PROACTOR: TIMING-AWARE REINFORCEMENT LEARNING FOR PROACTIVE TASK SCHEDULING AGENTS

Lei Ding[1] **Bin He**[2] **Chenguang Wang**[1] **Yang Liu**[1]
[1]University of California, Santa Cruz [2]Zillow Group
{lding25, chenguangwang, yangliu}@ucsc.edu binh@zillowgroup.com

ABSTRACT

Proactive task-oriented agents must autonomously anticipate user needs, identify actionable opportunities, and trigger software actions at appropriate moments—fundamentally shifting from reactive systems that await explicit instructions. However, existing approaches lack generalizable end-to-end solutions for measuring and optimizing such anticipatory behaviors.

This paper introduces ProActor, a unified framework for conversational task scheduling that integrates: (1) a domain-agnostic automated annotation methodology that enables scalable proactiveness reinforcement learning (RL) by generating full opportunity time windows instead of rigid point labels, (2) systematic proactiveness metrics capturing both timing quality and reference action alignment, and (3) RL optimization using GRPO with various reward designs. Our insight is that RULER-based rewards with proactiveness rubrics are crucial for improving timing quality, and that proactiveness optimization enabled by stage-aware composite rewards is key to balancing timing quality and reference action alignment.

Furthermore, we introduce ART-F, an adaptive RL framework that combines request-adaptive inference clusters with asynchronous training for better GPU utilization, enabling LoRA training of 4-bit Qwen2.5-14B-ProActor-Q4 models on 4×H200 and 8×H100 GPUs with substantial speedups. Experiments on two newly auto-annotated datasets demonstrate significant improvements in proactive timing while maintaining action consistency comparable to state-of-the-art (SOTA) baselines. Ablations validate the effectiveness of distinct composite reward variations.

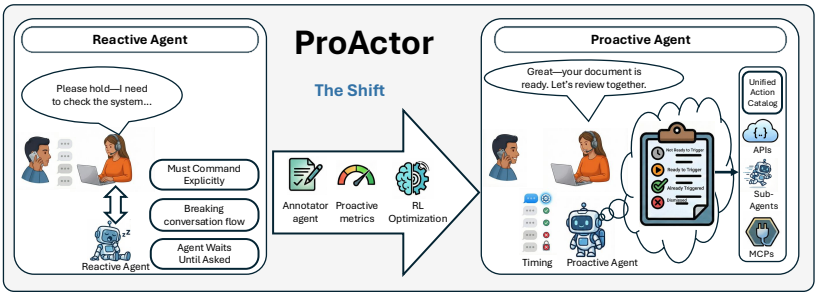


Figure 1: Our ProActor shifts conversational agents from reactive execution to proactive task scheduling. An automated annotation pipeline provides reference actions, proactiveness metrics quantify timing and prediction action alignment, and RL optimizes timing-aware policies over a unified action catalog—enabling agents to act proactively without disrupting conversation flow.

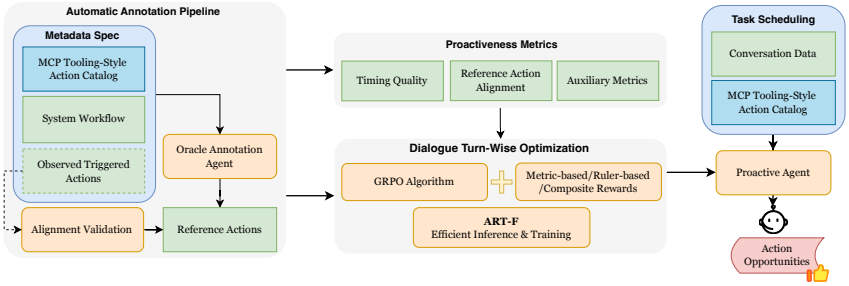


Figure 2: Overview of ProActor for end-to-end proactiveness optimization. ProActor integrates (1) an automated annotation pipeline that generates high-quality MCP-style reference actions, (2) a suite of proactiveness metrics that quantify action quality and timing, and (3) ART-F, an efficient RL framework enabling scalable training in GPU-constrained environments. Proactiveness optimization in conversational task scheduling is formalized as a dialogue turn–level RL problem.

1 INTRODUCTION

As AI assistants evolve, a paradigm shift is emerging—from **reactive** systems that wait for explicit instructions to **anticipatory** agents (Lu et al., 2025; Wu et al., 2025b). Empowered by large language models (LLMs), proactive agents can anticipate user needs, surface relevant actions, and guide workflows before being asked. A case in point is an agent that continuously monitors conversations between professionals and clients, identifies emerging opportunities, and intervenes seamlessly when action is appropriate (Figure 1). Such ambient agents point toward a future in which AI agents are deeply embedded in daily interactions, supporting decision-making across customer service, enterprise automation, and real-time assistance.

Recent work has explored proactive behavior optimization via stronger prompting (Deng et al., 2023), improved context engineering (Yoon et al., 2025), supervised fine-tuning (SFT) (Dong et al., 2025), and reward-driven optimization (Lu et al., 2025; Wu et al., 2025b). However, they remain insufficient in conversational task scheduling. First, proactive behavior allows multiple valid timing choices rather than a single correct answer (Sodhi et al., 2023), making SFT unsuitable because reproducing exact reference timings penalizes valid alternatives and obscures underlying timing principles (Sodhi et al., 2023; Kim et al., 2025). Moreover, the lack of automated pipelines for proactive action annotation (Wu et al., 2023; Sodhi et al., 2023) further amplifies challenges of proactive optimization. To fill these gaps, we introduce a domain-agnostic automated annotation pipeline that constructs a unified action catalog and leverages an oracle LLM annotator to generate proactive action candidates, referred to as **reference actions**. Our pipeline also supports evaluating the quality of reference actions with the ready status by the alignment with **action observations** actually triggered, and preserving only high-quality references.

Second, existing methods (Wu et al., 2023; Deng et al., 2023; Zhou et al., 2024a) struggle to quantitatively capture timing-sensitive factors—such as readiness and termination—crucial for conversational task scheduling. By contrast, we treat reference actions as proactiveness anchors and define **a suite of quantitative metrics** to evaluate timing quality and action consistency, complemented by preference-based RULER rewards (OpenPipe, 2025) for ambiguity-aware proactiveness evaluation. Guided by these signals—particularly stage-aware composite rewards—we apply Group Relative Policy Optimization (GRPO) (Shao et al., 2024) at the dialogue turn level, formalizing proactiveness optimization as a principled trade-off between better proactive timing and consistency with reference actions.

Last but not least, RL post-training for LLMs is highly resource-intensive, requiring large-scale rollout generation, interaction among policy, reference, and reward models, and tightly coordinated distributed training across GPUs (Ouyang et al., 2022; Shao et al., 2024). Motivated by evidence that LoRA-based RL can match full-parameter tuning (Schulman & Lab, 2025), we apply **Low-Rank Adaptation (LoRA)** (Hu et al., 2022) to **quantized LLMs** and introduce Adaptive Resource Training Framework (**ART-F**), an efficient end-to-end RL framework that integrates a request-adaptive inference cluster with DDP-based training to maximize GPU utilization on single-node, multi-GPU systems. Using ART-F, we train Qwen2.5-14B-ProActor-Q4, a 4-bit LoRA-tuned model, achieving

4–8× training speedups on 4×H200 and 8×H100 GPUs while delivering substantial gains in proactive timing and maintaining SOTA action consistency on two newly annotated datasets, ABCD+ and Home Loan. Ablation studies further confirm that the stage-aware composite reward is critical for achieving our optimization goal. We will open-source ART-F and annotation pipelines in the future.

In summary, our main contributions are:

- We introduce **an automated domain-agnostic annotation pipeline** to extract a unified action catalog and generate high-quality reference actions, delivering new datasets: ABCD+ and Home Loan.
- We propose **a comprehensive suite of proactiveness metrics** that quantitatively capture timing quality, alignment between predicted and reference actions, enabling systematic evaluation and optimization of task-scheduling agents.
- We formulate proactiveness optimization as **a trade-off between timing accuracy and reference alignment**, and introduce a dialogue turn–level RL-based proposal that leverages metric-based, RULER-based, and composite rewards to optimize proactive decision-making.
- We introduce **ART-F**, an efficient RL post-training framework for quantized LLMs, and demonstrate its effectiveness by training **Qwen2.5-14B-ProActor-Q4** that attains 4–8× faster training while achieving **strong gains in proactive timing** without sacrificing action consistency.

2 Related Works

Proactive Agents and Task Scheduling Recent advances in proactive agents (Lu et al., 2025; Wu et al., 2025b; Deng et al., 2024) fundamentally reshape intelligent assistant design (Abbas et al., 2025; Liu et al., 2025c) and their application to task scheduling (Xu et al., 2025; Yoon et al., 2025; Yu et al., 2025; Liu et al., 2024). To optimize proactive behaviors, researchers invested efforts in generalizing tool abstractions, standardizing action representations (Rastogi et al., 2020; Liu et al., 2025b; Qin et al., 2024) and constructing data pipelines (Zhang et al., 2025; Wei et al., 2025) in specific domains. Despite progress, existing evaluations (Liu et al., 2025a; Lu et al., 2025; AlShikh et al., 2025) largely overlook time-sensitive proactive behaviors (Kim et al., 2025): they treat proactive timing as a single-answer problem, penalizing predictions that deviate from labels even when earlier timings represent valid proactive behavior (Sodhi et al., 2023). This limitation motivates our RL-based approach: we generate *reference action ranges* and use turn-level optimization to explore proactive timing rather than replicating exact one reference point.

RL Optimization & Efficient Training RL has been applied to conversational task scheduling, from dialogue policy learning (Peng et al., 2018; Li et al., 2021; Zhao et al., 2024) to multi-step tool optimization (Qin et al., 2024; Zheng et al., 2024; Zhang et al., 2024). Despite explicit timing models (Chen et al., 2025; Zhou et al., 2024a), time-sensitive proactivity in conversational settings remains underexplored. We address this gap by formulating proactive timing as a step-wise optimization problem with composite rewards to balance early action with correctness. Furthermore, RL training incurs substantial computational cost and infrastructure complexity (Lattimore et al., 2013; Espeholt et al., 2018; Krishnan et al., 2019). While distributed RL frameworks improve scalability (Wang et al., 2025b; Bartoldson et al., 2025), LoRA-based (Hu et al., 2022) quantized RL on single-node GPUs often suffers from rollout–training imbalance and under-utilization (Dettmers et al., 2023; Hu et al., 2022). ART-F targets this by enabling efficient, collocated RL training through a lightweight request-adaptive inference and asynchronous payload distribution in GPU-constrained environments.

3 Methodology

3.1 Domain-Agnostic Automated Annotation Pipeline for Reference Actions

Given RL needs large volumes of reference action ranges, which is infeasible for humans to label, it is crucial to develop an automated pipeline that can produce these annotations at scale. Following Chen et al. (2021), we introduce a unified metadata schema that normalizes heterogeneous action interfaces into a standardized JSON representation, capturing action ontology, type signatures, and

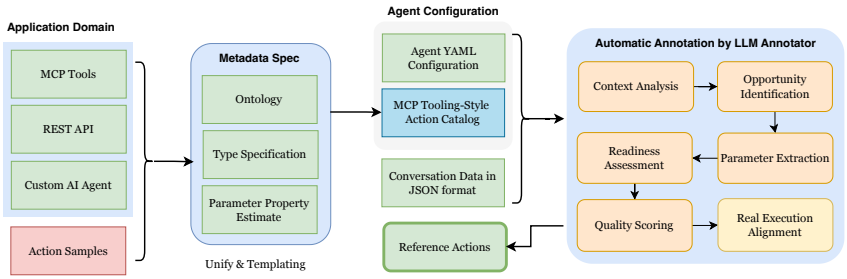


Figure 3: Automated annotation pipeline: Heterogeneous tools are unified through a domain-agnostic, configuration-driven *Catalog Generator*, while an *LLM annotator* with a global dialogue view produces high-quality reference actions at each dialogue turn.

parameter properties. The schema supports diverse automation backends and is rendered into a unified action catalog (Appendix D.1) using Jinja2 (Pallets, 2025).

Given a dialogue and the generated tool catalog, we initialize an oracle LLM annotator (Appendix D.2) with access to the full conversation—including future turns—and instruct it to identify proactive opportunities at each dialogue turn by following the predefined procedure in Figure 3. This deliberate design endows the annotator with *action-forecasting* capability, enabling it to identify actions that *would have been* beneficial at each moment rather than merely retrospectively explaining outcomes. Importantly, the entire pipeline is **fully automated and model-agnostic**: adapting to new domains or LLMs requires only configuration changes.

Reference Actions: Guidance Over Ground Truth. We term annotation outputs as **reference actions** because they represent *one or more valid timing choices given each dialogue turn context*—not ground truth. The oracle annotator identifies *suitable* actions, not necessarily *optimal* ones; more aggressive proactive timing could also be valid. This multiplicity motivates RL over SFT: we optimize policies exploring proactive opportunities rather than replicating exact timings. For datasets with action observations, we further validate the alignment between reference actions labeled with ready status and the corresponding observed actions with triggered status (Appendix D.3).

3.2 Proactiveness Metrics

Proactive task scheduling requires agents to decide *what*, *how*, and *when* to act. Accordingly, we define a set of metrics that evaluate action quality along two dimensions: *action consistency* (AC, Max AC, and Difference) and *proactive timing* (PT, FTR, and RAR). For convenience, predictions with status READY_TO_TRIGGER or TRIGGERED are treated as *ready actions*. Our setting permits partial parameter specification and explicit readiness tracking, unlike standard tooling calls that assume complete parameter mappings (Patil et al., 2025). Formally, given a unified action catalog $\mathcal{A}$, we evaluate agent behavior at each dialogue turn t by comparing agent prediction actions $\hat{A}_t$ with reference actions A_t^{ref}.

Action Consistency (AC) measures the average alignment level between $\hat{A}_t$ and A_t^{ref}:

$$\text{AC}(\hat{A}_t, A_t^{\text{ref}}) = \frac{1}{|\hat{A}_t|} \sum_{\hat{a}_i \in \hat{A}_t} \max_{\substack{a_j \in A_t^{\text{ref}} \\ a_j.\text{name}=\hat{a}_i.\text{name}}} \frac{C(R(\hat{a}_i), R(a_j)) + C(O(\hat{a}_i), O(a_j))}{|R(a_j)| + |O(a_j)|} \tag{1}$$

where $R(\cdot)$, $O(\cdot)$ denote required and optional parameters, and $C(\cdot,\cdot)$ is the parameter match score, i.e., the fraction of parameters that match between prediction and reference.

Maximum AC (Max AC) captures the best alignment level between $\hat{A}_t$ and A_t^{ref}:

$$\text{Max AC}(\hat{A}_t, A_t^{\text{ref}}) = \max_{\hat{a} \in \hat{A}_t} \text{AC}(\{\hat{a}\}, A_t^{\text{ref}}) \tag{2}$$

Consistency Difference (Difference) measures prediction reliability as the relative gap between AC and Max AC. Given statistics of AC (A, δ_A) and Max AC (M, δ_M) over multiple runs, we define

$$\mu = \frac{M - A}{A}, \delta \approx \sqrt{\left(\frac{\delta_M}{A}\right)^2 + \left(\frac{M \cdot \delta_A}{A^2}\right)^2} \tag{3}$$

A high value indicates inconsistent predictions, while a low value reflects more stable behaviors.

Reference Range [1] characterizes how ready actions are distributed across dialogue turns. For action $a \in \mathcal{A}$, its reference range is defined as

$$\mathcal{R}_a = \{\, t \mid a \in_{\text{name}} A_t^{\text{ref}} \wedge a \text{ is ready at turn } t \,\}, \tag{4}$$

and the total reference range is $\mathcal{R} = \bigcup_{a \in \mathcal{A}} \mathcal{R}_a$.

Proactive Timing (PT) rewards predicted actions that occur *no later than* the reference-ready window:

$$\text{PT}(t, \hat{A}_t, \mathcal{R}) = \frac{\sum_{\hat{a} \in \hat{A}_t} \mathbb{I}[\exists \tau \in \mathcal{R}_{\hat{a}} : \tau \geq t]}{|\hat{A}_t|} \tag{5}$$

Fault Trigger Rate (FTR) penalizes predicted ready actions that fall outside the reference coverage. Denoting the set of predicted ready actions at dialogue turn t as

$$\hat{A}_t^{\text{ready}} = \{\, \hat{a} \in \hat{A}_t \wedge \hat{a} \text{ is a ready action} \,\},$$

FTR is defined as

$$\text{FTR}(t, \hat{A}_t^{\text{ready}}, \mathcal{R}) = \frac{|\{\hat{a} \in \hat{A}_t^{\text{ready}} \wedge \hat{a} \notin \mathcal{R}\}|}{|\hat{A}_t^{\text{ready}}|} \tag{6}$$

Ready Action Rate (RAR) measures the proportion of predictions marked ready at turn t:

$$\text{RAR}(t, \hat{A}_t) = \frac{|\{\hat{a} \in \hat{A}_t \wedge \hat{a} \text{ is a ready action}\}|}{|\hat{A}_t|} \tag{7}$$

Low RAR indicates conservatism; high RAR with high FTR suggests over-eager triggering.

More metrics refer to Appendix C.

3.3 Reinforcement Learning Optimization

Task Definition Given a reference-ready dataset and a unified action catalog, our goal is to optimize a policy model π that, at dialogue turn t, produces a candidate action set $\hat{A}_t$. The policy is trained to maximize accumulated rewards by aligning $\hat{A}_t$ with the corresponding reference actions A_t^{ref}, measured both action consistency and proactive timing. This formulation raises questions: how can reward design and reward granularity effectively translate proactiveness metrics and evaluation rubrics into learning signals?

Reward Granularity Decision In general, **trajectory-level rewards**, which average action feedback over an entire dialogue, are ill-suited for task scheduling: long dialogues ($\geq$ 23 turns) worsen credit assignment, early actions create cascading effects that are hard to disentangle, and sparse end-of-trajectory signals provide limited learning guidance. These observations are consistent with our empirical findings (Section M.1), motivating the adoption of **turn-level reward modeling** that evaluates action quality per dialogue turn.

