA	**10.44**	12.67	11.00	**7.67**	**23.91**	**20.00**	**27.83**
xLAM-2-3B	Base Model	58.67	66.50	55.00	54.50	25.45	24.00	26.90
	Reasoning	59.17	67.50	55.00	55.00	22.85	26.00	21.70
	SFT	59.17	67.00	55.50	55.00	23.30	24.00	22.61
	ST-GRPO	59.00	67.67	54.00	55.33	25.84	27.33	24.35
	ST-Dr.GRPO	59.22	**68.00**	54.17	**55.50**	25.81	24.67	26.96
	MT-GRPO	58.89	66.67	55.00	55.00	26.35	24.00	28.70
	MT-Dr.GRPO	**59.50**	**68.00**	55.50	55.00	24.32	24.00	24.64
	PROGRA	**60.00**	**68.00**	**57.00**	55.00	**31.52**	**30.00**	**33.04**
xLAM-2-8B	Base Model	69.25	74.75	69.25	63.75	46.70	35.20	58.20
	Reasoning	67.67	72.83	68.33	61.83	50.10	**42.00**	58.20
	SFT	64.72	75.00	55.17	64.00	46.57	34.00	59.13
	ST-GRPO	69.17	74.00	69.00	64.50	48.19	36.67	59.71
	ST-Dr.GRPO	69.33	74.50	69.50	64.00	50.54	39.33	61.74
	MT-GRPO	64.56	74.67	54.83	64.17	49.52	39.33	59.71
	MT-Dr.GRPO	64.89	74.83	55.67	64.17	46.94	37.33	56.54
	PROGRA	**69.56**	**75.67**	**69.83**	**64.50**	**51.85**	38.00	**65.70**

Table 1: Performance of PROGRA and other methods across different benchmarks and LLMs. Within each group, the **bolded** values denote the best performance, while the underlined values indicate the second-best performance. *Overall* denotes the average score across different categories.

4.4 Effectiveness of PROGRA

We evaluate PROGRA against existing training strategies on three backbone LLMs across two benchmarks as Table 1, revealing the following key conclusions: 1) As shown in Table 1, PROGRA consistently outperforms all baselines (e.g., SFT, MT-GRPO) in terms of *Overall* performance across all three backbones and both benchmarks, indicating that it delivers stable gains over diverse datasets and model architectures. 2) As model size increases from xLAM-2-3B to xLAM-2-8B, the improvement of PROGRA on benchmarks gradually decreases, indicating that larger LLMs already possess strong multi-turn function calling capabilities; nevertheless, PROGRA still provides substantial and stable additional improvement on strong backbones. 3) On τ-Bench, which emphasizes long-horizon planning, the performance gains of PROGRA over the baselines are more promising, implying that explicit progress modeling becomes increasingly advantageous when tasks rely heavily on extended interaction history and complex decision chains.

4.5 Ablation Study on Each Stage

To examine the contribution of each stage and component of PROGRA, we conduct detailed ablation experiments on BFCL and τ-Bench with three backbones. The results are summarized in Table 2 and reveling the following conclusions: 1) Across all models and benchmarks, complete PROGRA consistently outperforms, highlighting the essential role of integrating progress awareness training with RL. 2) Table 2 shows that excluding either PAG or PAG-RL consistently lowers both BFCL and τ-Bench scores across model scales. The drop is more pronounced when removing PAG-RL, underscoring that RL contributes the largest share of the improvement, while PAG alone also provides steady gains. Replacing PAG-RL with vanilla multi-turn GRPO yields only partial benefits, confirming the necessity of our tailored reinforcement stage. 3) Using only PAG-RL does not surpass the performance of the vanilla MT-GRPO. However, when comparing PAG + MT-GRPO against the full PROGRA, we observe that the progress aware-

Model	Method	BFCL	τ-Bench	Δ Avg.
Qwen2.5-7B	Base Model	8.25	16.60	-
	MT-GRPO	9.17	19.00	13.3%
	w/o PAG-RL	9.00	15.61	-1.0%
	w/o PAG	9.25	19.00	13.6%
	PAG + MT-GRPO	9.00	20.48	18.6%
	PROGRA	**10.33**	**23.91**	37.7%
xLAM-2-3B	Base Model	60.50	25.45	-
	MT-GRPO	60.83	26.35	1.4%
	w/o PAG-RL	60.50	25.91	0.5%
	w/o PAG	61.00	26.04	1.3%
	PAG + MT-GRPO	60.92	28.53	4.1%
	PROGRA	**60.00**	**31.52**	8.1%
xLAM-2-8B	Base Model	69.25	46.70	-
	MT-GRPO	64.56	49.52	2.6 %
	w/o PAG-RL	67.33	50.70	1.8%
	w/o PAG	69.00	50.44	3.0%
	PAG + MT-GRPO	67.50	46.84	-1.4%
	PROGRA	**69.56**	**51.85**	5.1%

Table 2: Ablation study of incremental gains across stages. Δ Avg.' is improvement over base models on the *overall* metric. 'w/o' means this stage is removed. 'PAG+MT-GRPO' replaces PAG-RL with MT-GRPO.