[1] For $A_t \in \{\hat{A}_t, A_t^{\text{ref}}\}$, we denote $a \in_{\text{name}} A_t$, if there exists $a' \in A_t$ such that $a'.\text{name} = a.\text{name}$; correspondingly, $a \notin_{\text{name}} A_t$ denotes that no such a' exists.

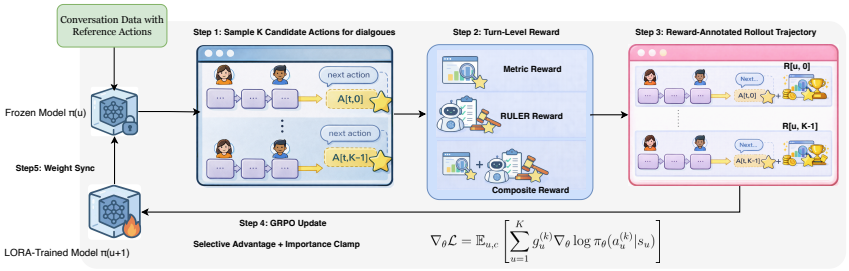


Figure 4: Turn-level GRPO optimization: we sample K action candidates from the policy model $\pi_\theta(u)$ for each dialogue turn, score action trajectories with the chosen reward, update $\pi_\theta(u)$ to $\pi_\theta(u+1)$ using GRPO with reward-weighted gradients.

Turn-Level Optimization At optimization step[2] u, we rollout K times for each dialogue turn and update the policy model π_θ using turn-level GRPO, as illustrated in Figure 4:

$$\nabla_\theta \mathcal{L} = \mathbb{E}_{u,c}\left[\sum_{k=1}^{K} g_u^{(k)} \nabla_\theta \log \pi_\theta(a_u^{(k)}|s_u)\right] \tag{8}$$

where c is the episode context, $s_u, a_u^{(k)}$ are the state/action. $g_u^{(k)}$ is defined as the turn-level advantage $A_u^{(k)}$ scaled by an importance ratio $\bar{r}_u^{(k)}$ (set 10 based on Appendix M.2.1 and H.4), with PPO-style clipping applied to limit gradient magnitude.

We further design **several turn-level rewards**, beginning with *single-objective rewards* that only optimize action consistency or timing.

Metric-based rewards use AC and Max AC to compute the alignment value between $\hat{A}_t$ and A_t^{ref} as reward signals, referred to as *RAC* and *Max RAC*, respectively, which are intentionally designed to improve action consistency.

RULER-based rewards ART's default RULER (OpenPipe, 2025; Hilton et al., 2025), referred to as **General RULER**, applies a generic evaluation rubric to assess how well predicted actions follow prompt instructions. We also introduce **Custom RULER** to **explicitly evaluate proactive behaviors** and expect it to benefit the optimization of proactive timing behaviors.

Next, we propose *composite rewards* that combine action consistency and timing with fixed coefficients:

Weighted Metric Reward Since PT and FTR are relatively sparse timing signals and AC/Max AC yields dense action consistency signals, we define a simple combination reward:

$$R_{\text{wM}}^{\mathcal{F}}(u) = \mathcal{F}(u) + w_1 \cdot \text{PTR}(u) - w_2 \cdot \text{FTR}(u) \tag{9}$$

where $\mathcal{F} \in \{\text{AC}, \text{MaxAC}\}, w_1 = 0.05, w_2 = 0.01$.

Furthermore, we introduce *stage-aware rewards* to adjust factor weights as the training progresses.

Adaptive Metric Reward Given a total of U training steps, this design tries to encourage broad exploration early (Max RAC + a small bonus for PT) when $u < \frac{U}{3}$ as *the exploration phase*, then transitions to balanced optimization (RAC with the increased bonus for PT and penalty for FTR) when $\frac{U}{3} \leq u < \frac{2U}{3}$ as *the balanced phase*, and finally becomes conservative (RAC with the final penalty for FTR) late in conversations to avoid noise when $u \geq \frac{2U}{3}$ as *the conservative phase*.

$$R_{\text{adM}}(u) = \begin{cases} w_1 \cdot \text{MaxRAC}(u) + (1 - w_1) \cdot \text{PTR}(u) & u < \frac{U}{3} \\ w_2 \cdot \text{RAC}(u) + w_3 \cdot \text{PTR}(u) - (1 - w_2 - w_3) \cdot \text{FTR}(u) & \frac{U}{3} \leq u < \frac{2U}{3} \\ w_4 \cdot \text{RAC}(u) - (1 - w_4) \cdot \text{FTR}(u) & \text{otherwise} \end{cases} \tag{10}$$

[2]In this paper, t denotes dialogue turns; u indexes RL steps

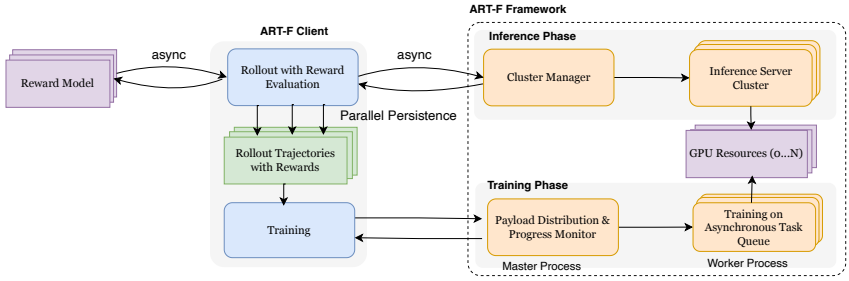


Figure 5: ART-F framework: An efficient collocated RL training system that combines an adaptive inference cluster with asynchronous, distributed training to maximize GPU utilization.

where $w_1 = 0.8$, $w_2 = 0.6$, $w_3 = 0.3$, and $w_4 = 0.6$ are fixed coefficients, selected empirically.

Adaptive RULER Reward As RULER-based rewards tend to gradually overemphasize timing and Metric-based rewards promote action consistency but weaken timing, we design adaptive formulations balancing these two aspects:

$$R_{\text{adR}}(u) = (1 - \lambda_u) R_{\text{metric}}(u) + \lambda_u R_{\text{RULER}}(u) \tag{11}$$

where λ_u increases from 0 to $\lambda_{\max} = 0.3$, progressively emphasizing timing quality until the upper limit. More auxiliary metrics are in Appendix H.

3.4 Efficient Training Framework: ART-F

To address ART (Hilton et al., 2025)'s inefficiency under LoRA-based RL training, we introduce ART-F (Figure 5) that extends ART to enable an efficient collocated RL training (Wu et al., 2025a; Wang et al., 2025a) through improved resource orchestration (Appendix F and G).

Adaptive Inference Cluster ART-F dynamically instantiates multiple vLLM (Kwon et al., 2023) servers per GPU, each reserving a fixed memory fraction, thereby increasing parallel request capacity without exceeding memory limits. A centralized load-balancing cluster manager continuously monitors server health, throughput, and latency, exposing real-time routing information to rollout clients, enabling informed scheduling decisions—such as selecting available servers or early terminating rollouts under saturation—substantially improving inference throughput and GPU utilization.

Asynchronous Payload-Distribution Training ART-F implements a custom asynchronous payload-distribution training framework with a master–worker architecture. The master process coordinates training by partitioning rollout trajectory groups, distributing them to worker processes, and tracking execution progress via asynchronous queues. ART-F supports *symmetric (replicated) and asymmetric (partitioned) DDP (PyTorch Team, 2025; Unsloth AI, 2024) modes* (Appendix F.2), enabling trade-offs between stability and data efficiency, while step-level synchronization ensures consistent gradient updates. This design mitigates rollout–training imbalance, supports dynamic batch sizing, and enables stable, high-throughput RL optimization on a single multi-GPU node.

4 Experiments

4.1 Datasets

With action observations: ABCD+ extends the ABCD dataset (Chen et al., 2021) with proactive annotations in customer-support scenarios, representing settings *with* historical action execution logs. It contains 7,042 dialogues (5,647/703/692 train/dev/test), 114,978 annotated actions, and an average dialogue length of 21.2 ± 7.2 turns.

Without action observations: Home Loan is constructed from mortgage consultation transcripts between loan officers and clients (proprietary dataset), representing domains *without* observed soft-

	PRI ↑	Consistency			Timing		
		AC ↑	Max AC ↑	Difference ↓	Proactive Timing ↑	Fault Trigger Rate ↓	Ready Action Rate ↑
ABCD+							
GPT-5.1 Non-Reasoning	0.5104	0.318±0.005	0.706±0.012	0.717±0.013	0.2023±0.0019	0.0460±0.0009	0.419±0.010
GPT-5.1 Reasoning	0.6003	0.429±0.006	0.789±0.002	0.839±0.026	0.1643±0.0018	**0.0165±0.0002**	0.214±0.003
GPT-5.1 Reasoning + ASG	0.5547	0.420±0.004	0.733±0.007	0.712±0.063	0.1874±0.0022	0.0647±0.0019	0.281±0.003
Gemini-2.5-flash Non-Reasoning (4)	**0.6251**	0.417±0.004	**0.834±0.001**	1.000±0.019	0.2133±0.0030	0.0399±0.0006	0.288±0.003
Gemini-2.5-flash Reasoning	0.6216	**0.430±0.001**	0.813±0.002	0.891±0.006	0.1782±0.0011	**0.0245±0.0006**	0.252±0.001
Gemini-2.5-flash Reasoning + ASG	0.5257	0.384±0.001	0.737±0.001	0.919±0.006	0.1715±0.0015	0.0286±0.0005	0.241±0.004
Claude-4 Non-Reasoning	0.6216	0.427±0.001	0.831±0.001	0.946±0.005	0.2119±0.0013	0.0526±0.0012	0.297±0.001
Claude-4 Reasoning (3)	**0.6318**	0.421±0.001	0.749±0.004	0.779±0.010	0.2136±0.0023	0.0482±0.0023	0.314±0.003
Claude-4 Reasoning + ASG	0.6031	0.403±0.005	**0.852±0.004**	1.114±0.028	0.2269±0.0022	0.0634±0.0032	0.360±0.002
Qwen2.5-14B-Instruct Non-Reasoning	0.2996	0.295±0.004	0.564±0.002	0.914±0.004	0.2237±0.0012	0.0773±0.0010	0.515±0.002
Qwen2.5-14B-Instruct Reasoning	0.6246	0.423±0.005	0.449±0.006	0.062±0.019	0.1842±0.0012	0.0307±0.0008	0.279±0.003
Qwen2.5-14B-Instruct Reasoning + ASG	0.4331	0.316±0.004	0.385±0.004	0.218±0.020	0.1918±0.0043	0.0432±0.0011	0.359±0.004
Qwen2.5-14B-ProActor-Q4 + Custom RULER (1)	**0.7293**	0.426±0.015	0.484±0.019	0.136±0.048	**0.2347±0.0201**	0.0708±0.0078	**0.546±0.036**
Qwen2.5-14B-ProActor-Q4 + Adaptive RULER (2)	**0.6842**	**0.431±0.022**	0.586±0.044	0.320±0.123	**0.2515±0.0272**	0.1089±0.0083	**0.521±0.052**
Home Loan							
GPT-5.1 Non-Reasoning	0.5047	0.272±0.001	0.467±0.003	0.717±0.013	0.0462±0.0013	0.0049±0.0002	0.116±0.003
GPT-5.1 Reasoning	0.5047	0.363±0.007	**0.579±0.004**	0.595±0.033	0.0186±0.0026	**0.0003±0.0003**	0.031±0.002
GPT-5.1 Reasoning + ASG	0.5010	0.281±0.007	0.481±0.013	0.712±0.063	0.0374±0.0061	0.0034±0.0004	0.105±0.010
Gemini-2.5-flash Non-Reasoning	0.6165	0.349±0.004	**0.688±0.004**	0.972±0.024	0.0632±0.0011	0.0072±0.0005	0.173±0.005
Gemini-2.5-flash Reasoning (1)	**0.7303**	0.345±0.004	0.527±0.006	0.528±0.024	0.0757±0.0011	**0.0001±0.0001**	0.241±0.002
Gemini-2.5-flash Reasoning + ASG	0.6067	0.283±0.007	0.479±0.002	0.693±0.042	0.0764±0.0039	0.0063±0.0018	0.251±0.012
Claude-4 Non-Reasoning	0.5416	0.224±0.003	0.347±0.004	0.549±0.024	**0.0811±0.0029**	0.0118±0.0008	**0.332±0.002**
Claude-4 Reasoning (3)	**0.7039**	0.332±0.006	0.472±0.003	0.422±0.028	0.0742±0.0027	0.0063±0.0016	0.261±0.003
Claude-4 Reasoning + ASG (2)	**0.7262**	**0.375±0.002**	0.607±0.006	0.619±0.022	0.0760±0.0031	0.0127±0.0013	0.307±0.005
Qwen2.5-14B-Instruct Non-Reasoning	0.4288	0.253±0.002	0.383±0.003	0.514±0.017	0.0120±0.0006	0.0003±0.0004	0.054±0.005
Qwen2.5-14B-Instruct Reasoning	0.4012	0.217±0.005	0.239±0.009	0.101±0.049	0.0037±0.0010	0.0006±0.0001	0.032±0.003
Qwen2.5-14B-Instruct Reasoning + ASG	0.3223	0.113±0.011	0.144±0.006	0.274±0.133	0.0184±0.0083	0.0033±0.0022	0.090±0.014
Qwen2.5-14B-ProActor-Q4 + Custom RULER	0.5603	0.206±0.024	0.234±0.021	0.137±0.161	**0.0846±0.0095**	0.0355±0.0079	**0.465±0.074**
Qwen2.5-14B-ProActor-Q4 + Adaptive RULER (4)	**0.6232**	**0.395±0.029**	0.466±0.038	0.180±0.129	0.0501±0.0055	0.0131±0.0013	0.156±0.009

Table 1: Model performance comparison on test set: Baselines are averaged over 3 runs; ProActor-Q4 results are aggregated over the last $N{=}4$ checkpoints. Adaptive RULER uses Max RAC with $\lambda_{\max} = 0.3$. Superscripts $^{(1)}$–$^{(4)}$ indicate the top-1 to top-4 methods ranked by PRI per group.

ware action triggers. It includes 968 dialogues (774/97/97 train/dev/test), 30,610 annotated actions, and longer conversations averaging 47.4 ± 1.1 turns. More details are in Appendix I.

4.2 SETUP

Baselines With the unified action catalog and specific dataset, we design: **(1) Non-Reasoning**, where the model directly identifies action opportunities, status and timing; **(2) Reasoning** requires the model to explicitly reason (via `<think>`) before predicting action opportunities, status, and timing; **(3) Reasoning + ASG** further maintains an action-state graph (ASG) to track predicted actions across turns and conditions the reasoning on this evolving action state. We test baselines on GPT-5.1, Gemini-2.5-flash, Claude-sonnet-4[3], Qwen2.5-14B-Instruct. More details are in Appendix E.

RL Training We train Qwen2.5-14B-ProActor-Q4, a 4-bit quantized Qwen2.5-14B-Instruct model optimized via LoRA[4], using our ART-F client (Appendix G.2). Training is performed on 4×H200 (ABCD+) and 8×H100 (Home Loan) with a maximum 9,216-token context. We enable symmetric DDP training (Appendix F.2) for 2 epochs, equivalent to 8 (ABCD+) and 16 (Home Loan) epochs under non-symmetric partitioning. End-to-end training completes in 3.5–5.7 days on ABCD+ and 1.5–2.15 days on Home Loan. Full settings are in Appendix J.

Proactiveness Ranking Index (PRI) To measure whether models strike a better proactiveness balance, we introduce PRI (Appendix K.4), a composite ranking score that aggregates Consistency Index (CI) and Timing Index (TI) using harmonic mean: $\text{PRI} = \frac{2\times\text{CI}\times\text{TI}}{\text{CI}+\text{TI}}$ where CI combines action consistency metrics (AC, Max AC, Difference) and TI combines timing metrics (PT, FTR, RAR), all min-max normalized within comparison groups. Rankings are reported separately for the main results (Table 1) and ablations (Table 2).

#Dialogs Train/Test	Reward Type	PRI ↑	AC ↑		Max AC ↑		Proactive Timing (PT) ↑		Fault Trigger Rate (FTR) ↓		Ready Action Rate (RAR) ↑	
			Statistics	Δ	Statistics	Δ	Statistics	Δ	Statistics	Δ	Statistics	Δ
ABCD+												
100/50	RAC	0.2596	0.3239±0.01606	-0.1528	0.3881±0.02590	-0.1348	0.1223±0.02364	-0.1030	**0.0640±0.01960**	0.9636	0.3002±0.08088	0.2770
100/50	Max RAC	0.2264	0.3263±0.02468	-0.4740	**0.4384±0.02264**	-0.1587	0.1643±0.00207	-0.0616	0.0886±0.01212	2.65	0.3862±0.01111	0.2262
100/50	General Ruler †	**0.4918**	0.3494±0.02714	-0.2750	0.3978±0.02984	-0.2436	**0.2324±0.01984**	0.9056	0.1019±0.01910	3.7188	**0.5945±0.05405**	1.4867
100/50	Custom Ruler ⋆	**0.6140**	**0.3850±0.02544**	-0.1995	0.4203±0.03200	-0.2476	0.2315±0.02393	0.6755	0.1068±0.02327	4.8358	0.5707±0.05717	1.2846
5647/692	RAC	0.4352	0.3368±0.02401	-0.1478	0.3857±0.01949	-0.1327	0.1799±0.00452	0.0128	0.0621±0.00765	0.1760	0.4148±0.02354	0.1818
5647/692	Max RAC	0.3737	0.3745±0.03439	-0.0926	0.5014±0.06809	0.0492	0.1172±0.03605	-0.0351	**0.0365±0.01013**	0.0919	0.2426±0.06160	-0.0008
5647/692	Custom Ruler ⋆	**0.7217**	0.4257±0.01469	-0.4791	0.4837±0.01848	-0.4731	0.2347±0.02014	0.56561	0.0708±0.00784	2.4654	0.5456±0.04169	1.8482
5647/692	Weighted Metric (Max RAC)	0.2169	0.3068±0.02606	-0.2975	0.4399±0.05253	0.0011	0.1465±0.03091	0.0338	0.0700±0.01253	1.5204	0.3769±0.05893	0.3607
5647/692	Adaptive Metric	0.5159	0.3537±0.02264	-0.1950	0.3922±0.03489	-0.2524	0.2268±0.04591	0.2966	0.1137±0.01925	2.5661	**0.6332±0.11113**	1.0600
5647/692	Adaptive RULER ($\lambda_{\max} = 0.3$, Max RAC) †	**0.6026**	**0.4314±0.02240**	-0.05045	**0.5861±0.04335**	0.1150	**0.2515±0.02722**	0.4410	0.1089±0.00832	1.9003	0.5212±0.05153	0.5818
Home Loan												
774/97	RAC	0.5668	**0.4701±0.03285**	0.8483	**0.5195±0.03016**	0.8462	0.0264±0.00590	-0.2502	0.0041±0.00245	-0.7494	0.0612±0.01856	-0.5365
774/97	Max RAC	0.4311	0.3894±0.01327	0.4891	0.4780±0.02065	0.6927	0.0093±0.00378	-0.8214	**0.0015±0.00072**	-0.9409	0.0177±0.00787	-0.9178
774/97	Custom Ruler	0.4220	0.2057±0.02409	-0.3760	0.2338±0.02114	-0.3329	**0.0846±0.00951**	2.5245	0.0355±0.00785	5.2040	**0.4653±0.07412**	4.8508
774/97	Weighted Metric (RAC)	0.5376	0.4359±0.01376	0.6695	0.4805±0.02399	0.7160	0.0205±0.00349	-0.2859	0.0020±0.00066	-0.8762	0.0481±0.00722	-0.6039
774/97	Weighted Metric (Max RAC)	0.4809	0.4133±0.02947	0.6461	0.5346±0.03620	0.9601	0.0270±0.00730	-0.4212	0.0049±0.00090	-0.4268	0.0506±0.02000	-0.7042
774/97	Adaptive Metric	0.5742	0.4677±0.01760	0.8090	0.5184±0.02184	0.8390	0.0282±0.00552	-0.3170	0.0042±0.00121	-0.7683	0.0653±0.01628	-0.5919
774/97	Adaptive RULER ($\lambda_{\max} = 0.3$, RAC) ⋆	**0.6154**	0.4173±0.00768	0.5111	0.4403±0.01019	0.4510	0.0397±0.00528	0.4382	0.0071±0.00070	-0.1682	0.1231±0.01985	0.4329
774/97	Adaptive RULER ($\lambda_{\max} = 0.3$, Max RAC) †	**0.5734**	0.3950±0.02936	0.5366	0.4658±0.03821	0.6764	0.0501±0.00547	1.0451	0.0131±0.00131	1.5299	0.1561±0.00930	0.5190

Table 2: Performance using different rewards on test set. Δ denotes the relative change of each metric after RL training w.r.t. its pre-training value. ⋆ and † indicate top-1 and top-2 methods by PRI per group.

4.3 Main Results

Baseline Analysis *(1) Reasoning improves consistency but harms timing.* Reasoning reduces Consistency Difference (–19.9/–28.6%[5] on average for non-Qwen baselines, and over –80% for Qwen), while making AC/Max AC vary across models. However, reasoning degrades timing by ≈10%, with only isolated gains for Claude-4 on ABCD+ and Gemini-2.5-flash on Home Loan (Table 6).

(2) ASG introduces varied timing gains at a substantial consistency cost. ASG adds varied PT gains (–3.8% to 14.1% on ABCD+ and 0.9% to 101.1% on Home Loan) but degrades AC/Max AC (2% to 22.6%, except Claude-4 on Home Loan) and increases inconsistency (10% to 40%) for most models, showing that structure alone can't hit a better consistency and timing balance (Table 7).

(3) No baseline resolves the consistency-timing tradeoff. Across both datasets, baselines achieving high Max AC ($\geq$0.8) incur large Consistency Difference (0.75 to 1.11) and exhibit conservative behavior with low PT ($\leq$0.23) and RAR ($\leq$0.36); while a few baselines approach ProActor's PT, none maintain Difference below 0.2, confirming our claim (Appendix K.5).

ProActor-Q4 *RL enables 4-bit ProActor-Q4 models to achieve a better trade-off between action consistency and proactive timing*, as shown in Table 1. We attribute it to reward designs that optimize action consistency via RAC/Max RAC, while leveraging explicit rubrics for proactiveness evaluation in RULER-based rewards. Two ProActor-Q4 variants exhibit complementary strengths along different dimensions:

Custom RULER achieves the best timing overall, with PT = 0.2347/0.0846 and RAR = 0.546/0.465, higher than the strongest baselines (typically PT$\leq$0.2023/0.0811, RAR$\leq$0.288/0.251), while keeping Consistency Difference at 0.136/0.137, comparable to or lower than most SOTA baselines.

Adaptive RULER delivers the most balanced improvement, reaching the highest AC (0.431/0.395) among all methods and strong PT (0.2515/0.0501), while maintaining a lower Difference than high-Max AC baselines (e.g., GPT-5.1, Gemini, Claude often exceed 0.7 to 1.1 Difference).

Explicit reasoning induces hesitation, while ProActor breaks this paralysis by using RL to internalize timing intuition. ASG's accumulation of past opportunities creates a reasoning burden that distracts static policies. RL offers a feasible path to effectively harness the rich ASG memory rather than becoming overwhelmed by it.

4.4 Ablation Study

Reward Variations **RULER-based** rewards favor smaller-scale settings, where data patterns are simple and limited. In contrast, **Adaptive RULER** better captures the proactiveness trade-off at larger scales, and even some reward variants outperform on individual metrics. In addition, Adaptive

[3] Abbreviated to Claude-4 in the remaining sections

[4] rank 8, $\alpha = 16$, applied to attention and MLP projections.

[5] t_1/t_2 indicates the dataset-specific metric value, with t_1 for ABCD+ and t_2 for Home Loan

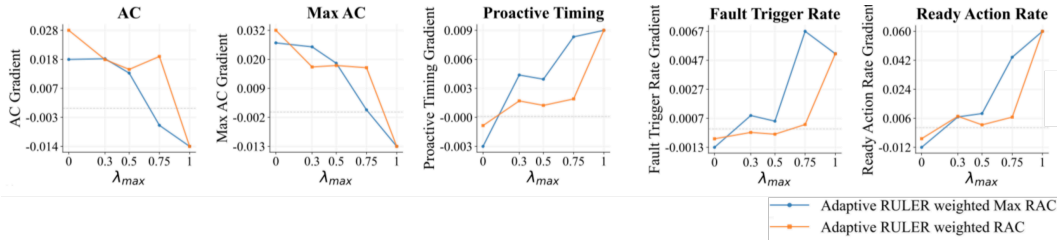


Figure 6: Effect of λ_{max} on Adaptive RULER weighted by Max RAC (blue) and RAC (orange), showing how **Average Gradient** of six main metrics change with $\lambda_{max} \in \{0, 0.3, 0.5, 0.75, 1.0\}$.

RULER combined with RAC typically achieves a higher PRI than its Max RAC counterpart, because AC aims at optimizing average rather than the best-case action alignment.

Single-objective Rewards at ABCD+ 100/50 RULER-based rewards dominate PRI (0.49 to 0.61) over RAC and Max RAC (0.23 to 0.26), driven by substantially higher PT ($\approx$0.23 vs. 0.12 to 0.16) and RAR ($\approx$0.57 to 0.59 vs. 0.30 to 0.39). While Max RAC rewards maintain a strong Max AC (0.4384), they exhibit conservative behavior with smaller PT ($\leq$0.165) and RAR ($\leq$0.4). As General RULER underperforms Custom RULER in PRI and other metrics are close, we omit it in later experiments.

Reward Variations at ABCD+ 5647/692 & Home Loan 774/97 Adaptive RULER attains top-2 PRI (0.60–0.62) on both dataset, reflecting a balanced trade-off. In contrast, some single-objective rewards (e.g. RAC on Home Loan) achieve strong AC/Max AC (up to 0.52) but exhibit low PT and RAR (PT $<$ 0.03), showing a greater imbalance.

$\lambda_{\max}$ **Effect.** For Adaptive RULER rewards weighted by RAC and Max RAC, $\lambda_{\max}$ functions as a speed-control knob for proactive timing: those with higher $\lambda_{\max}$ tend to optimize timing metrics more aggressively. We enumerate $\lambda_{\max} \in \{0, 0.3, 0.5, 0.75, 1.0\}$ on the Home Loan 774/97 setting, with 5 runs per configuration, and compute **the average metric gradients on the testing set for each neighborhood training step** in each run. Illustrated by Figure 6, the average gradients of PT, FTR, and RAR scale **approximately proportionally with** $\lambda_{\max}$, whereas those of AC and Max AC exhibit **an inverse trend** (Appendix L).

5 Conclusion

We present ProActor, a unified framework for training timing-aware proactive task-scheduling agents. Our approach integrates: (1) a domain-agnostic annotation pipeline generating *reference actions* with multiple valid timing choices, (2) systematic proactiveness metrics with rubric evaluations, and (3) turn-level GRPO with several reward variations, especially composite rewards to controllably balance action consistency and timing. The key insight is that turn-level rewards with an explicit proactiveness evaluation rubric are essential for learning timing-aware policies.

Experiments on ABCD+ and Home Loan demonstrate significant improvement in proactive timing while maintaining action consistency. Our ART-F enables this at scale with up to 4–8$\times$ speedup. Domain adaptation requiring only configuration changes validates ProActor's generalizability for real-world task automation.

Acknowledgments

We thank Andy Martin for valuable feedback and guidance throughout this work, and Jyoti Prakash Maheswari, Taleb Zeghmi, Supriya Anand, and Amir Reza Rahmani for helpful technical discussions. This work was conducted during an internship at Zillow Group.

References

Adnan Abbas, Caleb Wohn, Arnav Jagtap, Eugenia H Rho, Young-Ho Kim, and Sang Won Lee. " having lunch now": Understanding how users engage with a proactive agent for daily planning

and self-reflection. *arXiv preprint arXiv:2509.24073*, 2025.

Waseem AlShikh, Muayad Sayed Ali, Brian Kennedy, and Dmytro Mozolevskyi. Towards outcome-oriented, task-agnostic evaluation of ai agents. *arXiv preprint arXiv:2511.08242*, 2025.

Viktor Andersen. awesome-serverless-gpu: Curated list of services/platforms for serverless gpu and ai inference. `https://github.com/viktorfa/awesome-serverless-gpu`, 2024. Accessed: 2025-11-19.

Anthropic. Model context protocol: A standard for tool integration. `https://github.com/anthropics/model-context-protocol`, 2024.

Brian R Bartoldson, Siddarth Venkatraman, James Diffenderfer, Moksh Jain, Tal Ben-Nun, Seanie Lee, Minsu Kim, Johan Obando-Ceron, Yoshua Bengio, and Bhavya Kailkhura. Trajectory balance with asynchrony: Decoupling exploration and learning for fast, scalable llm post-training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Derek Chen, Howard Chen, Yi Yang, Alexander Lin, and Zhou Yu. Action-based conversations dataset: A corpus for building more in-depth task-oriented dialogue systems. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3002–3017, 2021.

Tuochao Chen, Nicholas Scott Batchelder, Alisa Liu, Noah A Smith, and Shyamnath Gollakota. Llamapie: Proactive in-ear conversation assistants. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 13801–13824, 2025.

Databricks, Inc. Databricks. `https://www.databricks.com/`, 2025. Accessed: 2025-11-18.

Yang Deng, Lizi Liao, Liang Chen, Hongru Wang, Wenqiang Lei, and Tat-Seng Chua. Prompting and evaluating large language models for proactive dialogues: Clarification, target-guided, and non-collaboration. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10602–10621, 2023.

Yang Deng, Lizi Liao, Zhonghua Zheng, Grace Hui Yang, and Tat-Seng Chua. Towards human-centered proactive conversational agents. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 807–818, 2024.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2305.14314.

Wenjie Dong, Sirong Chen, and Yan Yang. Protod: Proactive task-oriented dialogue system based on large language model. In *Proceedings of the 31st International Conference on Computational Linguistics (COLING)*, pp. 9147–9164, 2025.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Daniel Han, Michael Han, and the Unsloth Team. Unsloth. `https://github.com/unslothai/unsloth`, 2023. Accessed: 2025-03.

Brad Hilton, Kyle Corbitt, David Corbitt, Saumya Gandhi, Angky William, Bohdan Kovalenskyi, and Andie Jones. Art: Agent reinforcement trainer. `https://github.com/openpipe/art`, 2025.

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 *ICLR*, 2022. arXiv:2106.09685.

Luo Ji, Gao Liu, Mingyang Yin, Hongxia Yang, and Jingren Zhou. Hierarchical reinforcement learning for temporal abstraction of listwise recommendation. *CoRR*, 2024.

Namyoung Kim, Kai Tzu-iunn Ong, Yeonjun Hwang, Minseok Kang, Iiseo Jihn, Gayoung Kim, Minju Kim, and Jinyoung Yeo. PRINCIPLES: Synthetic strategy memory for proactive dialogue agents. In *Findings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025. arXiv:2509.17459.

Srivatsan Krishnan, Max Lam, Sharad Chitlangia, Zishen Wan, Gabriel Barth-maron, Aleksandra Faust, and Vijay Janapa Reddi. Quarl: Quantization for fast and environmentally sustainable reinforcement learning. *Transactions on Machine Learning Research*, 2019.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Tor Lattimore, Marcus Hutter, and Peter Sunehag. The sample-complexity of general reinforcement learning. In *International Conference on Machine Learning*, pp. 28–36. PMLR, 2013.

Ziming Li, Xi Chen, et al. Toward conversational recommendation over multi-type dialog states. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

Tianjian Liu, Hongzheng Zhao, Yuheng Liu, Xingbo Wang, and Zhenhui Peng. Compeer: A generative conversational agent for proactive peer support. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2024.

Tianjian Liu, Fanqi Wan, Jiajian Guo, and Xiaojun Quan. Proactiveeval: A unified evaluation framework for proactive dialogue agents. *arXiv preprint arXiv:2508.20973*, 2025a.

Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, et al. Toolace: Winning the points of llm function calling. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Xingyu Bruce Liu, Shitao Fang, Weiyan Shi, Chien-Sheng Wu, Takeo Igarashi, and Xiang'Anthony' Chen. Proactive conversational agents with inner thoughts. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 1–19, 2025c.

Yaxi Lu, Shenzhi Yang, Cheng Qian, Guirong Chen, Qinyu Luo, Yesai Wu, Huadong Wang, Xin Cong, Zhong Zhang, Yankai Lin, et al. Proactive agent: Shifting llm agents from reactive responses to active assistance. In *International Conference on Learning Representations (ICLR)*, 2025.

OpenPipe. RULER: Relative universal LLM-elicited rewards. `https://openpipe.ai/blog/ruler-a-model-for-llm-generated-rewards`, Jul 2025. Accessed: 2025-11-18.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744, 2022. arXiv:2203.02155, InstructGPT.

Pallets. Jinja documentation: Templates and code generation. `https://jinja.palletsprojects.com/en/stable/templates`, 2025. Documentation describing Jinja's text templating and code generation capabilities.

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Karl Pearson. Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58:240–242, 1895.

Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, Kam-Fai Wong, and Sining Chen. Deep reinforcement learning for task-oriented dialogue systems. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

PyTorch Team. Distributed data parallel (ddp). https://pytorch.org/docs/stable/generated/torch.nn.parallel.DistributedDataParallel.html, 2025. Accessed: 2025-02-19.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations*, 2024. arXiv:2307.16789.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 8689–8696, 2020.

John Schulman and Thinking Machines Lab. Lora without regret. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250929. https://thinkingmachines.ai/blog/lora/.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, and et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. Introduces GRPO (Group Relative Policy Optimization) as a variant of PPO.

Paloma Sodhi, Felix Wu, Ethan R Elenberg, Kilian Q Weinberger, and Ryan McDonald. On the effectiveness of offline rl for dialogue response generation. In *International Conference on Machine Learning*, pp. 32088–32104. PMLR, 2023.

Unsloth AI. Multi-gpu training with unsloth (ddp). https://docs.unsloth.ai/basics/multi-gpu-training-with-unsloth/ddp, 2024. Accessed: 2025-03.

Jiali Wang, Yankui Wang, Mingcong Han, and Rong Chen. Colocating {ML} inference and training with fast {GPU} memory handover. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pp. 1657–1675, 2025a.

Zhixin Wang, Tianyi Zhou, Liming Liu, Ao Li, Jiarui Hu, Dian Yang, Siyuan Feng, and Yuan Cheng. Distflow: A fully distributed rl framework for scalable and efficient llm post-training. *arXiv preprint arXiv:2507.13833*, 2025b. URL https://arxiv.org/abs/2507.13833.

Fei Wei, Daoyuan Chen, Ce Wang, Yilun Huang, Yushuo Chen, Xuchen Pan, Yaliang Li, and Bolin Ding. Grounded in reality: Learning and deploying proactive LLMs from offline logs. *arXiv preprint arXiv:2510.25441*, 2025.

Weights & Biases. W&b weave documentation. Online; accessed 2025, 2025a. URL https://docs.wandb.ai/weave.

Weights & Biases. Weights & biases (wandb) website. https://www.wandb.com/, 2025b. Accessed: 2025-01.

Sven Wientjes and Clay B Holroyd. The successor representation subserves hierarchical abstraction for goal-directed behavior. *PLOS Computational Biology*, 20(2):e1011312, 2024.

Bo Wu, Sid Wang, Yunhao Tang, Jia Ding, Eryk Helenowski, Liang Tan, Tengyu Xu, Tushar Gowda, Zhengxing Chen, Chen Zhu, et al. Llamarl: A distributed asynchronous reinforcement learning framework for efficient large-scale llm training. *arXiv preprint arXiv:2505.24034*, 2025a.

Shirley Wu, Michel Galley, Baolin Peng, Hao Cheng, Gavin Li, Yao Dou, Weixin Cai, James Zou, Jure Leskovec, and Jianfeng Gao. Collabllm: From passive responders to active collaborators. In *International Conference on Machine Learning (ICML)*, 2025b.

Zeqiu Wu, Yushi Hu, Weijia Shi, Nouha Dziri, Alane Suhr, Prithviraj Ammanabrolu, Noah A Smith, Mari Ostendorf, and Hannaneh Hajishirzi. Fine-grained human feedback gives better rewards for language model training. *Advances in Neural Information Processing Systems*, 36:59008–59033, 2023.