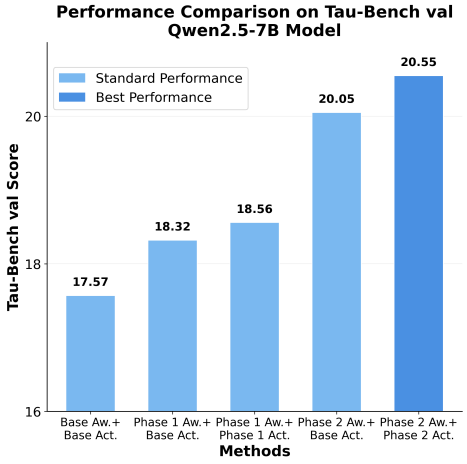

Figure 4: Awareness capability across phases. Ablation on the reserved τ-Bench dev set with Qwen2.5-7B-Instruct at three training stages (Base / Phase-1 / Phase-2), using each variant as both awareness generator ('Aw.') and action verifier ('Act.').

Model	Method	*Overall*	*Airline*	*Retail*	*Telecom*
Qwen2.5-7B	Base Model	16.60	8.00	25.20	13.62
	SFT	21.30	20.00	22.61	10.82
	SFT+MT-GRPO	18.32	12.00	24.64	14.62
	ST-GRPO	18.19	10.00	26.38	15.94
	ST-Dr.GRPO	18.33	10.00	26.67	14.49
	MT-GRPO	19.00	11.33	26.67	12.87
	MT-Dr.GRPO	18.52	10.67	26.38	12.28
	Progra	**23.91**	**20.00**	**27.83**	**16.23**
xLAM-2-8B	Base Model	46.70	35.20	58.20	10.14
	SFT	46.57	34.00	59.13	9.65
	SFT+MT-GRPO	48.96	36.00	61.92	10.82
	ST-GRPO	48.19	36.67	59.71	10.72
	ST-Dr.GRPO	50.54	39.33	61.74	10.82
	MT-GRPO	49.52	39.33	59.71	10.82
	MT-Dr.GRPO	46.94	37.33	56.54	10.23
	Progra	**51.85**	**38.00**	**65.70**	**13.91**

Table 3: Generalization Experiments of methods across backbone models. *Overall* refers to the average score across the *Airline* and *Retail* subsets, while *Telecom* is the out-of-domain test set.

ness introduced by PAG substantially enhances the LLM's capabilities. Moreover, continuing with PAG-RL leads to even greater improvements. For example, on τ-Bench with xLAM-2-8B, replacing PAG-RL with MT-GRPO results in a 10.69% performance drop.

4.6 Guidance of Progress Awareness

To validate the impact on the improvement of progress awareness after each training phase in PROGRA, we conducted experiments on the reserved τ-Bench validation set using Qwen2.5-7B-Instruct after different phases. With the action verifier fixed to Base Act., replacing the awareness module with Phase-1 and Phase-2 awareness yields steady gains, showing that each phase further strengthens summarization and planning. Upgrading the verifier from Base Act. to Phase-1 and Phase-2 Act. brings additional, though smaller, improvements, indicating that a stronger policy head can better exploit enhanced awareness. The monotonically increasing scores from "Base Aw.+Base Act." to "Phase 2 Aw.+Phase 2 Act." thus confirm that PROGRA 's staged training provides complementary improvements to both awareness and decision-making for multi-turn function calling.

4.7 Cross-Domain Performance

To further evaluate PROGRA's cross-domain generalization, we conduct out-of-domain experiments in Table 3 across τ-bench and τ^2-bench subdomains. Following Section 4.4, methods are trained on the Airline and Retail subdomains of τ-bench and evaluated on the Telecom subset of τ^2-bench. Across both Qwen2.5-7B and xLAM-2-8B, PROGRA achieves the best overall and Telecom performance, showing that progress-aware training simultaneously strengthens in-domain task execution and out-of-domain generalization. In contrast, SFT and various GRPO variants yield only modest or inconsistent gains and often fail to improve, or even degrade, performance on the unseen Telecom domain, suggesting a tendency to overfit to the training subdomains. These results indicate that explicitly modeling task progress enables the policy to extract more domain-invariant decision patterns, leading to more robust behavior under cross-domain distribution shifts.