Heng-Da Xu, Xian-Ling Mao, Fanshu Sun, Tian-Yi Che, Chun Xu, and Heyan Huang. Agenttod: A task-oriented dialogue agent with a flexible and adaptive api calling paradigm. *ACM Transactions on Information Systems*, 43(5):1–32, 2025.

Yejin Yoon, Yuri Son, Namyeong So, Minseo Kim, Minsoo Cho, Chanhee Park, Seungshin Lee, and Taeuk Kim. Beyond task-oriented and chitchat dialogues: Proactive and transition-aware conversational agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13291–13317, 2025.

ChengZhang Yu, YingRu He, Hongyan Cheng, Zhixing Liu, Dongxu Mu, Zhangrui Shen, Zhanpeng Jin, et al. From passive to proactive: A multi-agent system with dynamic task orchestration for intelligent medical pre-consultation. *arXiv preprint arXiv:2511.01445*, 2025.

Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, and Mani Parkhe. Mlflow: An open platform for the machine learning lifecycle. `https://mlflow.org/`, 2018. Accessed: 2025-01.

Yichi Zhang, Xin Luna Dong, Zhaojiang Lin, Andrea Madotto, Anuj Kumar, Babak Damavandi, Joyce Chai, and Seungwhan Moon. Proactive assistant dialogue generation from streaming egocentric videos. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 12055–12079, 2025.

Yifan Zhang et al. Progra: Progress-aware reinforcement learning for multi-turn tool use. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

Xueliang Zhao et al. Rescuing conversations from dead-ends: A reinforcement learning perspective. *Transactions of the Association for Computational Linguistics (TACL)*, 2024.

Zifan Zheng et al. Workbench: Benchmarking large language models for agentic workplace tasks. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

Helen Zhou, Audrey Huang, Kamyar Azizzadenesheli, David Childers, and Zachary Lipton. Timing as an action: Learning when to observe and act. In *International Conference on Artificial Intelligence and Statistics*, pp. 3979–3987. PMLR, 2024a.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Aviral Kumar, and Sergey Levine. Archer: Training language model agents via hierarchical multi-turn rl. In *Forty-first International Conference on Machine Learning*, 2024b.

A LIMITATIONS

Action Observation Scope. A fundamental limitation concerns the nature of observed triggers in our evaluation. Current datasets only capture actions that *actually occurred* in the system, but suitable proactive actions form a broader range—not limited to what happened in the observed trajectory. This is precisely why we term our annotations *reference actions* rather than ground truth, which represent one valid timing choice among potentially many. Our quality filtering aligns against these actual triggers, but this represents a subset of valid opportunities. Future work should expand human annotation efforts to capture *all* suitable trigger points, enabling more comprehensive evaluation of proactive exploration beyond observed behaviors.

Evaluation Scope. Our experiments focus on two English-language datasets: ABCD+ (7,042 dialogues) and Home Loan (968 dialogues). Generalization to other languages and domains remains to be validated.

Model and Training Constraints. Our RL experiments focus on 4-bit quantized Qwen2.5-14B-Instruct with LoRA adapters. Generalization to other model families, parameter scales, and quantization schemes warrants investigation. We limit training to dialogues with $\leq$50 turns; performance on longer conversations remains to be evaluated.

ART-F Extension Due to the time limit, we only support training on a single node with multiple GPUs. However, ML platforms (Databricks, Azure, Google Cloud, etc.) provide additional GPU capacity through multi-node serverless GPU clusters, and numerous community resources are also available (Andersen, 2024). Extending ART-F to operate across multiple nodes is therefore a natural next step and an important technical advancement toward the scalable, resource-adaptive RL training.

B ETHICS STATEMENT

This work utilizes two datasets: a previously published public corpus of customer support transcripts ABCD+ and a proprietary dataset of loan officer-borrower interactions. Due to the sensitivity of the financial-related data, neither the proprietary dataset nor the model fine-tuned by the proprietary data will be publicly released. To ensure data privacy and security, the model training and experimental analysis were conducted within a secure compute environment with strict access controls, consistent with our organization's compliance standards.

We emphasize that this work is strictly experimental. Any real-world deployment of the framework would necessitate rigorous fairness testing to mitigate potential allocative harms. LLM and AI agents that misjudge situations may trigger unwanted actions, and miscalibration remains possible in out-of-distribution scenarios. We recommend thorough human-in-the-loop assessment before any production use to ensure both robustness and operational safety.

C MORE EVALUATION METRICS

Besides metrics presented in Section 3.2, we include several auxiliary evaluation metrics that capture alternative aspects. Because of limited experimental bandwidth, these metrics are reported only for baseline evaluations and are omitted for RL-trained models.

Information Consistency (IC) In task-oriented conversations, questions are typically used to fill information gaps required to advance the task. To assess whether an agent can proactively pose appropriate questions that naturally align with human behavior, we evaluate its ability of raising appropriate questions that gather missing information in a manner consistent with the original conversations. Specifically, an appropriate question should satisfy two criteria: (1) it exhibits high semantic similarity to questions posed by human professionals, and (2) it corresponds to missing parameters in the action state graph. Formally, we define Information Consistency (IC) based on the average semantic similarity between predicted questions and the corresponding reference questions in the original dialogue turn, as follows:

$$\text{IC}(Q_t^{\text{ref}}, \hat{Q}_t) = \frac{1}{|Q_t^{\text{ref}}||\hat{Q}_t|} \sum_{\substack{q^{\text{ref}} \in Q_t^{\text{ref}} \\ \hat{q} \in \hat{Q}_t}} \text{sim}(q^{\text{ref}}, \hat{q}) \tag{12}$$

where $\hat{Q}_t, Q_t^{\text{ref}}$ are the predicted question set and the reference question set at the same dialogue turn t, $\text{sim}(\cdot, \cdot)$ is the semantic similarity function between two questions.

Action Dependency Alignment (AD) Since predicted actions are scheduled sequentially, valid action sequences should respect the system's default workflows and dependency constraints. We therefore define Action Dependency Alignment (AD) to measure the extent to which consecutive predicted actions $\hat{a_{t-1}}, \hat{a_t}$ conform to these dependencies:

$$\text{AD}(\hat{A}) = \frac{1}{|\hat{A}_t|} \sum_{\hat{a}_t \in \hat{A}_t} \left(\sum_{\hat{a}_{t-1} \in \hat{A}_{t-1}} P(\hat{a}_t \mid \hat{a}_{t-1}) \right) \tag{13}$$

where $P(\hat{a}_t \mid \hat{a}_{t-1})$ represents the conditional probability of action $\hat{a}_t$ following $\hat{a}_{t-1}$ based on the learned action dependency graph G from the system's default workflows stored in the knowledge

base $\mathcal{A}$. Specifically, $P(\hat{a}_t \mid \hat{a}_{t-1})$ is computed by normalizing 1-gram frequency counts from G **with Laplace smoothing for unseen transitions**:

$$P(\hat{a}_t \mid \hat{a}_{t-1}) = \frac{C(\hat{a}_{t-1}, \hat{a}_t) + \epsilon}{\sum_{\hat{a}' \in \mathcal{A}} [C(\hat{a}_{t-1}, \hat{a}') + \epsilon]} \tag{14}$$

where $\epsilon = 1e^{-7}$ as the smoothing parameter for unseen action pairs, $C(\hat{a}, \hat{a}_t)$ is the count of transitions from action $\hat{a}_{t-1}$ to action $\hat{a}_t$ in the knowledge base $\mathcal{A}$, $\sum_{\hat{a}'} C(\hat{a}_{t-1}, \hat{a}')$ is the total count of all transitions starting from action $\hat{a}_{t-1}$.

Without losing generality, the rate-based metrics introduced so far treat datasets with and without observed triggers equally. However, **in settings without observed triggers at the dialogue turn** t, the alignment between predicted action opportunities $\hat{a}_t \in \hat{A}_t$ and the truly-triggered actions in the system A_t requires more special considerations. To this end, we introduce the following three metrics.

Actual Trigger Rate (ATR) To measure how many triggered actions actually appear in the predicted action opportunities, we use the percentage of triggered actions out of all identified opportunities:

$$\text{ATR}(t, \hat{A}_t) = \frac{\left|\left\{\hat{a} \in \hat{A}_t \wedge \hat{a} \in_{\text{name}} A_t^{\text{ref}} \wedge \hat{a} \text{ is a TRIGGERED action}\right\}\right|}{|\hat{A}_t|} \tag{15}$$

False Triggered Action Rate (FTR) As a counterpart to ATR, we measure the proportion of falsely triggered actions—i.e., actions predicted as triggered do not appear in the observed triggered action set. FTR quantifies the extent of over-triggering by computing the fraction of predicted actions that are not actually triggered:

$$\text{FTR}(t, \hat{A}_t) = \frac{\left|\left\{\hat{a} \in \hat{A}_t \wedge \hat{a} \notin_{\text{name}} A_t^{\text{ref}} \wedge \hat{a} \text{ is a TRIGGERED action}\right\}\right|}{|\hat{A}_t|} \tag{16}$$

False Ready Action Rate (FRR) As the counterpart to RAR 3.2, we measure the proportion of incorrectly marked-as-ready actions that do not appear in the reference action set A_t:

$$\text{FRR}(t, \hat{A}_t) = \frac{\left|\left\{\hat{a} \in \hat{A}_t \wedge \hat{a} \notin_{\text{name}} A_t^{\text{ref}} \wedge \hat{a} \text{ is a ready action}\right\}\right|}{|\hat{A}_t|} \tag{17}$$

Together, ATR, FTR, and FRR serve as complementary metrics for evaluating whether actions are triggered appropriately, ensuring correct activation while minimizing unnecessary side effects.

Proactiveness Rating (PR) To quantify the degree of proactiveness exhibited by predicted action opportunities, we adopt an LLM-based subjective evaluation that assigns a discrete proactiveness score to each action on a scale from -1 to 5:

- 5: Highly proactive assistance without any explicit request
- 4: Proactive with minimal prompting
- 3: Moderately proactive
- 2: Slightly proactive
- 1: Minimally proactive
- -1: Cannot be evaluated

In contrast to **Proactive Timing (PT)** in Section 3.2, which provides an objective measure of alignment with reference actions, **PR** serves as a simplified "subjective" assessment of proactive behavior, evaluated by a profile-based LLM-based judger configured (Appendix 13, 14 and 15).

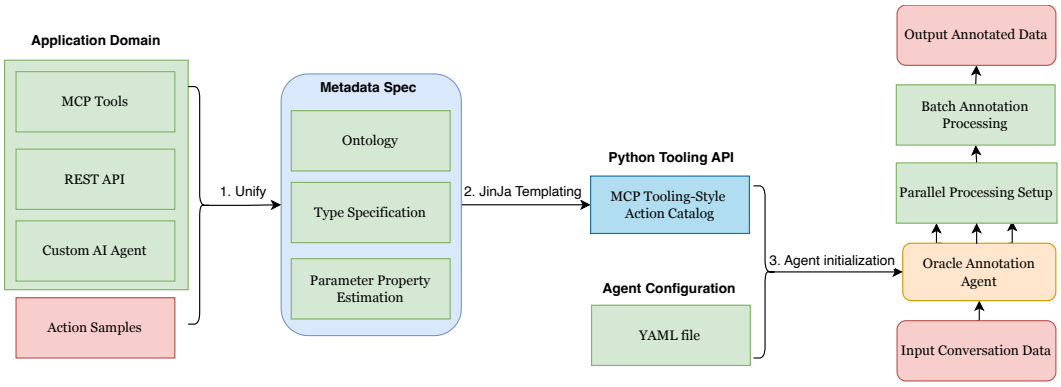


Figure 7: The annotation data pipeline unifies domain metadata and dialogue inputs into standardized tool catalogs, initializes a configurable AI agent, and performs scalable batch annotation to produce structured annotated outputs.

D Annotation Data Pipeline

The workflow of the annotation data pipeline is illustrated in Figure 7. Section D.1 briefly describes how metadata specifications are generated from automation domain APIs and automation samples. Section D.2 then details how the oracle agent is initialized using a configuration-driven design, and how conversation data are partitioned and annotated in parallel to produce the final annotated outputs. Finally, we describe the alignment validation procedure for datasets with action observations, such as ABCD+.

D.1 Metadata Specification & Tooling Catalog Model

Metadata specification generation aims to define and reconsolidate task automation operations in a unified representation, enabling the subsequent task scheduling process to be **reformulated as action opportunity discovery in a manner analogous to MCP-style tooling calls**. A key distinction from standard MCP tooling, however, is that MCP tool calls typically assume complete parameter mapping, restricting agents to cases where all required parameters have been fully resolved. In contrast, our framework **explicitly supports partial parameter mapping and readiness analysis**, which is essential for the annotation process as well as for baseline and RL-trained models.

As the entry point of the entire pipeline, metadata specification generation interfaces with target systems through supported APIs—including MCP tools, REST APIs, or custom AI agents—to extract the following components:

1. **Ontology**: a JSON file that defines the hierarchical structure of actions, the workflows composed of these actions, and their inter-action relationships.
2. **Type specification**: a JSON file that describes each supported action, including its semantic description and detailed parameter types.
3. **Parameter property estimation**: a JSON file that determines whether each parameter is required or optional. To derive this, we employ a heuristic approach that generates automation samples from system interactions and evaluates parameter presence across all cases. Parameters observed in every instance are marked as *required*, while others are marked as *optional*.

These three components jointly serve as inputs to a Jinja-basedPallets (2025) text templating process, which compiles them into a self-described Python dictionary containing all available function call definitions. This resulting artifact constitutes what we refer to as the *MCP tooling-style action catalog*.

```
{
  "catalog_metadata": {
    "name": "domain_catalog",
    "version": "1.0.0",
    "domain": "target_domain"
  },
  "mcp_tools": [...],
  "api_definitions": [...],
  "custom_agents": [...]
}
```

Figure 8: A JSON snippet of Tool Catalog

D.1.1 UNIFIED TOOL CATALOG ARCHITECTURE

Our tool catalog supports three distinct types of automation capabilities:

1. **MCP (Model Context Protocol) Tools**: Standard MCP-compliant tools with JSON schema validation, providing native compatibility for modern LLM tool use.
2. **API Definitions**: REST API endpoints with configurable headers, methods, and parameters, enabling integration with existing services.
3. **Custom AI Agents**: Complex multi-step workflows and specialized automation routines that extend beyond simple API calls.

Essentially, this unified workflow converts domain-specific metadata into standardized and consistent tool catalogs that integrate seamlessly with the annotation pipeline, independent of the underlying business domain.

D.1.2 CONFIGURATION-DRIVEN DESIGN

The annotation system adopts a configuration-driven design in which a unified tool catalog is specified via JSON configuration files. This abstraction enables rapid adaptation to new domains without requiring code-level modifications. Figure 8 illustrates a representative tool catalog schema.

This explicit separation between tool definitions and annotation logic allows the same annotation pipeline to generalize across domains—ranging from customer service scenarios in the ABCD dataset to financial workflows such as loan applications—without architectural changes. Moreover, the configuration-based design facilitates the evolution of available tools by supporting straightforward addition, deprecation, or modification of catalog entries.

D.1.3 ORACLE ANNOTATION PROCESS

Given a dialogue and tool catalog, an LLM-powered oracle agent identifies proactive opportunities by:

1. **Context Analysis**: Processing the full dialogue to understand user intent and information state
2. **Opportunity Identification**: Matching dialogue context against available tools to identify actionable moments
3. **Parameter Extraction**: Determining which tool parameters can be filled from the conversation
4. **Readiness Assessment**: Evaluating whether sufficient information exists to trigger each tool
5. **Quality Scoring**: Assigning confidence scores to filter high-quality annotations

The oracle sees the complete conversation (past and future), providing reference annotations for what proactive actions would have been beneficial at each turn. This hindsight-enabled annotation creates realistic training targets that online agents can learn to approximate.

D.1.4 VALIDATION WITH ACTION OBSERVATIONS

For systems/datasets with execution logs (we formally name them after action observations), e.g., ABCD, we align output prediction annotations with action observations (in ABCD, they are triggered actions at the ASSISTANT turns) to validate annotation quality. This provides an additional quality control and enables analysis of the gap between ideal proactive behaviors and current reactive patterns.

D.2 ORACLE ANNOTATION AGENT

To facilitate faster annotation, the oracle annotation agent is initialized using a YAML configuration with flexible, easily modifiable parameters, and the annotation workflow can be parallelized according to available system resources.

Agent Initialization Figure 9 shows a typical YAML configuration for the oracle annotation agent, enabling configurable LLM backends, tool catalog integration, and annotation customization. The agent is instantiated using the specified backend API (e.g., OpenAI or Novita).

Parallel Processing The agent supports parallel execution via the `max_workers` parameter, which specifies the number of concurrent annotation workers (default: 4). For robust recovery and controlled execution, we provide `start_index` to resume processing from a specified dialogue index, and `annotated_max_dialogues_num` to cap the number of dialogues processed in a single run. The system also tracks progress by reporting completed dialogues with timing statistics and stores outputs using gzip compression to efficiently handle large-scale datasets.

Reference Ready Action Range In addition to annotated action opportunities, we record the *reference ready action range* for each dialogue in the annotation data. For action $a \in \mathcal{A}$, where $\mathcal{A}$ is the unified action catalog, dialogue $D[i]$ from dialogue dataset $D, i \in [1, |D|]$ and T is the maximum dialogue turn number in $D[i]$, we define:

- $\mathcal{R}^{s_t}_{a,D[i]} = \{(t, s_t) : t \in [1, T], s_t \in \text{Status Set}\}$ - all occurrences with status s_t for dialogue $D[i]$
- $\mathcal{R}^{\text{ready}}_{a,D[i]} = \{t : (t, s_t) \in \mathcal{R}^{\text{all}}_{a,D[i]}, s_t \text{ is ready status}\}$ - ready windows for dialogue $D[i]$

The resulting reference-ready action range $\mathcal{R}^{\text{ready}}_{a,D[i]}$ is recorded per dialogue and used to assess annotation quality and evaluate proactive timing for both baseline and RL-trained models.

D.3 VALIDATION OF READY ACTION ALIGNMENT ON DATASETS WITH ACTION OBSERVATIONS

To ensure action consistency between the output reference action dataset and **observed triggered actions**, we validate the alignment between the reference ready action ranges and the observed triggered actions. Dialogues that exhibit misalignment are filtered out, resulting in a more consistent annotation dataset.