Model	Method	Overall	Airline	Retail
xlam2-3B	Base Model	25.45	24.00	26.90
	Progra + GPT-3.5	30.09	30.00	32.17
	Progra + GPT-4o	31.52	30.00	33.04
xlam2-8B	Base Model	46.70	35.20	58.20
	Progra + GPT-3.5	50.84	37.33	64.35
	Progra + GPT-4o	51.85	38.00	65.70

Table 4: Ablation on the strength of the progress-awareness generator in the PAG stage.

4.8 Ablation on Supervisor of LLM

To assess whether the improvements of PROGRA are driven primarily by the use of a strong progress-awareness generator, we conduct an ablation study on the supervisor model used in the PAG stage. Specifically, we replace GPT-4o with the weaker GPT-3.5-turbo for progress-awareness generation, while keeping all other training configurations unchanged. The results are reported in Table 4.

Across both xLAM-2-3B and xLAM-2-8B, substituting GPT-4o with GPT-3.5-turbo leads to a moderate performance drop relative to the full PROGRA setting, but still preserves clear gains over the corresponding base models. This trend suggests that stronger generators can improve the quality of synthesized progress-awareness signals and further enhance downstream performance. At the same time, the fact that PROGRA remains substantially better than the base model even with a weaker generator indicates that its improvements cannot be explained solely by strong external supervision. Instead, the gains mainly stem from the progress-aware training framework itself.

4.9 Case Study

In order to better demonstrate the enhancements of PROGRA in multi-turn function calling tasks, we conduct a case study on a trajectory from the airline task in τ-bench using the xLam-2-3B model, and provide the corresponding quantitative analysis results for all trajectories in the airline task. In this trajectory, the PROGRA-trained model proactively summarized conversations, capturing both the user request and earlier context (e.g., flight numbers) to support accurate modifications. In contrast, the untrained model failed to summarize and relied only on the current step's flight number, leading to repetitive function calls.

Building on this case study, we further evaluate the two methods on 50 τ-Bench conversations. As summarized in Table 5, PROGRA consistently achieves higher accuracy with fewer steps than Direct inference, confirming its effectiveness in multi-turn conversation scenarios. The case study shows that PROGRA improves multi-turn function calling by leveraging progress-aware summarization. Quantitative results in Table 5 further highlight its advantage, with a higher success rate and fewer steps in successful traces. Overall, these findings show that PROGRA outperforms the base model in both efficiency and reliability, making it better suited for multi-turn function calling tasks.

Base Model	PROGRA
Process:	**Process:**
• Assumes details without verification	• Protocoling: request user ID & reservation ID PA
• Skips user/booking checks	• Factual retrieval: `get_user_details`, `get_reservation_details`
• Pure direct reasoning	• Reasoning with summary and plan PA
	• Early stop: no change needed PA

Figure 5: Case Study. A piece of case on τ-bench with the following user request: *"Hi! I'd like to make some changes to my upcoming trip from New York to Chicago... I want to upgrade to economy class, add 3 checked bags, and change the passenger name to myself, Omar Rossi."* Steps marked with PA are explicitly guided by *progress awareness*.

Method	Acc.	Avg. Steps	Risk
Direct	0.26	27.08	High
PROGRA	0.40	23.30	Low

Table 5: Quantitative comparison. 'Acc' denotes the average success rate over all traces, 'Avg Step' refers to the average number of steps in successful traces, and 'Risk' represents the assessment of trace failure risk.

5 Conclusion

In this paper, we propose PROGRA, a novel training framework that introduces task progress awareness into the multi-turn function calling. We first identify the performance bottleneck of multi-turn function calling as the LLM's lack of overall task progress awareness. Based on this, we design a two-phase training process. In Phase 1, an automated data synthesis process is used to enhance the LLM's task awareness. In Phase 2, progress awareness is integrated into end-to-end multi-turn reinforcement learning, significantly improving the LLM's performance in multi-turn function calling. The performance improvements achieved exceed those of existing training strategies across two public datasets and three backbone LLMs.

Limitations

Computational cost PROGRA introduces extra computation and token usage by decoupling awareness generation from action generation at each step. This effectively adds additional forward passes during training (and potentially inference), leading to higher latency and resource consumption compared to standard RL fine-tuning. Such overhead may be restrictive for large models or latency-sensitive scenarios unless further efficiency techniques (e.g., distillation or lightweight awareness heads) are applied.

Awareness representation and evaluation The framework assumes that task progress can be faithfully captured by textual summaries and plans produced by LLMs, and the evaluation focuses primarily on downstream task success and schema-level correctness. Progress awareness itself may still contain hallucinations, biases, or omissions that are not explicitly measured, and we do not systematically analyze failure modes where awareness is partially incorrect yet still yields valid function calls. Extending PROGRA with richer diagnostics (e.g., human judgments, safety constraints, or uncertainty signals) is left for future work.

Limited analysis of failure modes While quantitative results show consistent gains, the work provides only a small number of qualitative case studies and does not systematically categorize failure patterns of PROGRA versus baselines (e.g., over-planning, unnecessary tool use, or brittle awareness summaries). A deeper error analysis could reveal where progress awareness is most beneficial and where it still fails, guiding future refinements of the framework.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback and suggestions. This work was supported in part by the National Key R&D Program of China (2024YFC3505402) and the National Natural Science Foundation of China (62322603, U2244217, 624B2096, 72542012, 72595872).

References

Emre Can Acikgoz, Jeremiah Greer, Akul Datta, Ze Yang, William Zeng, Oussama Elachqar, Emmanouil Koukoumidis, Dilek Hakkani-Tür, and Gokhan Tur. 2025. Can a single model master both multi-turn conversations and tool use? coalm: A unified conversational agentic language model. *Preprint*, arXiv:2502.08820.

Tamer Alkhouli, Katerina Margatina, James Gung, Raphael Shu, Claudia Zaghi, Monica Sunkara, and Yi Zhang. 2025. Confetti: Conversational function-calling evaluation through turn-level interactions. *Preprint*, arXiv:2506.01859.

Jiajun Chai, Guojun Yin, Zekun Xu, Chuhuai Yue, Yi Jia, Siyu Xia, Xiaohan Wang, Jiwen Jiang, Xiaoguang Li, Chengqi Dong, Hang He, and Wei Lin. 2025. Rlfactory: A plug-and-play reinforcement learning post-training framework for llm multi-turn tool-use. *Preprint*, arXiv:2509.06980.

Mingyang Chen, Haoze Sun, Tianpeng Li, Fan Yang, Hao Liang, Keer Lu, Bin Cui, Wentao Zhang, Zenan Zhou, and Weipeng Chen. 2025a. Facilitating multi-turn function calling for llms via compositional instruction tuning. *Preprint*, arXiv:2410.12952.

Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z. Pan, Wen Zhang, Huajun Chen, Fan Yang, Zenan Zhou, and Weipeng Chen. 2025b. Research: Learning to reason with search for llms via reinforcement learning. *Preprint*, arXiv:2503.19470.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. Retool: Reinforcement learning for strategic tool use in llms. *Preprint*, arXiv:2504.11536.

Chendi Ge, Xin Wang, Zeyang Zhang, Hong Chen, Jiapei Fan, Longtao Huang, Hui Xue, and Wenwu Zhu. 2025. Dynamic mixture of curriculum lora experts for continual multimodal instruction tuning. *arXiv preprint arXiv:2506.11672*.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, and 168 others. 2026. Glm-5: from vibe coding to agentic engineering. *Preprint*, arXiv:2602.15763.

Jianghao Lin, Yuanyuan Shi, Xin Peng, Renjie Ding, Hairui Wang, Yuxuan Peng, Bizhe Bai, Weixi Song, Fengshuo Bai, Huacan Chai, and 1 others. 2025a. Toolprm: Fine-grained inference scaling of structured outputs for function calling. *arXiv preprint arXiv:2510.14703*.

Jianghao Lin, Xinyuan Wang, Xinyi Dai, Menghui Zhu, Bo Chen, Ruiming Tang, Yong Yu, and Weinan Zhang. 2025b. Masstool: A multi-task search-based tool retrieval framework for large language models. *arXiv preprint arXiv:2507.00487*.

Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang,

Chenchen Zhang, Ge Zhang, Jiebin Zhang, and 1 others. 2025a. A comprehensive survey on long context language modeling. *arXiv preprint arXiv:2503.17407*.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, and 1 others. 2024. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025b. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2025. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *Preprint*, arXiv:2408.04682.

Youssef Mroueh, Nicolas Dupuis, Brian Belgodere, Apoorva Nitsure, Mattia Rigotti, Kristjan Greenewald, Jiri Navratil, Jerret Ross, and Jesus Rios. 2025. Revisiting group relative policy optimization: Insights into on-policy and off-policy training. *arXiv preprint arXiv:2505.22257*.