D.3.1 ALIGNMENT VALIDATION FORMALIZATION

Given **the action observations set** G with observed triggers and **the reference ready action range** $\mathcal{R}^{\text{ready}}_{a,D}$ in corresponding oracle annotation dataset D, we try to measure the consistency level between observed triggered actions in $G[i, t]$ and ready action ranges $\mathcal{R}^{\text{ready}}_{a,D[i]}, i \in [1, |D|]$ at the dialogue turn $t, t \in [1, |D[i]|]$, then aggregate at dialogue level. For each observed triggered action $g \in G[i, t]$, we calculate its **Early ready criteria score (EC)** by checking whether a corresponding reference ready action of the same type becomes ready sufficiently early—i.e., no later than the early turn threshold parameter σ before the triggering turn t:

$$\text{EC}(g) = \begin{cases} 1.0, & \text{if } a^\star.\text{start} + \sigma \leq t, \\ 0.0, & \text{otherwise}, \end{cases} \tag{18}$$

```
# LLM Settings
llm:
  model: "gpt-4o-mini"
  temperature: 0.1
  max_tokens: 4000
# Tool Catalog Settings
tool_catalog:
  use_common_tools: true
  custom_tools: {}
# Annotation Settings
annotation:
  batch_size: 5
  validate_output: true
  retry_on_failure: true
  max_retries: 3
# Output Settings
output:
  output_suffix: "_annotated"
  pretty_print: true
  save_individual_turns: false
# Prompt Settings
prompts:
  system_prompt: |
    You are an expert AI assistant specialized in analyzing dialogues to identify proactive automation
      opportunities.
    Your task is to annotate dialogue turns to identify when proactive actions could be beneficial in multi-
      party conversations.
    For each dialogue turn, you will identify:
    1. Action opportunities that could be triggered
    2. Required and optional inputs for each action
    3. Readiness maturity of participants
    4. Confidence in triggering the action
    5. Current status of the action trigger
    Always provide output in valid JSON format following the specified schema exactly.

  task_prompt: |
    ## TASK
    Analyze the following dialogue and annotate turn {turn_number}
    ## FULL DIALOGUE CONTEXT
    {dialogue_context}
    ## CURRENT TURN TO ANNOTATE
    Turn {turn_number}: {current_speaker} says: "{current_text}"
    ## AVAILABLE ACTION OPPORTUNITIES
    {tool_catalog}
    ## ANNOTATION SCHEMA
    {{
      "dialogue_turn": {turn_number},
      "speaker": "{current_speaker}",
      "text": "{current_text}",
      "proactive_annotations": [{{
        "action_opportunity": {{
          "name": "ActionName",
          "description": "What this action does",
          "inputs": {{
            "required": [{{
              "input_name": "InputName",
              "provided": true/false,
              "value": "The value of the input if provided"
            }}],
            "optional": [{{
              "input_name": "OptionalInput",
              "provided": true/false,
              "value": "The value of the input if provided"
            }}],
            "readiness_maturity": "low/medium/high",
            "trigger_confidence": "low/medium/high",
            "action_trigger_status": "pending/ready_to_trigger/triggered/repeatable/dismissed"
          }}
        }}]
      }}
    }}
    ## GUIDELINES
    - Only include action opportunities that are relevant to the current turn
    - Consider the full dialogue context when assessing input availability
    - Be conservative with "ready_to_trigger" status - only use when conditions are clearly met
    - If no proactive opportunities exist for this turn, return empty "proactive_annotations" array
    - Ensure all JSON is valid and follows the schema exactly
    ## RESPONSE
    Provide only the JSON response, no additional text.

# Logging Settings
logging:
  level: "INFO"
  log_file: "annotation.log"
  log_to_console: true
```

Figure 9: Oracle Agent Configuration

Threshold σ	Number of Actions with EC(g) = 1.0	Percentage of Actions with EC(g) = 1.0	Dataset Score EC(G)	Dataset Score EC(G) $>$ 0.8
4	3,477	11.78%	0.1217 ± 0.2443	4.07%
3	4,982	16.88%	0.1837 ± 0.2883	6.62%
2	7,424	25.16%	0.2850 ± 0.3326	11.42%
1	12,602	42.71%	0.4682 ± 0.3499	21.10%
0	25,715	87.14%	0.8877 ± 0.1926	70.27%

Table 3: Impact of early turn threshold on action consistency (EC), and dialogue-level performance metrics.

$$a^\star = \underset{\substack{a \in \mathcal{R}^{\text{ready}}_{a,D[i]} \\ a.\text{name}=g.\text{name}}}{\arg\min} \; a.\text{start}, \tag{19}$$

where $a.start$ is the first turn at which a becomes ready.

Aggregating over the entire observed action set G, we define **the dataset-level Early-Ready Criterion score** EC(G) as:

$$\text{EC}(G) = \frac{1}{|G|} \sum_{g \in G} \text{EC}(g) \tag{20}$$

D.3.2 Alignment Validation on ABCD+

To examine consistency changes in ABCD-derived annotations, we vary $\sigma \in \{0, 1, 2, 3, 4\}$ and report the distribution of actions with EC=1.0 and dialogue consistency scores above the threshold value (empirically, set by 0.8), as shown in Table 3.

This observation aligns with our intuition that, as the early turn threshold σ increases, the number of valid dialogues decreases gradually. When setting early_turn_threshold = 0—meaning that the reference ready action range covers just the observed triggered action at its exact turn—only 70.27% of the annotated dialogues remain valid. We consider these dialogues as consistent and retain them for baseline validation and training purposes, resulting in an ABCD+ split of 5,647 training dialogues, 703 validation dialogues, and 692 test dialogues from the original set of 10,042 annotated dialogues.

E Baseline

This section presents the design of each baseline with provided YAML configurations, the workflow to collect runtime metrics, and the visualization tools developed to diagnose potential issues in predicted action opportunities. The overall workflow is illustrated in Figure 10.

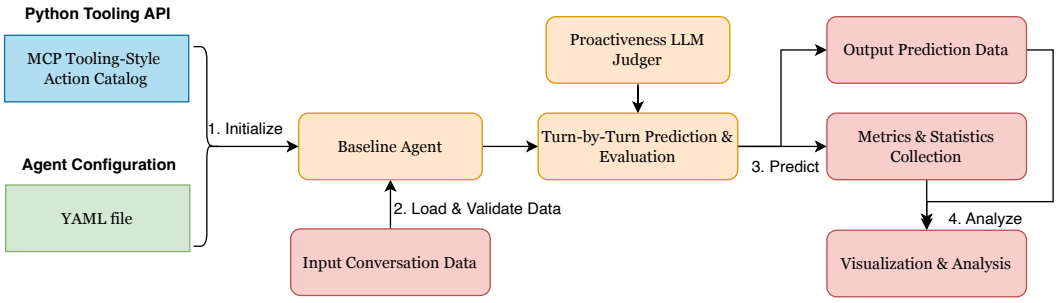


Figure 10: Baseline Workflow: We initialize each baseline agent using the unified action catalog and its configuration. The agent then operates in a turn-by-turn manner, generating predictions while collecting performance statistics. Beyond predefined metrics, an LLM-based judger evaluates proactive behavior, and visualization tools support further analysis.

E.1 DESIGN RATIONALE

Prompting Strategy Design. Our three prompting strategies progressively externalize reasoning and memory. Non-Reasoning relies solely on internal model knowledge, Reasoning makes deliberation explicit prior to action prediction, and ASG further maintains temporal state across turns. Each strategy isolates a specific capability—contextual understanding, deliberative reasoning, and temporal state awareness—allowing us to analyze which components of proactive behavior benefit from explicit prompting versus learning-based optimization. These observations motivate our RL approach, which learns to balance such trade-offs dynamically.

SFT Exclusion Rationale. We deliberately exclude supervised fine-tuning (SFT) as a baseline because our application domain requires agents to learn flexible patterns from conversations rather than replicate exact behaviors from examples. The space of possible conversation patterns and action timings is too difficult to be fully covered by supervised examples. Instead, our baselines focus on training-free approaches that can be enhanced through reinforcement learning, which learns optimal policies from interaction with the environment rather than static examples.

We evaluate these baselines using GPT-4.1-mini, GPT-5.1, Gemini-2.5-flash, Claude-sonnet-4-20250514, and Qwen2.5-14B-Instruct. We adopt Qwen2.5-14B-Instruct as our baseline choice, as its 4-bit quantized variant is later fine-tuned with LoRA to obtain ProActor-Q4, enabling a direct and controlled comparison of performance gains.

E.2 NON-REASONING & REASONING BASELINE

Direct Prompting: Non-Reasoning Agent Our first baseline tests whether LLMs can naturally identify action opportunities, capture action dependencies, and judge triggering timing without explicit reasoning. The agent predicts actions and their parameters based solely on the current conversation, and we also prompt it to raise clarifying questions when needed. This baseline establishes the lower bound of what can be achieved through contextual understanding.

Reasoning Agent On top of direct prompting, this baseline introduces explicit proactive reasoning to improve decision-making. We prompt the model to first output a `<think>` section where it analyzes the current conversation context, identifies potential action opportunities, and reasons about appropriate triggering timing. Only after this reasoning step is completed, it outputs an `<action>` section with specific actions and parameters, and triggering readiness. This tests whether forcing models to "think about specific aspects" before acting improves their proactive behavior.

E.3 ASG+REASONING BASELINE DETAIL

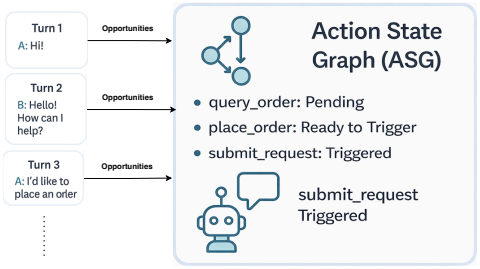
In this section, we focus on illustrating the details of Action State Graph (ASG) and Action State Update Heuristics shown in Figure 11.

In the ASG+Reasoning agent, the action trigger status follows a fully connected graph where any valid state can transition to any other valid state, representing a complete directed graph with self-loops allowed:

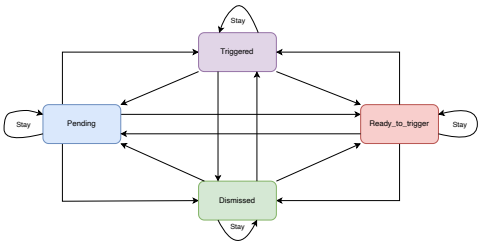
$$\begin{aligned} \mathcal{S} = \{&\text{PENDING}, \text{READY\_TO\_TRIGGER}, \\ &\text{TRIGGERED}, \text{REPEATABLE}, \text{DISMISSED}\}, \\ &\forall\, s_i, s_j \in \mathcal{S}, s_i \to s_j. \end{aligned} \tag{21}$$

In-Memory Action State We use the UPDATEACTIONSTATES procedure to update the action state in the agent's runtime memory within the context of a single dialogue. For Non-Triggered action opportunities, their state transitions must follow the defined rules. In comparison, triggered actions directly overwrite any existing state without validation. More details, refer to Algorithm 1.

Action State Representation In Prompt When the ASG+Reasoning agent predicts action opportunities or raises questions, the in-memory action state `actions` (Algorithm 1) is converted



(a) ASG overview: The Action State Graph (ASG) maintains predicted action opportunities and tracks their status transitions in application memory. The updated state is exposed to the baseline model by injecting a structured JSON representation into the prompt.



(b) Action State Update Heuristics: For each action opportunity, we design state-transition heuristics that specify the set of permissible target statuses given the current status. In ABCD+ and Home Loan, the resulting transition graph is simplified to be fully connected.

Figure 11: Action State Graph (ASG). The left panel illustrates the ASG overview, while the right panel presents the heuristic update rules governing action state evolution.

Algorithm 1 Update Action States with Validation

procedure UPDATEACTIONSTATES($new\_actions$, $is\_triggered\_action$)
  $actions \leftarrow$ current proactive action opportunities list
  for all $new\_action \in new\_actions$ **do**
    $existing\_action \leftarrow$ NONE; $existing\_index \leftarrow -1$ ▷ Find if action already exists by name
    for $i = 0$ **to** $|actions| - 1$ **do**
      if $actions[i].name = new\_action.name$ **then**
        $existing\_action \leftarrow actions[i]$; $existing\_index \leftarrow i$; **break**
      end if
    end for
    if $is\_triggered\_action =$ TRUE **then** ▷ Case 1: Triggered actions - overwrite or append
      if $existing\_action \neq$ NONE **then**
        $actions[existing\_index] \leftarrow new\_action$
      else
        $actions$.APPEND(new_action)
      end if
    else ▷ Case 2 & 3: Predicted actions
      if $existing\_action \neq$ NONE **then**
        $current\_status \leftarrow existing\_action.action\_trigger\_status$
        $new\_status \leftarrow new\_action.action\_trigger\_status$
        if ISVALIDTRANSITION($current\_status$, new_status) **then**
          $actions[existing\_index] \leftarrow new\_action$
        end if ▷ If invalid transition, reject update (do nothing)
      else
        $actions$.APPEND(new_action) ▷ Case 3: New predicted action
      end if
    end if
  end for
end procedure

function ISVALIDTRANSITION($current\_status$, new_status)
  $valid\_states \leftarrow$ {PENDING, READY_TO_TRIGGER, TRIGGERED, REPEATABLE, DISMISSED}
  if $current\_status \notin valid\_states$ **or** $new\_status \notin valid\_states$ **then**
    return FALSE
  end if
  return TRUE
end function

into a structured JSON representation (Figure 12) and injected into the action- and question-related prompts by replacing the `action_states` variable.

```
[
  {
    "action_opportunity": {
      "name": "ActionName",
      "description": "What this action does",
      "inputs": {
        "required": [
          {
            "input_name": "InputName",
            "provided": true,
            "value": "The value of the input if provided"
          }
        ],
        "optional": [
          {
            "input_name": "InputName",
            "provided": false,
            "value": "The value of the input if provided"
          }
        ]
      },
      "parameters_readiness_maturity": "high",
      "trigger_confidence": "high",
      "action_trigger_status": "ready_to_trigger"
    },
    ...More Actions
  }
]
```

Figure 12: JSON representation of action states used in the Reasoning+ASG prompt to replace the `action_states` variable.

E.4 Agent Configuration

Similar to the oracle annotation agent configuration (Figure D.2), we provide a YAML configuration file for baseline initialization. This file specifies the LLM backend, tool catalog settings, and the prompts used for action opportunity prediction and question generation. Non-reasoning E.2, reasoning E.2, and ASG+reasoning E.3 baselines all use the shared `dialogue_context` and `tool_catalog` variables to dynamically inject the current dialogue context and the unified tool catalog definition in Python. The only difference is that the ASG+reasoning agent needs to use `action_states` to maintain the latest action states.

Baseline Configurations Baseline YAML files support model switching via `llm.model` and allow configuration of non-OpenAI APIs through `llm.api_key_name` and `llm.base_url`. Sample configurations are given below:

1. YAML configuration of Direct Prompting agent baseline, refer to Figure 13

2. YAML configuration for Reasoning agent baseline, refer to Figure 14

3. YAML configuration for ASG+reasoning agent baseline, refer to Figure 15

```yaml
# Baseline Configuration File
# Configuration for LLM-based actor model
# LLM Settings
llm:
  model: "openai/gpt-4.1-mini"
  temperature: 0.1
  max_tokens: 4096
  using_short_model_name: true
# Tool Catalog Settings
tool_catalog:
  use_common_tools: true
  custom_tools: {}
# Prompt Settings: Action Prompts and Question Prompts
# 1. Action Prompts
action_prompts:
  system_prompt: |
    You are an expert AI assistant for professional, specializing in analyzing dialogues to identify
      automation opportunities and their parameters based on the client (e.g., customer) and professional (e.
      g., sales agent) conversations. Given the conversation history and available system action
      opportunities, your task is to analyze the latest turn and determine the following for all relevant
      actions:
    1. Action opportunities that could be triggered once ready
    2. Required and optional inputs for each action
    3. Parameter Readiness (readiness_maturity)
    4. Confidence in triggering the action (trigger_confidence)
    5. Current status of the action trigger (action_trigger_status)
    ## AVAILABLE ACTION OPPORTUNITIES
    <tools>
    {tool_catalog}
    </tools>
    ## RESPONSE SCHEMA
    {{
      "proactive_action_opportunities": [...]
    }}
  task_prompt: |
    ## TASK
    Analyze the following conversation for proactive automation opportunities. Provide only the JSON
      response, no additional text.
    ## DIALOGUE CONTEXT
    {dialogue_context}

# 2. Question Prompts
question_prompts:
  system_prompt: |
    You are an expert AI assistant specialized in analyzing the
    conversation between customer and sales agent to predict what
    question the sales agent is asking.
    ## RESPONSE SCHEMA
    {{
      "raised_questions": [...]
    }}
  task_prompt: |
    ## TASK
    Analyze the following dialogue and predict the question that the sales agent is asking to gather the
      missing information for the non-ready action opportunities.
    ## DIALOGUE CONTEXT
    {dialogue_context}
```

Figure 13: YAML Configuration of Direct Prompting Baseline Agent.

```yaml
# Baseline Configuration File
# Configuration for LLM-based actor model

# LLM Settings
llm:
  model: "openai/gpt-4.1-mini"
  temperature: 0.1
  max_tokens: 4096
  using_short_model_name: true

# Tool Catalog Settings
tool_catalog:
  use_common_tools: true
  custom_tools: {}

# Prompt Settings: Action Prompts and Question Prompts
# 1. Action Prompts
action_prompts:
  system_prompt: |
    You are an expert AI assistant for professional, specializing in analyzing dialogues to identify
      automation opportunities...

    ## RESPONSE SCHEMA
    <think>
    Describe your thinking about how to decide the action opportunities to be triggered in the last turn.
    </think>
    <action>
    {{
      "proactive_action_opportunities": [...]
    }}
    </action>

  task_prompt: |
    ## TASK
    Analyze the following conversation for proactive automation opportunities and output automation
      opportunities triggered by the latest turn. Output only the <think> and <action> sections, with no
      additional text.

    ## DIALOGUE CONTEXT
    {dialogue_context}

# 2. Question Prompts
question_prompts:
  system_prompt: |
    You are an expert AI assistant specialized in analyzing the conversation between customer and sales
      agent...

    ## RESPONSE SCHEMA
    <think>
    Describe your thinking about how to decide the question to be raised in the last turn...
    </think>
    <question>
    {{
      "raised_questions": [...]
    }}
    </question>

  task_prompt: |
    ## TASK
    Analyze the following dialogue and predict the question that the sales agent is asking... Output only
      the <think> and <question> sections, with no additional text.

    ## DIALOGUE CONTEXT
    {dialogue_context}
```

Figure 14: YAML Configuration of Reasoning Baseline Agent: reasoning with <think> and <action>/<question> sections

```yaml
# Baseline Configuration File
# Configuration for LLM-based actor model

# LLM Settings
llm:
  model: "openai/gpt-4.1-mini"
  temperature: 0.1
  max_tokens: 4096
  using_short_model_name: true

# Tool Catalog Settings
tool_catalog:
  use_common_tools: true
  custom_tools: {}

# Prompt Settings: Action Prompts and Question Prompts
# 1. Action Prompts
action_prompts:
  system_prompt: |
    You are an expert AI assistant for professional...

    ## TRACKING ACTION STATES
    You are also given the ACTION STATES in the input data: the action opportunities predicted or triggered
      in the previous turns. Please use it to track and decide the action opportunity candidates in the
      latest turn.

    ## RESPONSE SCHEMA
    <think>
    Describe your thinking about how to decide the action opportunities to be triggered in the last turn.
    </think>
    <action>
    {{
      "proactive_action_opportunities": [...]
    }}
    </action>

  task_prompt: |
    ## TASK
    Analyze the following conversation for proactive automation opportunities and output automation
      opportunities triggered by the latest turn. Output only the <think> and <action> sections, with no
      additional text.

    ## ACTION STATES
    {action_states}

    ## DIALOGUE CONTEXT
    {dialogue_context}

# 2. Question Prompts
question_prompts:
  system_prompt: |
    You are an expert AI assistant specialized in analyzing the conversation between customer and sales
      agent...

    ## TRACKING ACTION STATES
    You are also given the ACTION STATES in the input data: the action opportunities predicted or triggered
      in the previous turns. Please use it to track and decide the action opportunity candidates in the
      latest turn.

  task_prompt: |
    ## TASK
    Analyze the following dialogue and predict the question...
    Output only the <think> and <question> sections, with no additional text.

    ## ACTION STATES
    {action_states}

    ## DIALOGUE CONTEXT
    {dialogue_context}
```

Figure 15: YAML Configuration of Reasoning Baseline Agent: reasoning with <think> with Action State Graph support

Due to page length, we omit the schema structures of action opportunities and raised questions, which are similar to action opportunities specification in YAML file 9.