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2024. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Advances in Neural Information Processing Systems*.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. 2025a. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *Preprint*, arXiv:2504.03601.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. 2025b. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *CoRR*, abs/2504.03601.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2020. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. *Preprint*, arXiv:1909.05855.

Abraham Sanders, Tomek Strzalkowski, Mei Si, Albert Chang, Deepanshu Dey, Jonas Braasch, and Dakuo Wang. 2022. Towards a progression-aware autonomous dialogue agent. *Preprint*, arXiv:2205.03692.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024a. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

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

Wentao Shi, Mengqi Yuan, Junkang Wu, Qifan Wang, and Fuli Feng. 2024. Direct multi-turn preference optimization for language agents. *arXiv preprint arXiv:2406.14868*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652. Curran Associates, Inc.

Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. 2025. Agentic reasoning and tool integration for llms via reinforcement learning. *Preprint*, arXiv:2505.01441.

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

Hanlin Wang, Chak Tou Leong, Jiashuo Wang, Jian Wang, and Wenjie Li. 2025a. Spa-rl: Reinforcing llm agents via stepwise progress attribution. *arXiv preprint arXiv:2505.20732*.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, and 1 others. 2025b. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*.

Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. 2025a. Agentnet: Decentralized evolutionary coordination for llm-based multi-agent systems, 2025. *URL https://arxiv. org/abs/2504.00587*.

Yingxuan Yang, Huacan Chai, Yuanyi Song, Siyuan Qi, Muning Wen, Ning Li, Junwei Liao, Haoyi Hu, Jianghao Lin, Gaowei Chang, and 1 others. 2025b. A survey of ai agent protocols. *arXiv preprint arXiv:2504.16736*.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

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

Fan Yin, Zifeng Wang, I-Hung Hsu, Jun Yan, Ke Jiang, Yanfei Chen, Jindong Gu, Long T. Le, Kai-Wei Chang, Chen-Yu Lee, Hamid Palangi, and Tomas Pfister. 2025. Magnet: Multi-turn tool-use data synthesis and distillation via graph translation. *Preprint*, arXiv:2503.07826.

Siliang Zeng, Quan Wei, William Brown, Oana Frunza, Yuriy Nevmyvaka, and Mingyi Hong. 2025. Reinforcing multi-turn reasoning in llm agents via turn-level credit assignment. *arXiv preprint arXiv:2505.11821*.

Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. 2025. Nemotron-research-tool-n1: Exploring tool-using language models with reinforced reasoning. *Preprint*, arXiv:2505.00024.

Congmin Zheng, Jiachen Zhu, Jianghao Lin, Xinyi Dai, Yong Yu, Weinan Zhang, and Mengyue Yang. 2025a. Cold: Counterfactually-guided length debiasing for process reward models. *arXiv preprint arXiv:2507.15698*.

Congming Zheng, Jiachen Zhu, Zhuoying Ou, Yuxiang Chen, Kangning Zhang, Rong Shan, Zeyu Zheng, Mengyue Yang, Jianghao Lin, Yong Yu, and 1 others. 2025b. A survey of process reward models: From outcome signals to process supervisions for large language models. *arXiv preprint arXiv:2510.08049*.

Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, Congming Zheng, Jiachen Zhu, Zeyu Zheng, Zhuosheng Zhang, Xingyu Lou, Changwang Zhang, Zhihui Fu, Jun Wang, Weiwen Liu, and 2 others. 2026. Externalization in llm agents: A unified review of memory, skills, protocols and harness engineering. *Preprint*, arXiv:2604.08224.

Jiachen Zhu, Congmin Zheng, Jianghao Lin, Kounianhua Du, Ying Wen, Yong Yu, Jun Wang, and Weinan Zhang. 2025. Retrieval-augmented process reward model for generalizable mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8453–8468, Vienna, Austria. Association for Computational Linguistics.

Backbone	Method	Score	Improv.(%)	Time (GPU*s)	Token
xLAM-2-3B	Base Model	25.45	–	–	–
	MT-GRPO	26.35	3.53	1903.20	1040.67
	PROGRA	31.52	23.85	2900.16	2243.32

Table 6: Deployment Performance Analysis Efficiency comparison on τ-bench. "Score" refers to the average score on τ-bench. "Improv." indicates the performance improvement relative to the base model, "Time" represents the training time per step (in GPU*seconds), and "Token" indicates the number of tokens required per step of training.

A Deployment Analysis

In multi-turn function call tasks, in addition to accuracy and generalization ability, training and deployment efficiency are also critical factors in determining whether a training paradigm can be applied to real-world systems. This section investigates whether PROGRA, with the introduction of a task progress awareness mechanism, can maintain competitive training and inference/deployment efficiency compared to existing methods. This question is especially important for industrial applications, where performance improvements are often accompanied by strict constraints on efficiency metrics such as response latency and resource consumption. The section will analyze the time and token overheads in the training and deployment stages.

During the training process, each step in PROGRA is decomposed into two sub-steps: (i) explicitly generating task progress-aware content, and (ii) issuing function calls. This design inevitably introduces additional time and token overheads. Table 6 presents the GPU computation time and the number of tokens generated per step during the training of xLAM-2-3B on the τ-bench dataset.

As shown in the table, PROGRA increases the training time per step by approximately 50% compared to MT-GRPO, but on τ-bench, PROGRA achieves a performance improvement of about 20 percentage points over MT-GRPO. Therefore, from an overall cost-benefit perspective, this trade-off is still reasonable and cost-effective. Additionally, MT-GRPO results in only a 3.53% improvement compared to the baseline model, while PROGRA yields a significant 23.85% improvement, demonstrating a much larger relative gain. This indicates that the additional time and token cost invested in the training phase results in substantial performance returns.

Furthermore, PROGRA 's two-stage training strat-

egy reaches the performance goal more quickly than traditional single-stage end-to-end reinforcement learning. This is partly because the warm-up fine-tuning phase (Stage 1) enhances the target model's initialization strategy, allowing the subsequent reinforcement learning phase to start from a higher baseline, thus reducing the noise exploration in the reinforcement learning phase. Additionally, task progress-aware reinforcement learning can more clearly align the reward signal with global task progress, avoiding common issues in traditional multi-turn reinforcement learning, such as reward sparsity and unstable value estimation when dealing with lengthy contexts, thereby improving training convergence efficiency.

In the testing phase, PROGRA, like all baseline methods, uses BFCL with the default prompts provided by τ-bench. The model only outputs explicit reasoning content and function calls, without explicitly generating lengthy progress-aware paragraphs; the progress-aware information is implicitly embedded in the reasoning output. This design ensures a fair comparison without introducing additional inference overhead while retaining feasibility for actual deployment, avoiding artificially inflating evaluation performance by increasing the computational load in the testing phase.

B Data synthesis

We provide a detailed account of the data synthesis methodology employed in this study. The process is adapted from the APIGEN-MT framework (Prabhakar et al., 2025a), which comprises a two-phase approach: (1) task configuration and groundtruth generation, and (2) human-agent-environment interaction trajectory collection.

In the initial phase, an objective, its corresponding function call, and the resulting output are generated for each data instance. This generation is guided by provided APIs, a set of predefined rules, and a specified domain. Subsequently, each generated instance undergoes a rigorous verification protocol that assesses its syntactic formatting and operational executability. A majority voting mechanism, arbitrated by an LLM, is then employed to ascertain the correctness of the data. Should an instance fail these verification checks or the majority vote, the model will analyze the failure cases, formulate a corrective strategy, and reiterate the generation-verification cycle until the data successfully meets all validation criteria.

Following successful validation in the first phase, the synthesized data is deployed into a simulated human-agent-environment to produce an interaction trajectory via rollout. This resultant trajectory is then compared against the ground truth trajectory established in the initial phase. Only those instances where the two trajectories exhibit exact correspondence are retained for the final synthetic dataset.

The data synthesis pipeline in this study is adapted from the APIGEN-MT framework, with significant modifications to four core stages: initial configuration generation, query generation, action generation, and correctness verification. The following sections detail these customized processes.

Initial Config Generation: The process commences with the generation of an initial_config, which serves as the foundational state for each conversation trajectory. To ensure contextual relevance, this stage is seeded with data from a predefined JSON dataset. Specifically, a subset of data entries is first filtered based on a specified involved_class criterion. From this filtered subset, a random selection of existing initial_config instances is sampled. These sampled configurations act as reference templates or exemplars. A generative model then synthesizes a new, distinct initial_config that adheres to the structural and schematic patterns of the references. This targeted sampling strategy ensures that the generated initial states are not only well-formed but also thematically aligned with the desired domain, thereby enhancing the relevance of the subsequent conversation synthesis.

Query Generation: The conversation synthesis process begins each interaction cycle with a query generation step. In this phase, the system leverages the complete preceding context as input, namely comprising the initial_config and the historical conversation trajectory. Conditioned on this context, a large language model (LLM) is invoked via a single API call to produce a new query. The design of this step ensures that each generated query is a logical and progressive continuation of the interaction. Queries are formulated to be explicit requests for environmental modification that depend on the outcomes of prior turns, thus establishing a coherent and causally linked chain of reasoning throughout the conversation.