Besides, a proactiveness LLM judger can be configured via the `llm_judger_config_file` command-line argument, which specifies the corresponding YAML file. An example is shown in Figure 16.

E.5 Result Analysis & Visualization

Since the ABCD+ and Home Loan datasets consist of long dialogues (typically exceeding 20 turns), all performance metrics—including proactiveness metrics from the LLM-based judger—are com-

```
# LLM Judger Configuration File for ASG Support
judgers:
  - name: "proactiveness_evaluator"
    model: "openai/gpt-4.1-mini"
    temperature: 0.1
    max_tokens: 4096
    using_short_model_name: true
    api_key_name: <INTERNAL_API_KEY>
    base_url: <INTERNAL_SERVER_BASE_URL>
    settings:
      max_retries: 3
      retry_on_failure: true

    system_prompt: |
      You are a helpful assistant tasked with judging whether a proactive action prediction is truly
      proactive - meaning it was not explicitly requested by the customer but still aligns with their intent.

      You are given:
      - PROACTIVE ACTION PREDICTIONS AT CURRENT TURN
      - DIALOGUE UNTIL CURRENT TURN

      Your task is to:
      1. Determine if the customer explicitly requested the predicted action.
      2. Assess the proactiveness level - how well the prediction matches the customer's intent without a
      direct request.

      # RATING CRITERIA
      The proactiveness level is calculated based on:
      -1 - Conversation is insufficient to judge.
       1 - Explicit or almost explicit request, target action missing.
       2 - Explicit request, target action present, inaccurate parameters.
       3 - Explicit request, target action present, accurate parameters.
       4 - Implicit request, target action present, inaccurate parameters.
       5 - Implicit request, target action present, accurate parameters.

      # RESPONSE SCHEMA
      Return only valid JSON without any additional text:
      {
        "reasoning": "Brief explanation (<=50 words)",
        "rating": -1/1/2/3/4/5
      }

    task_prompt: |
      ## TASK
      You are given the conversation history so far and the proactive action predictions from the proactive
      agent for the latest turn.

      # DIALOGUE UNTIL CURRENT TURN
      {dialogue_context}

      # PROACTIVE ACTION PREDICTIONS AT CURRENT TURN
      {proactive_action_predictions}

      Evaluate the proactiveness of these predictions and return your assessment in the specified JSON
      format.
```

Figure 16: LLM Judger Configuration: Proactiveness evaluation with 5-point rating scale

puted online at the generation time of predicted action and raised question. All metrics and statistics are stored in JSON format for analysis. To support human inspection in nuanced cases, we additionally developed a web-based interface that loads both the aggregated metrics and corresponding prediction outputs, enabling evaluators to examine summary statistics as well as dialogue-level states and predictions (Figure 17).

F ART-F FRAMEWORK

This section introduces core components of ART-F, our open-source framework for end-to-end RL training, with a focus on how its inference cluster and training infrastructure maximize resource utilization under GPU constraints (Figure 4). The design and implementation of the ART-F client—which matches inference server capacity with training throughput—are detailed in Section G.

F.1 INFERENCE PHASE

Although ART's original inference design supports only a single inference server per GPU, quantized LoRA models typically consume only a small fraction of GPU resources, leading to substantial under-utilization and limited inference throughput. To address this limitation, ART-F introduces an

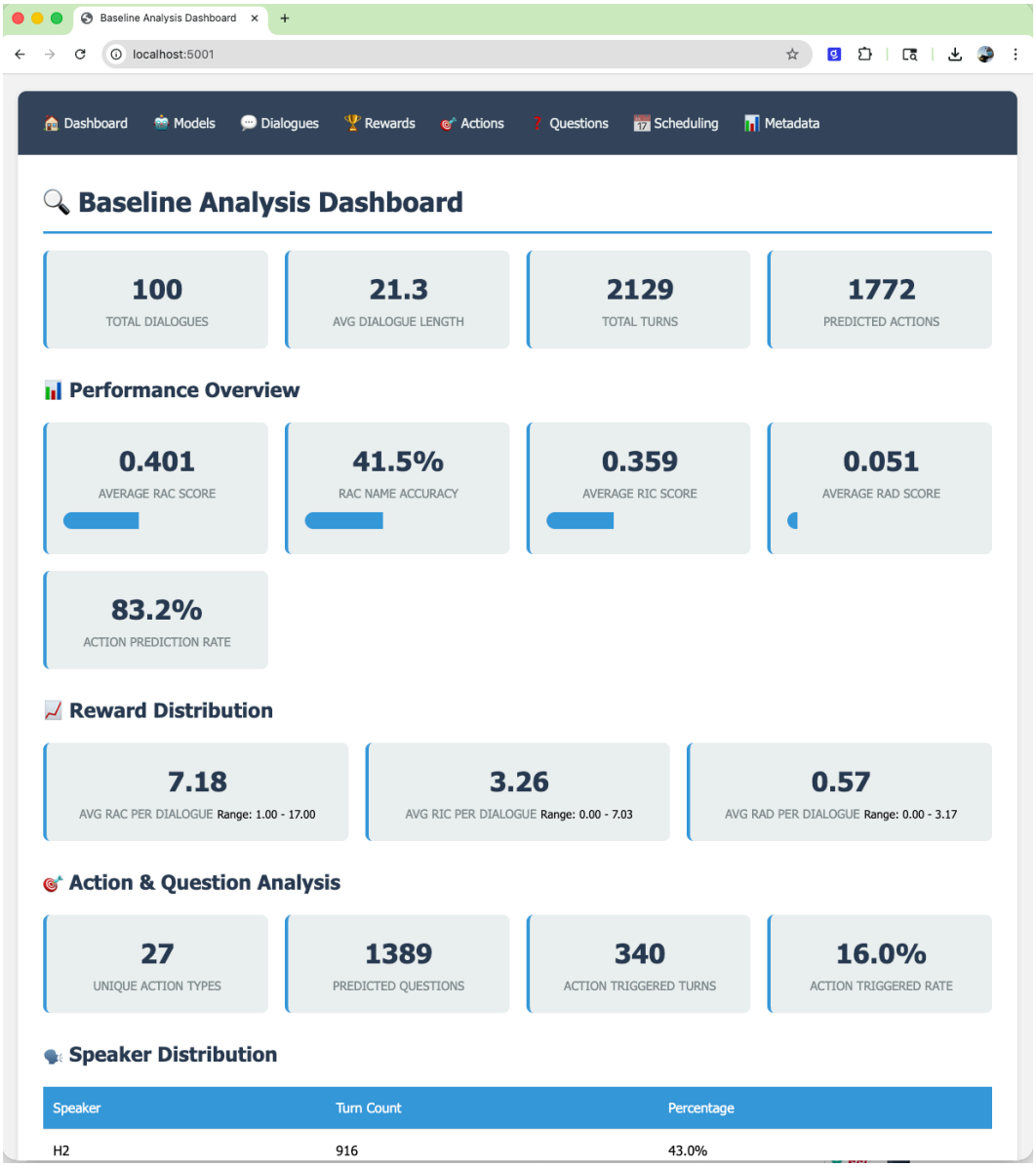


Figure 17: Web-based visualization interface for baseline performance analysis. The dashboard aggregates dialogue-level and turn-level metrics, including action prediction statistics, reward distributions, and trigger behaviors, and supports interactive inspection of individual dialogues and predicted action states.

inference cluster composed of multiple inference server instances coordinated by a load-balancing proxy that monitors runtime status. This design improves GPU utilization and significantly increases rollout throughput. Moreover, by exposing real-time cluster status, the load balancer enables RL rollout clients to make informed scheduling decisions—such as selecting available servers or early-terminating rollouts under saturation—thereby enhancing runtime flexibility and system robustness.

The entire lifecycle of the inference cluster is managed by the inference cluster manager. For simplification purposes, we use **a unique cluster configuration** for both inference and training, and allows the inference cluster manager and the training manager to access their separate sections. An overview of the inference phase is illustrated in Figure 18.

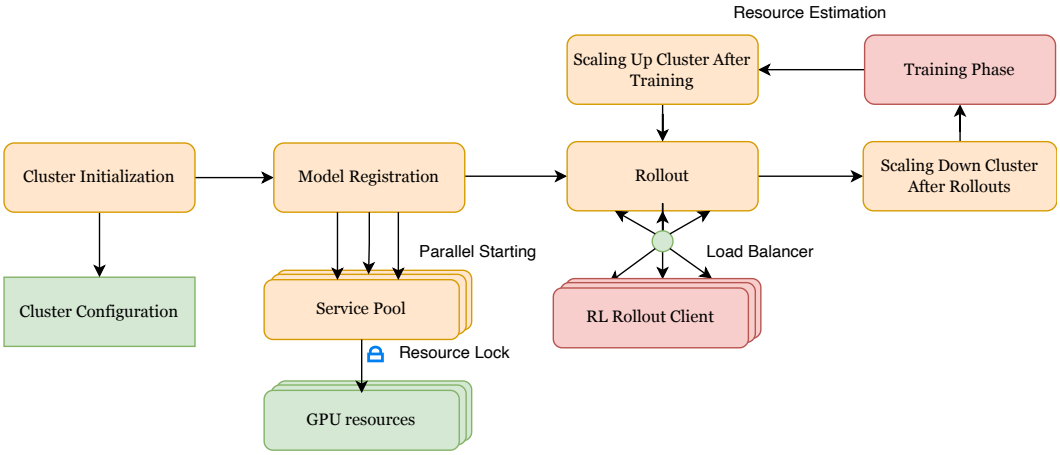


Figure 18: Inference Phase Summary: During inference, ART-F dynamically initializes and scales an inference cluster, allocates GPU resources via a service pool with resource locking, and executes parallel rollouts through the RL rollout client, before scaling down the cluster to release resources once rollouts are completed.

Initialization The inference cluster manager reads the cluster configuration from the YAML configuration, which specifies the scaling policy—including the maximum number of servers per model (`max_servers_per_model`), the server port allocation range (`port_range_start` to `port_range_end`, e.g., 8000–8010), and the load-balancing strategy (`load_balancing_strategy`, such as round-robin, random, or response-time–based). After validating it, the manager registers the model and initializes N vLLM inference server instances, each reserving a fixed fraction of maximum GPU memory (`gpu_memory_per_server`) on a single GPU.

To minimize startup latency while respecting Unsloth's process-level lock Han et al. (2023) during model loading, the N vLLM instances are launched in parallel with a cross-process resource lock held throughout the loading phase. Once all inference servers successfully pass health-check (PING) tests, the inference cluster manager marks the cluster as ready and begins serving requests from RL rollout clients.

Parallel Rollout After the inference cluster turns ready to handle rollout client requests, parallel requests will be distributed by the Load Balancer, which gets the next server from the server pool based on the load-balancing strategy. This Load Balancer also take charge of health monitoring & failure recovery during request handling. It also tracks the request counters and response time metrics for each vLLM server instance.

Cluster Scaling After the rollout phase, the inference cluster will be scaled down to allow all GPU resources used for training purposes. There are two different strategies for Non-DDP and DDP trainings:

1. For Non-DDP training, we scale down the vLLM server instances to only keep the server with the lowest port number and pause server monitor before training so that GPU memory locked by N-1 servers can be utilized in the following training. After the training completion, we resume monitoring for the leftover server and scale up back to N server and prepare for next rollout cycle.
2. For DDP training, we shut down the inference cluster completely and let the DDP training manager start the training cycle. Once the training is completed, we restart the whole inference cluster and enter the next rollout cycle. The reason is that in ART-F's DDP training, we trigger a multi-processes collaboration to not only resume the normal DDP training, but also monitor the progress and guarantee the step synchronization, which needs higher resource demands. More training details will be covered in next section F.2.

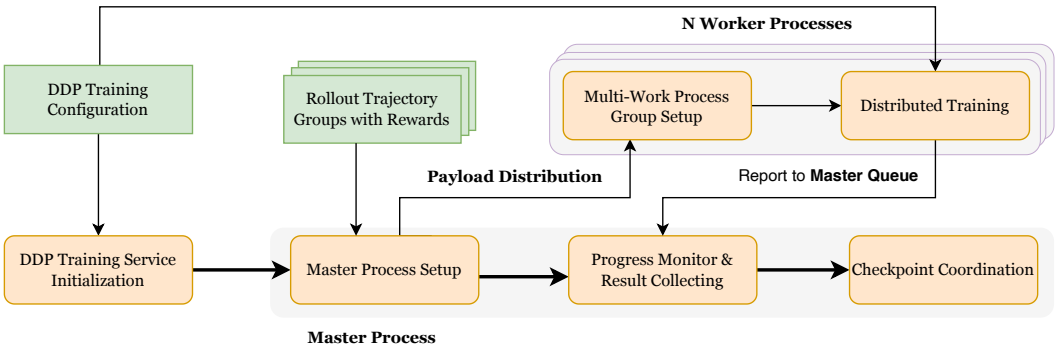


Figure 19: DDP-Based Training Phase Summary: ART-F orchestrates distributed data-parallel (DDP) training by initializing a master process and multiple worker processes, distributing training payloads—rollout trajectory groups with rewards—across workers, coordinating checkpoints, and aggregating progress and results throughout training.

F.2 DDP-Based Training

Since non-DDP training largely follows the original ART framework with only minor adaptations, we focus this section on the DDP-based training phase implemented in ART-F. Unlike standard DDP setups that rely on `torchrun`, our training framework is **custom-built** to support advanced process coordination, fine-grained progress monitoring, and flexible data distribution. The framework is designed to fully utilize GPU resources on a single multi-GPU node. As illustrated in Figure 19, ART-F adopts a master–worker architecture in which a centrally managed controller process orchestrates multiple training workers.

DDP Training Service This service is compatible with the existing local Unsloth training service in the ART framework and is responsible for loading and validating the DDP training configuration. Specifically, it performs the following checks: (1) verifies that the configured `world_size` in `DDPTrainingConfig` does not exceed the number of available GPUs when `enable_ddp` is set to `True` (2) validates the DDP backend selection (e.g., `nccl` or `gloo`) and ensures that all required process-coordination parameters—such as `master_addr`, `port`, and `ddp_timeout_minutes`—are properly specified; and (3) checks that trainer arguments cloned across worker processes (e.g., `per_device_train_batch_size` and `per_device_eval_batch_size`) remain consistent, and that the effective runtime batch size is reasonable when `batch_size_allow_adjusting` is enabled. If all validations pass, the service triggers the master process to initiate DDP training.

Master Process This process, spawned by the DDP Training Service, serves two primary roles: 1) setting up the DDP process group and initializing N worker processes; and 2) preparing a shared asynchronous queue—referred to as the Master Queue MQ — through which worker processes report their execution progress.

Specifically, we initialize the process context using `mp.get_context("spawn")`, and use `mp.Process` to launch N worker processes and create the shared MQ within the same context. The master process synchronously waits for progress updates by invoking `MQ.get()`, which naturally blocks execution until feedback is received from the workers. Only after all trajectory group payloads dispatched to the worker processes have been fully processed does the master process proceed to the final synchronization point. This design guarantees that all worker processes terminate gracefully and that the DDP training phase completes in a consistent and orderly manner.

More importantly, the master process determines how the raw training payloads—rollout trajectory groups with rewards, denoted as RG—are distributed across worker processes according to the configuration option `replicate_dataset_across_ranks`. We first align the effective payload size to the number of workers by rounding it down to $\lfloor \frac{RG}{N} \rfloor \cdot N$, ensuring an even partition across N worker processes.