Action Generation: Following each query generation turn, the pipeline proceeds to synthesize a corresponding action. In this phase, the query formulated in the immediately preceding turn is

supplied to an LLM prompted specifically for action synthesis. To mitigate generation errors and enhance the reliability of the output, a distributed majority voting mechanism is employed. This is implemented by triggering multiple, parallelized API calls to the LLM to produce a set of candidate actions. These candidates are then subjected to a consensus protocol, where votes for each unique candidate are aggregated. The action that surpasses a predefined frequency threshold is selected as the definitive result. A critical constraint is that all generated actions must conform to a strict, machine-parsable format, such as a list of function calls (e.g., [func_name1(arg1=value1,...), func_name2(...)]). If no candidate action achieves the required consensus threshold, the generation process for the current trajectory is aborted to prevent the inclusion of low-confidence or ambiguous data.

Correctness Verification: The data verification process is a rigorous, automated pipeline designed to ensure the functional and semantic correctness of multi-turn AI agent trajectories. For each trajectory, a hermetic execution environment is instantiated from an initial configuration to guarantee reproducible validation. The pipeline then employs a two-tiered verification protocol for each turn: first, it deterministically checks if an agent's action causes an observable state transition in the environment. If no change is detected (a common result for read-only operations), a secondary check uses a majority-voting consensus from a Large Language Model (LLM) to adjudicate the action's semantic validity based on the user's query. Crucially, the process adheres to a strict sequential policy, terminating immediately upon the first failed turn to ensure that only fully coherent and causally valid interaction sequences are retained in the final dataset.

This data synthesis methodology offers several distinct advantages that collectively enhance the quality, relevance, and reliability of the generated dataset. By seeding the process with domain-specific exemplars, the pipeline ensures that all synthesized trajectories are thematically relevant and contextually grounded from their inception. The iterative, context-aware generation of queries and actions promotes the creation of coherent, multi-turn conversations that exhibit logical and causal consistency.

Furthermore, the integration of a consensus-based validation mechanism for action generation significantly improves the accuracy and reliability of the output by filtering out erroneous or low-confidence predictions. Crucially, the final execution-based verification stage provides a rigorous guarantee of functional fidelity, ensuring that every data point corresponds to a verifiable and correct interaction within the target environment. This multi-layered approach to generation and validation yields a high-quality dataset that is not only syntactically sound and semantically coherent but also empirically validated for functional correctness.

C Prompts

The prompts we use when synthesizing data include the prompt for the -Initial Config Generation(Fig. 6), Task Generation(Fig. 7), Action Generation(Fig. 8).

Initial Config Generation Prompt

You are an environment generator. Your task is as follows: you will be provided with several examples of environment initial configurations, along with some notes. Based on these, generate **ONE** new environment initial configuration. The content is entirely up to you, but the overall format should be inspired by the examples (not an exact copy). Your output must be a valid, directly parseable JSON string. You **MUST NOT** include any extra text, explanations, or JSON hints outside of the JSON itself. You only need to output ONE new configuration.

Figure 6: Initial Config Generation Prompt.

Task Generation Prompt

You are a helpful assistant, which will generate a trajectory containing Queries and Actions. You will be provided with a basic description of an existing scenario and an introduction to the tools available in that scenario.
Your task is to output a reasonable and clear query based on the result of the previous message. If the previous message represents an action or the initialization of a trajectory (no previous message), you must output a Query that can be solved by calling the tools within the environment. Your output query

must involve a change to the environment state (e.g., adding or moving files, modifying content and so on).
There are some important notes you should follow.
1. The query should be progressive, where each query depends on the successful answer of the previous one, forming a realistic problem-solving process.
2. The query should not be too complex, it is better to cost 2-4 function calls to complete the query.
3. Please ensure that the Query you output is very clear and explicit, and that it allows only one possible solution.
4. Your query should involve a change to the environment state, try not only to display information.
Here is the initial config: {initial_config}

Figure 7: Task Generation Prompt.

Action Generation Prompt

You are a helpful assistant, which will generate a trajectory containing Queries and Actions. You will be provided with a basic description of an existing scenario and an introduction to the tools available in that scenario.
Your task is to output a correct function call output based on the result of the previous message and query. Your output should include some function calls that strictly follow the required format. The functions you call must come from the provided tool descriptions. The function call should be the following format:
[func_name1(arg_name1=value1,...), func_name2(arg_name1=value1,...)...]

Figure 8: Action Generation Prompt.