If `replicate_dataset_across_ranks` is set to `True`, ART-F performs **symmetric** DDP training, in which the aligned payload RG is replicated identically across all worker processes. Otherwise, ART-F adopts **asymmetric** DDP training, where each worker process receives a disjoint subset of size $\lfloor \frac{RG}{N} \rfloor$. In theory, both settings are expected to converge under identical hyperparameter configurations. However, in our preliminary experiments, the heterogeneous rollout groups and the resulting gradient discrepancies in the **asymmetric** setting lead to noticeably less stable convergence compared to the **symmetric** setting.

Consequently, we adopt the **symmetric** DDP configuration with a reduced number of training epochs. For example, in our ABCD+ experiments on a single node with 4 $\times$ H200 GPUs, we set the number of epochs to 2 with `replicate_dataset_across_ranks` = `True`, which yields performance comparable to training for 8 epochs with `replicate_dataset_across_ranks` = `False` on the same payload RG.

Worker Processes After being triggered by the master process, each worker process first initializes the DDP process group on its assigned GPU rank via PyTorch's distributed interface `init_process_group()`, which primarily handles CUDA device assignment and validation. Given the payload allocated to each rank, workers then execute training in parallel on their assigned data partitions.

Each worker process instantiates its own `GRPOTrainer` equipped with an asynchronous task queue. The trainer consumes data chunks from this queue and reports step-wise training progress to the shared **Master Queue** (`MQ`). To prevent synchronization issues during training, we employ **step-wise barrier synchronization to enforce batch-level gradient consistency** across ranks. Worker queue states are continuously monitored and propagated to all ranks to maintain a consistent global view of execution status. If any worker reports an error condition, the DDP training session is immediately terminated and the current model checkpoint is saved, preventing indefinite blocking or timeouts caused by stalled processes.

Once all worker processes successfully complete training, the master process enters the checkpoint coordination phase. In this phase, the rank-0 worker exclusively saves the final checkpoint to persist the model state to disk and then terminates gracefully. Afterward, the master process concludes process synchronization and releases all resources reserved by the worker processes.

F.3 Other Features

Beyond the inference cluster and DDP-enabled training pipeline, ART-F introduces several additional features to simplify the end-to-end RL workflow:

1. **Integrated experiment tracking.** Besides `wandb.ai` Weights & Biases (2025b), ART-F provides built-in MLflow integration with extended system-level metrics, including GPU memory utilization, disk I/O, and network activity. We allow developers to extend the existing `MlflowSystemMonitor` to enables more customized monitoring.
2. **Customizable RULER module.** We extend the original RULER to support user-defined evaluation rules (e.g., proactiveness assessment rubrics) and seamless integration of third-party models. Trajectory saving via W&B Weave Weights & Biases (2025a) is made optional to mitigate potential sensitive data leakage during training.
3. **Databricks compatibility enhancements.** ART-F includes additional utilities to facilitate Databricks-based training workflows Databricks, Inc. (2025), such as dynamic server address resolution at runtime, temporary directory management for memory-mapped files, trajectory compression, and incremental checkpoint saving.

F.4 ART-F Parameter Tuning

As our experiments target multiple GPUs within a single machine node, training and inference in ART-F share the same GPU and system resources. Careful tuning of training and inference parameters is therefore essential to prevent resource exhaustion and cascading "avalanche" failures, such as slow inference server startup, training deadlocks due to GPU memory locking, and frequent HTTP client timeouts. We accordingly consider two key aspects of parameter tuning in ART-F.

Data Size	ART (1×H200)			ART-F (1×H200)			ART-F (4×H200)		
	#Srv/#R	Stat.	Time	#Srv/#R	Stat.	Time	#Srv/#R	Stat.	Time
100	1/1	✓	00:26:40	2/50	✓	00:14:30	8/230	✓	00:05:30
300	1/1	✓	01:25:30	2/50	✓	00:44:20	8/230	✓	00:10:45
1000	1/1	×	–	2/50	×	–	8/230	✓	00:42:45
3000	1/1	×	–	2/50	×	–	8/230	✓	01:40:20
5647	1/1	×	–	2/50	×	–	8/230	✓	03:03:03

Table 4: Inference-time scaling under different rollout configurations. #Srv denotes the number of inference servers and #R the maximum number of parallel rollouts supported. Time reports the end-to-end rollout time per training epoch in `hh:mm:ss` format. ART-F achieves near-linear inference speedups by scaling inference servers and matching client request speed on multiple GPUs.

Dynamic Training Batch Size. When enabling dynamic training batch size, we observed that DDP training achieves approximately 5–10% speedup per training step in our setting. This aligns with our setting: for an initial batch size N, the runtime batch size switches between N and $N-1$. Thus, one additional batch can be processed when enabling this option, yielding approximately 5–10% speedup, consistent with our observations.

Rollout Group Size in Inference. This parameter significantly affects inference servers hosted on a single compute node, primarily determining how many parallel requests can be handled on that node. We observed that when setting the rollout group size to 250, rollout time per epoch step decreases initially but gradually slows as rollout continues due to the increasing context. Adjusting to 230 effectively alleviates this slowdown while maintaining a good request consumption rate.

Speed-up Analysis. To understand how ART-F accelerates and stabilizes RL training, we analyze three key aspects:

(1) Rollout speed-up at the inference time: By deploying a self-managed inference cluster with coordinated client request scheduling, we maximize GPU utilization and achieve near-linear inference throughput scaling with respect to the number of GPUs. Table 4 shows the inference-time scaling under different rollout configurations. Beyond rollout acceleration, we support concurrent startup of inference servers, retaining only a minimal process-level lock to avoid Unsloth initialization conflicts. This reduces startup latency per vLLM server from approximately 2 minutes to 1.5 minutes, corresponding to a 25% speedup.

(2) Training speed-up: Although ART exposes a configurable training batch size, training tensors are constructed per instance when enqueued, which prevents the optimizer from fully exploiting batch-level parallelism. ART-F resolves this limitation by aligning tensor construction with the intended batch size, enabling true batched forward and backward passes. Combined with the dynamic batch sizing strategy described above, **ART-F achieves an approximately N-fold training throughput improvement, where N denotes the effective batch size**. Another technical enhancement is that our DDP-supported ART-F training stabilizes RL training much earlier than the original ART single-GPU training. As a result, we observe smoother training curves when DDP is enabled.

(3) Parallel Reward Evaluation and Tensor Persistence: We also parallelize RULER client requests when groups of rollout trajectories are evaluated by the reward model cluster. We introduce the parameter `--max-concurrent-api-number` in the ART-F client, which enables concurrent reward-evaluation requests across multiple trajectories. This change reduces reward evaluation time from hours to approximately 10 minutes **per training step**, bringing the full-epoch reward evaluation time on ABCD+ (5,647 dialogues) down to approximately 130 minutes **per training epoch**.

Another significant optimization is the parallelization of training-tensor persistence after tokenization of rollout trajectories. After introducing parallel persistence in ART-F, tensor processing completes in approximately 4.2 minutes (compared to 10–12 minutes originally), yielding a 2.5–2.9× speedup in tensor persistence.

G ART-F CLIENT DESIGN

This section presents a typical ART-F client design to support end-to-end training of **ProAct-Q4**. Specifically, we describe (1) loading RL data and initializing the ART-F model according to client-specific configurations; (2) configuring parallel client requests at the inference phase; and (3) supporting multiple reward formulations, with particular emphasis on composite reward designs.

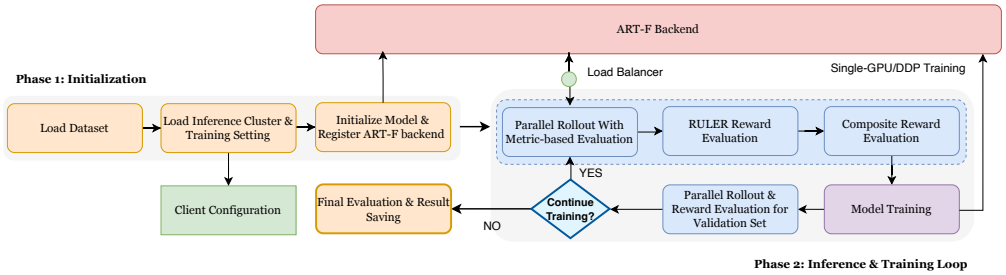


Figure 20: ART-F Client Workflow: The training client coordinates with the ART-F backend to manage parallel rollout requests through a load-balanced inference cluster, collect reward-annotated trajectories, and trigger single-GPU or DDP training, enabling efficient end-to-end reinforcement learning.

G.1 WORKFLOW

As shown in Figure 20, a typical ART-F client workflow covers two phases:

1) Initialization: The client first converts the raw dataset into rollout scenarios represented as serializable Python objects compatible with `Pydantic.BaseModel`, enabling them to be transmitted to the inference servers via HTTP requests. Dataset splitting into training, validation, and test sets is also performed at this stage. Next, the client loads a YAML-based configuration and merges it with the required command-line parameters to form a unified client configuration. This configuration is used to initialize the training job, register the model with the backend, and manage runtime behavior. For experiment tracking and reproducibility, the merged configuration is additionally logged to the corresponding Weights & Biases Weights & Biases (2025b) and MLflow Zaharia et al. (2018) projects.

2) Inference & Training Loop: Once the ART-F backend reaches the ready state, the client initiates parallel rollouts, with each inference server handling concurrent requests under semaphore-based rate limiting. During rollout, each scenario is executed T times as specified by the client configuration, and each resulting trajectory is evaluated using metric-based rewards and tagged with the corresponding scores. If enabled, RULER-based reward evaluation is applied during rollout.

Composite rewards are then computed, including **hybrid rewards** that combine RAC and RULER scores with fixed or dynamically scheduled weights, as well as **adaptive rewards** that emphasize different objectives (e.g., exploration, exploitation, or reliability) based on training progress. After reward processing, the client transitions to the training phase, where either single-GPU or distributed data-parallel (DDP) training optimizes the model using the collected trajectories. A validation rollout following the same procedure is scheduled after training.

Throughout the loop, comprehensive monitoring tracks training metrics, reward distributions, server utilization, and performance characteristics, with results logged to both Weights & Biases Weights & Biases (2025b) and MLflow Zaharia et al. (2018) projects according to the project setting. The aforementioned loop continues until the predefined epoch limit is reached.

G.2 CLIENT CONFIGURATION

The ART-F client is initialized using a YAML configuration file that specifies all runtime settings for both inference and training, as illustrated in Figure 21. This configuration governs logging and experiment tracking (e.g., Weights & Biases Weights & Biases (2025b) and MLflow Zaharia et al.

(2018)), rollout behavior and reward setting, inference cluster scaling and load-balancing policies, distributed data-parallel (DDP) PyTorch Team (2025) training parameters, and prompt templates for action prediction. By externalizing these components into a declarative configuration, ART-F enables flexible experimentation and rapid system reconfiguration without modifying the client source code.

```yaml
# Enhanced configuration for ART-F
logging:
  wandb:
    enabled: true
    project: "<Your project Name>"
    model_name: "<Your Model Name>"
    weave_trajectory_tag: "<Trajectory Tag on Weave>"
    weave_enabled: false
  mlflow:
    enabled: true
    local_root: "./.mlruns"
# LLM Settings
llm:
  model: "Qwen/Qwen2.5-14B-Instruct"
  max_tokens: 9216
  rollout:
    sample_num_per_training_scenario: 4
    sample_num_per_validation_scenario: 2
    temperature: 1.0
    messages: "system.user.only"
  # Reward setting
  trajectory_reward_rule: "schedule_ruler_weighted_max_rac_score"
  upper_limit_weight_for_scheduled_ruler: 0.3
  judger:
    model: "openai/gpt-4.1-mini"
    api_key_name: <INTERNAL_API_KEY>
    base_url: <INTERNAL_SERVER_BASE_URL>
    custom_ruler_placeholder:
      - <Custom RULER>
# Training Configuration
ddp_training:
  enable_ddp: true
  world_size: 4
  ddp_backend: "nccl"
  ddp_find_unused_parameters: false
  master_addr: ${HOST_IP}
  master_port: "29500"
  ddp_timeout_minutes: 30
  batch_size_allow_adjusting: true
  replicate_dataset_across_ranks: true
# Inference Configuration
art_multi_server:
  trainer_args:
    training_batch_size: 4
    logprob_calculation_chunk_size: 1024
    max_negative_advantage_importance_sampling_weight: 10
  server_pool:
    max_servers_per_model: 8
    concurrent_startup: true
    port_range_start: 8000
    port_range_end: 8100
    gpu_memory_per_server: 0.4
    scaling_policy:
      enable_auto_scaling: true
      scale_up_before_rollout: true
      scale_down_before_training: true
      scaling_timeout: 30
  rollout:
    enable_concurrent_rollouts: true
    rollout_batch_size: 24
    load_balancing_strategy: "round_robin"
# Rollout Prompts
tool_catalog:
  package: "Proactive_Benchmark.synthetic_data.proactive_annotation.abcd.tool_catalog_utils"
  tool_definition_cls: "COMMON_ABCD_ANNOTATED_TOOLING_API"

action_prompts:
  system_prompt: |
    You are an expert AI assistant for professional, specializing in analyzing dialogues to identify
      automation opportunities and their parameters based on the client and professional conversations.

    Given the conversation history and available system action opportunities, your task is to analyze the
      latest turn and determine the following for all relevant actions:
    1. Action opportunities that could be triggered once ready
    ......

  task_prompt: |
    ## CONVERSATION AT TURN {turn_idx}
    {dialogue_context}
```

Figure 21: ART-F Client YAML Configuration

In this section, we will discuss the key settings for inference cluster and DDP training. Details of the rollout prompt design are discussed in Section G.3, while the reward formulation and client setup are described in Section G.4.

LLM Rollout Setting To reduce rollout resource consumption, we support asymmetric sampling settings for training and validation during rollout, configured via `sample_num_per_training_scenario` and `sample_num_per_validation_scenario`.

Inference Cluster Setting The `art_multi_server.server_pool` configuration specifies inference cluster parameters, including `max_servers_per_model`, which defines the maximum number of inference server instances that can be launched for a given model, and `gpu_memory_per_server`, which controls the fraction of GPU memory reserved by each inference server on a single GPU. For instance, our LoRA-trained ProActor-Q4 model (9,216-token context) requires roughly 45 GB per inference server to support 15 concurrent rollout clients. On 4×H200 GPUs (141 GB each), allocating `gpu_memory_per_server`=0.4 enables two inference servers per GPU, allowing `max_servers_per_model` to be safely set to 8. Parameters `port_range_start` and `port_range_end` allow us to specify the dedicated port range for the inference cluster, while enabling `concurrent_startup` allows you to start the inference cluster in parallel with the minimal Unsloth process locking time.

During inference cluster startup, ART-F resolves the server bind address by prioritizing `VLLM_HOST_IP`, then `HOST_IP`, and finally falling back to `0.0.0.0`. This design is motivated by deployment environments such as Azure and Databricks, where binding to the default local or loopback address may fail.

Training Settings General training parameters are specified under `art_multi_server.trainer_args`, which defines settings shared by both single-GPU and DDP training, such as `training_batch_size` and `logprob_calculation_chunk_size`. Distributed data-parallel (DDP)–specific configurations are isolated in a dedicated `ddp_training` section. This section includes the DDP enable switch (`enable_ddp`), standard PyTorch-style DDP parameters, support for dynamic batch size adjustment via `batch_size_allow_adjusting`, and a symmetric data replication option (`replicate_dataset_across_ranks`).

Similar to the bind-address resolution used during inference cluster initialization, ART-F also supports environment variable macros in `ddp_training.master_addr`, allowing the runtime to identify the IP address dynamically and bind with the DDP process group under heterogeneous or managed cloud environments.

G.3 ROLLOUT

Parallelization Setting When `enable_concurrent_rollouts` is enabled, the ART-F client will launch `rollout_batch_size` parallel rollout groups for each scenario. Each rollout request is routed through the load balancer and assigned to an appropriate inference server according to the specified `load_balancing_strategy`, with fault recovery mechanisms to prevent bottlenecks on individual servers.

Rollout Prompt For each scenario, the rollout client constructs inference requests by using `action_prompts.system_prompt` as the system message and injecting the current-turn dialogue context into `action_prompts.task_prompt` as the user message. The resulting prompt pair is sent to the assigned LLM inference server to generate corresponding action opportunities or questions. Along with the dialogue context, these messages form a dialogue-style rollout trajectory composed of interleaved SYSTEM and USER message segments.

ART-F supports three configurable message-construction strategies, specified via `llm.rollout.messages`, to accommodate both single-turn and multi-turn rollout scenarios:

1. `first.last` — Designed for multi-turn rollouts, this setting constructs the request using only the first and the last messages from the trajectory scenario, enabling lightweight context compression while preserving long-range intent.

2. `track` — Also intended for multi-turn rollouts, this setting includes the full sequence of messages from the trajectory scenario, providing the model with complete conversational context.
3. `system.user.only` — Used for single-turn rollouts, this setting restricts the request to the system and user messages only, omitting historical turns to minimize prompt length.

`tool_catalog` is also injected into the prompt as the tooling specification. Considering that **the standard MCP tooling call** Anthropic (2024) doesn't support partial parameter mapping, we send requests to the inference server as **a normal message request**, and save the response to the "Assistant" section with "tool_call" mark in the returned response, yet **manually map the response body to the MCP-style action call** according to the action catalog specification.

Based on the `llm.rollout` configuration, each training scenario is rolled out `sample_num_per_training_scenario` times, while each validation scenario is rolled out `sample_num_per_validation_scenario` times. All rollout trajectories generated from the same scenario are grouped into a rollout trajectory group, becoming the data used for RL training.

Metadata-Derived Metrics at Post-Rollout As the metadata (dialogue information, reference actions, reference action range) has already been included in the rollout request body, **metadata-derived metrics**- `max_rac_score`, `rac_score`, `timing_reward_in_oracle`, and `fault_trigger_rate_in_oracle` - are computed immediately after rollout. Their values are then stored back in the metadata fields of the generated rollout trajectories. Meanwhile, ART-F automatically collects these metrics and reports them to the configured experiment-tracking backends.

G.4 Reward Calculation

If `trajectory_reward_rule` specifies **a metadata-derived metric**, rewards are computed deterministically and stored in rollout metadata immediately after generation. By contrast, RULER-based and composite rewards require additional coordination and computation. We next describe how ART-F supports both reward categories in the client.