Here are the prompts we used in Progra for generating progress awareness in PAG (Fig. 9), and Optimization in RAG-RL (Fig. 10)

Progress Awareness Generation Prompt

You are a help assistant responsible for summarizing important information based on the history of the conversation and providing a plan for invoking the correct function calls. Carefully analyze the current multi-turn conversation and generate a detailed summary (and helpful plan). **Do not directly output any function calls**, just output your summary in text format.
In your summary, ensure the following points are clearly addressed:
1. **User's Needs and Intent**: Please analyze the provided historical information and accurately identify the user's needs and goals. Summarize the content in a brief and clear manner to ensure that any subsequent work can fully understand the user's requirements and objectives based on your summary.
2. **Extracted Parameters**: Please list all relevant parameters mentioned or clarified in the conversation, ensuring complete understanding. Only include parameters that have been explicitly confirmed, and avoid making assumptions or guessing values for information that has not been clearly stated. If the historical information provided lacks any parameters necessary to meet the user's needs, confirm and point them out.
3. **Function Call History**: Carefully review previous function calls made, including whether they were successful or failed, and identify any potential issues. Ensure you clearly summarize the results of previous function calls without making assumptions about the context or next steps.
4. **Environment State Awareness**: Review and summarize the entire conversation so far, paying special attention to previous function calls and their results. In each turn of history, the environment sequentially executes the functions output in last turns. If a function fails, the subsequent ones won't be executed, but the successful ones will affect the environment state. In the current turn, you must reason the current environment state based on the executed functions and their results in last turns. You need to fully understand the current environmental state in order to make the correct choices. If you cannot understand it, you cannot make unreasonable assumptions about the state.

5. **Future Planning**: Based on the current and prior conversation, propose a clear action plan. Avoid speculation; instead, provide a plan that logically follows from the facts at hand. Do not directly output any function calls; instead output your plan in text format.
6. **Relevant Important Context**: Include any other context from the conversation that could aid in ensuring the next function call is accurate and appropriate. This can include non-explicit user preferences, hints from previous statements, or details that, while not directly related to the goal, may still influence the next action.
Ensure that your summary is detailed and comprehensive, clarifying any ambiguities and accurately reflecting the history of the conversation to allow the next response or action to be as accurate and informed as possible.
Here is the history of prior conversations and current user query.
History of Conversation
{current_history_str}
Current User Query
{query}
Please make a summary based on the above conversation and system prompt.

Figure 9: Progress Awareness Generation Prompt.

Optimization Prompt

You are an expert in invoking functions. You will be given a summary of a conversation between a user and an AI assistant, as well as a set of available functions. Based on the summary and the questions posed by the user, you must select and call functions to achieve the user's goal. You must return the response in the following format:
⟨ summary ⟩
Summarise the conversation so far for subsequent reasoning
⟨ /summary ⟩
⟨ think ⟩
Express your thought process. In this section, you should carefully consider the question's intent, your reasoning for selecting the appropriate function to call, and how you plan to fill in the function's parameters and arguments. Reflect on the user's needs to ensure the correct choice is made.
⟨ /think ⟩
⟨ answer ⟩
Output function call in the following format:
[func_name1(arg_name1=value1,arg_name2=value2...),
func_name2(arg_name1=value1,arg_name2=value2...)...]
⟨ /answer ⟩
- You MUST NOT include any text other than the ⟨summary⟩, ⟨ think⟩ and ⟨answer⟩ sections. Follow the provided format strictly.
- **You MUST call only the functions provided in the document**. The function names and parameters 'func_name1', 'func_name2' and 'params_name1' shown in the ⟨answer⟩⟨/answer⟩ section are placeholders used only to demonstrate the required output format. They are not actual function names to be called. Please select and use only from the real functions provided as possible. - When filling in the parameters in the function call, replace params_value1 and params_value2 with the correct values as per the document. Ensure the parameters are properly named and formatted. - If additional parameters are needed to call the function correctly, output "I need more information." in the ⟨answer⟩⟨/answer⟩ section with a text format rather than a list-style answer. - Please strictly adhere to the list of functions provided to complete the task. These functionns may be similar to common commands you are familiar with (such as cd, touch, etc.), but you are strictly prohibited from using any functions that are not on the provided list to complete the task. - If you find that none of the functions in the provided list can directly accomplish the task, you must not attempt to use other available functions in a roundabout way to force the completion of the task. In that case, simply output the following in the ⟨answer⟩⟨/answer⟩ tags: there are no appropriate functions. - In each round, the environment sequentially executes the functions output by you in last turns. If a

function fails, the subsequent ones won't be executed, but the successful ones will affect the environment state. In the next round, you must infer the current environment state based on the executed functions and their results in last turns, ensuring that failed calls are not executed again.

Figure 10: Optimization Prompt in PAG-RL