RULER-based rewards If `trajectory_reward_rule` is set to RULER-based rewards as "ruler", "hybrid_ruler_weighted_rac_score", "hybrid_ruler_weighted_max_rac_score", "schedule_ruler_weighted_rac_score", "schedule_ruler_weighted_max_rac_score", the collected rollout trajectories group will be further processed by the LLM judger to rate each trajectory in one comparison group based on the pre-defined rubric.

The `judger` configuration specifies parameters for LLM-based evaluation, including `model`, `api_key_name`, `base_url`, and an optional `custom_ruler_placeholder`. If no custom rules are provided, ART-F uses the default RULER rubric inherited from ART, referred to as **General RULER** (Section 3.3). When custom rules are specified, they are appended to the judger prompt as additional evaluation criteria, yielding **Custom RULER** (Section 3.3).

Figure 22 shows our criteria for proactiveness evaluation. From the experimental results, we clearly see that incorporating the proactiveness RULER can significantly improve the agent's behavior for proactive timing. ART-F framework takes custom RULER and merges with other rubrics, and from the completed prompt, as shown in Figure 23, to evaluate a rollout trajectory group.

The Figure 24 shows how a sample rating in the JSON format from the LLM judger for the evaluation on a rollout group with four different trajectories.

Composite Rewards When both metadata-derived metrics and RULER-based metrics are available, ART-F supports composing them into *composite rewards* according to user-defined reward formulations. Broadly, we categorize composite rewards into two classes:

1. **Weighted rewards.** This category combines multiple metrics using **fixed weighting coefficients specified in the client configuration**. The weighting scheme

```
# ...Other Sections...

judger:
  model: "openai/gpt-4.1-mini"
  api_key_name: <INTERNAL_API_KEY>
  base_url: <INTERNAL_SERVER_BASE_URL>
  custom_ruler_placeholder:
    - |
      Given the dialogue conversation, a trajectory with timely, context-aware actions should receive
      significantly higher scores than one that is delayed or misaligned with the conversation.

      Specifically,
      1. Reward early and correct triggering: Give bonus credit to trajectories that mark appropriate ready-
      to-trigger or triggered action opportunities at the earliest relevant dialogue turn in the conversation
      . The reward diminishes as they are delayed.

      2. Penalize inaccuracy: Deduct scores for trajectories that incorrectly assign ready-to-trigger or
      triggered status (false positives), or that fail to assign ready-to-trigger or triggered status when
      valid opportunities exist (false negatives).

# ...Other Sections...
```

Figure 22: Our Custom RULER Setting for proactiveness evaluation

```
<system message>
All of the trajectories below have been given the same goal. Your job is to
consider each of them and assign a score between 0 and 1 based on your
best judgment of how well the agent achieves its goal.
Grading standards:
- A trajectory that achieves its goal should always receive a significantly
  higher score than one that does not.
- A trajectory that achieves its goal more efficiently (e.g., avoiding
  unproductive detours) should receive a higher score.
- If one trajectory is only slightly better than another, the score
  difference should be small; if significantly better, the difference
  should be large.
- << Injected Custom RULER >>.
- Partial credit may be given for trajectories that make progress toward
  the goal but do not complete it.
</system message>
<user message>
<context>
[
  {"content": "<< Rollout System Message >>", "role": "user"},
  {"content": "<< Rollout User Message >>", "role": "user"}
]
</context>
Trajectories:
<trajectory id="1">
[
  {"role": "assistant", "content": "<think><< ACTION Decision Explanation >></think>\n<action><< ACTION >></
      action>"}
]
</trajectory>
<trajectory id="2">
[
  {"role": "assistant", "content": "<think><< ACTION Decision Explanation >></think>\n<action><< ACTION >></
      action>"}
]
</trajectory>
<trajectory id="3">
[
  {"role": "assistant", "content": "<think><< ACTION Decision Explanation >></think>\n<action><< ACTION >></
      action>"}
]
</trajectory>
... If more trajectories are present.
```

Figure 23: LLM Judger Prompt for Trajectory-Level Proactiveness Evaluation with Custom RULER Injection

is independent of training progress and does not require access to the current or total number of training steps, including `weighted_max_rac_score`, `weighted_rac_score`, `hybrid_ruler_weighted_rac_score`, and `hybrid_ruler_weighted_max_rac_score`.

2. **Stage-aware weighted rewards.** In contrast, this category explicitly incorporates training progress $\frac{u}{U}$ into reward computation by adapting metric weights according to the current training step u relative to the total number of steps U, including `adaptive_metric_score`, `schedule_ruler_weighted_max_rac_score`, and `schedule_ruler_weighted_rac_score`. Detailed formulations are provided in Equations 10 and 11.

```
{
  "scores": [
    {
      "trajectory_id": "1",
      "explanation": "<explanation for rating>",
      "score": 0.95
    },
    {
      "trajectory_id": "2",
      "explanation": "<explanation for rating>",
      "score": 0.90
    },
    {
      "trajectory_id": "3",
      "explanation": "<explanation for rating>",
      "score": 0.85
    },
    ...
    {
      "trajectory_id": "N",
      "explanation": "<explanation for rating>",
      "score": 0.30
    }
  ]
}
```

Figure 24: JSON Response Format Returned by the LLM Judger for Proactiveness Evaluation

H RL METHODOLOGY DETAILS

This section provides additional mathematical formalization for the RL approach described in Section 3.3, including both turn-level and trajectory-level considerations.

H.1 TASK FORMULATION

Given a conversational dataset $\mathbf{D}$, each dialogue $D[i]$ consists of $N = |D[i]|$ annotated turns:

$$D[i,t] = \big(C[i,t],\, A^{\text{ref}}_{i,t}\big), \quad t \in \{1,\dots,N\} \tag{22}$$

where $C[i,t]$ denotes the dialogue content at turn t, and $A^{\text{ref}}_{i,t}$ is the subset of reference actions annotated for that turn.

Let $\tau[i,\leq t] = \{C[i,1],\dots,C[i,t]\}$ denote the dialogue trajectory up to turn t. Given an unified action catalog $\mathcal{A}$ and a reward function R, the model defines a stochastic policy π_θ that conditions on $\tau[i,\leq t]$ and samples a set of action candidates

$$\hat{A}_{i,t} = \{\hat{a}^{(k)}_{i,t}\}_{k=1}^{K}, \quad \hat{a}^{(k)}_{i,t} \sim \pi_\theta(\cdot \mid \tau[i,\leq t]) \tag{23}$$

The objective is to optimize π_θ to maximize the expected reward comparing predicted and reference action sets:

$$\max_\theta \; \mathbb{E}_{i,t}\Big[R\Big(\hat{A}_{i,t},\, A^{\text{ref}}_{i,t} \mid \tau[i,\leq t]\Big)\Big] \tag{24}$$

Here the notation is a little bit different from Section 3.2 and Section 3.3, where we directly use $\hat{A}_t$ and $\hat{A}^{\text{ref}}_t$ at the dialogue turn t and simply omits the dialogue index i, as it is clear from context.

H.2 METRIC-BASED REWARDS

The trajectory-level accumulated RAC reward up to turn t is defined as

$$\text{RAC}(\tau[i,\leq t]) = \frac{1}{t}\sum_{t'=1}^{t} \text{AC}\Big(\hat{A}_{i,t'},\, A^{\text{ref}}_{i,t'}\Big) \tag{25}$$

where $\hat{A}_{i,t'}$ and $A^{\text{ref}}_{i,t'}$ denote the predicted and reference action sets at turn t'.

The corresponding turn-level RAC reward is

$$\text{RAC}(\tau[i,t]) = \text{AC}\Big(\hat{A}_{i,t},\, A^{\text{ref}}_{i,t}\Big) \tag{26}$$

Similarly, the trajectory-level Max RAC reward up to turn t is

$$\text{Max RAC}(\tau[i, \leq t]) = \frac{1}{t} \sum_{t'=1}^{t} \max_{\hat{a} \in \hat{A}_{i,t'}} \text{AC}\big(\{\hat{a}\},\, A^{\text{ref}}_{i,t'}\big) \tag{27}$$

which selects the highest-consistency action within the predicted set at each turn.

The corresponding turn-level Max RAC reward is

$$\text{Max RAC}(\tau[i, t]) = \max_{\hat{a} \in \hat{A}_{i,t}} \text{AC}\big(\{\hat{a}\},\, A^{\text{ref}}_{i,t}\big) \tag{28}$$

H.3 RULER-Based Rewards

RULER (Relative Universal LLM-Elicited Rewards) OpenPipe (2025) evaluates the quality of predicted action candidates using rubric-based LLM judgments. Given a dialogue trajectory $\tau[i, \leq t]$, the trajectory-level accumulated RULER reward is defined as

$$\text{RULER}(\tau[i, \leq t]) = \frac{1}{t} \sum_{t'=1}^{t} \text{Judger}\Big(\hat{A}_{i,t'},\, P,\, \tau[i, \leq t']\Big) \tag{29}$$

where P denotes the instruction prompt used by the LLM-based evaluator.

The corresponding turn-level RULER reward is

$$\text{RULER}(\tau[i, t]) = \text{Judger}\Big(\hat{A}_{i,t},\, P,\, \tau[i, \leq t]\Big) \tag{30}$$

To capture proactiveness nuances not fully specified by P, we further introduce a set of custom rubric rules C, yielding the turn-level Custom RULER reward:

$$\text{CUSTOM RULER}(\tau[i, t]) = \text{Judger}\Big(\hat{A}_{i,t},\, P,\, C,\, \tau[i, \leq t]\Big) \tag{31}$$

Similarly, a trajectory-level custom ruler can be defined as:

$$\text{CUSTOM RULER}(\tau[i, \leq t]) = \frac{1}{t} \sum_{t'=1}^{t} \text{Judger}\Big(\hat{A}_{i,t'},\, P,\, C,\, \tau[i, \leq t']\Big) \tag{32}$$

From these formulations, we observe that preliminary trajectory-level reward designs fail to capture fine-grained action dynamics along rollout trajectories and therefore are ill-suited for providing dense reward signals to optimize RL models. However, **hierarchical reward formulations that combine turn-level and trajectory-level signals can better balance reward density and learning efficiency**, as demonstrated in prior work on hierarchical and temporally abstract reinforcement learning Zhou et al. (2024b); Ji et al. (2024); Wientjes & Holroyd (2024).

H.4 Turn-Level GRPO Formulation

For each turn t in conversation c, we sample K action candidates and compute turn-specific rewards. At the optimization step u, the policy gradient update is:

$$\nabla_\theta \mathcal{L}_{\text{cap-clip}} = \mathbb{E}_{u,c}\left[\sum_{k=1}^{K} g_u^{(k)}\, \nabla_\theta \log \pi_\theta(a_u^{(k)} | s_u)\right] \tag{33}$$

$$g_u^{(k)} = \begin{cases} \bar{r}_u^{(k)} A_u^{(k)}, & \text{if } \bar{r}_u^{(k)} A_u^{(k)} \leq \text{clip}(\bar{r}_u^{(k)}, 1-\epsilon, 1+\epsilon_{\text{high}}) A_u^{(k)}, \\ \text{clip}(\bar{r}_u^{(k)}, 1-\epsilon, 1+\epsilon_{\text{high}}) A_u^{(k)}, & \text{otherwise.} \end{cases} \tag{34}$$

where c denotes the episode-level context (e.g., dialogue history and task configuration) sampled from the training distribution, $A_u^{(k)} \neq 0$ are computed from turn-level rewards rather than trajectory returns, and $\bar{r}_u^{(k)}$ denotes the importance sampling ratio cap ($\bar{r}_u^{(k)} = 10$, parameter tuning via performance consistency check, refer to Section M.2.1).

I DATASET DETAILS

This section provides additional details on the datasets used for training and evaluation. We create two datasets representing the two general types encountered in real-world task scheduling: (1) systems *with* historical action execution logs, where observed triggers enable alignment validation, and (2) systems *without* such logs, where only conversation transcripts are available. This design validates that our framework generalizes across both scenarios.

I.1 ABCD+ DATASET DETAILS

We enhance ABCD (Chen et al., 2021) with proactive annotations. The original data contains 10,042 task-oriented dialogues with 30 actions across customer service scenarios. Critically, our oracle annotation operates purely from conversation context—observed triggers are *not* required for the annotation process itself, enabling the same pipeline to work across domains. However, when observed triggers are available, they enable post-annotation quality validation: we measure alignment between oracle annotations and actual system executions, achieving 87.14% alignment. After filtering dialogues with quality scores > 0.8, we retain approximately 70% of the data—7,042 dialogues (5,647/703/692 train/dev/test) with 114,978 proactive action annotations. Validation details are in Appendix D.

Quality Validation. We validate annotation quality using the reference alignment validation D.3.1. For each triggered action, we compute Early Ready Criteria (EC)—whether oracle annotations predict actions before/at trigger time, as mentioned in Section D.3.2. Our validation on the full dataset shows 87.14% EC success rate, with 70.27% of dialogues achieving quality scores > 0.8.

ABCD+ Statistics. The enhanced dataset contains 114,978 proactive action annotations with 80.89% coverage of triggered actions. Each dialogue averages 3.6 actions with 5.98 ± 2.86 valid reference ranges. For high-quality training data, we filter dialogues with oracle scores > 0.8, yielding 7,042 dialogues (5,647/703/692 train/dev/test). This filtered dataset serves as our primary benchmark for proactive dialogue agent evaluation.

I.2 HOME LOAN DATASET DETAILS

To demonstrate generalizability, we apply our pipeline to real mortgage consultations—phone call transcripts between loan officers and clients in a financial services domain, distinct from ABCD's customer support context. This dataset contains only conversational data without any observed software action triggers. The dataset covers 13 action types across 6 workflow categories. Domain adaptation requires *only configuration changes*—no code modifications—validating our pipeline's domain-agnostic design.

Challenges Without Action Observations. Without recorded action triggers, our LLM-based oracle must infer appropriate action timing purely from conversational context. We address this through enhanced annotation guidelines: (1) conservative triggering requiring clear evidence of readiness, (2) strict parameter tracking to ensure all required inputs are present before marking actions as ready, and (3) multi-level readiness assessment considering both client intent and information completeness.

Privacy and Infrastructure. Due to stringent privacy requirements, all data processing is conducted within secure environments with limited GPU resources. These constraints reflect realistic enterprise settings and highlight our framework's ability to operate effectively while preserving annotation quality. Moreover, the lack of observed triggers, together with strict privacy restrictions, makes this dataset particularly challenging. Despite these limitations, our automated annotation pipeline can be readily configured to identify proactive opportunities throughout mortgage consultations without additional coding, enabling effective automation of complex workflows even in the absence of historical action data.

J TRAINING SETUP

ABCD+ Configuration. On our internal platform, we use a 4×H200 GPU server (each GPU with 141 GB memory) to conduct RL training on the ABCD+ dataset. For the inference server cluster, ART-F launches 8 inference instances, each configured to use 40% of a single GPU's memory. To match the cluster's throughput, we allocate 100–230 parallel rollout groups with training rollout sizes of 4 and validation rollout sizes 2, which fully saturates GPU and process capacity while maintaining stable performance. For training, we set the model's maximum sequence length to 9,216 tokens and use a maximum training batch size of 4 , with three parallel data-loading workers. We enable dynamic batch sizing, which automatically adjusts the effective batch size to respect a fixed training budget of 4 × 9,216 tokens per step. To ensure training stability and prevent memory over-utilization, we cap GPU utilization at 90% per device. We further enable Distributed Data Parallel (DDP) training `enable_ddp` and symmetric data mode `replicate_dataset_across_ranks`.

Home Loan Configuration. On Databricks, we set up 8×H100 GPU servers to do RL training on the Home Loan Dataset. Each of the GPUs has 80 GB of memory. Considering more GPU numbers while having less memory on each node, we reduce the training batch size to 2–3 with a maximum token length of 9,216, while using 8 inference instances, each configured to use 45% of a single GPU's memory. Similarly, we also enable Distributed Data Parallel (DDP) training `enable_ddp` and symmetric data mode `replicate_dataset_across_ranks`.

LoRA Setting We use LoRA with rank r=8 and scaling factor α=16, applied to attention (q, k, v, o) and MLP $(\text{gate}, \text{up}, \text{down})$ projections, with dropout disabled for maximal throughput, which aligns with the default ART-F PEFT setting as shown in Figure 25.

```
# LoRA configuration used in ART-F

{
  "max_lora_rank": 8,
  "lora_alpha": 16,
  "lora_dropout": 0,
  "target_modules": [
    "q_proj", "k_proj", "v_proj", "o_proj",
    "gate_proj", "up_proj", "down_proj"
  ]
}
```

Figure 25: LoRA configuration used in ART-F.

Inference Time Scaling. Table 4 reports inference-time scaling under different rollout configurations. ART-F achieves near-linear inference speedups by scaling inference servers to match client request throughput across multiple GPUs, effectively mitigating GPU under-utilization and addressing scalability limitations.

K EXTENDED EXPERIMENTAL ANALYSIS

This section provides additional analyses of baseline performance and reinforcement learning improvements that were summarized in the main text. We also include GPT-4.1-mini, an early baseline, to complete the comparative evaluation.

K.1 RATIONALE

Proactive agents must master three interconnected capabilities: *what* action to take, *how* to parameterize it correctly, and *when* to execute it. Traditional task-oriented dialogue evaluation focuses almost exclusively on the first two—measuring whether the agent selects correct actions with accurate parameters. However, for proactive automation, *timing* is equally critical: an action predicted too late provides no proactive value, while one predicted too early may lack sufficient information for correct parameterization. We therefore organize our evaluation into two categories:

	PRI ↑	Consistency			Timing		
		AC ↑	Max AC ↑	Difference ↓	Proactive Timing ↑	Fault Trigger Rate ↓	Ready Action Rate ↑
ABCD+							
GPT-4.1-mini Non-Reasoning	0.4405	0.363±0.097	0.524±0.040	0.363±0.097	0.0989±0.0703	0.0080±0.0060	0.161±0.115
GPT-4.1-mini Reasoning	0.3986	0.315±0.111	0.550±0.036	0.315±0.111	0.0587±0.0766	**0.0044±0.0062**	0.214±0.003
GPT-4.1-mini Reasoning + ASG	0.4092	0.303±0.072	0.537±0.058	0.303±0.072	0.0625±0.0864	**0.0067±0.0095**	0.281±0.003
GPT-5.1 Non-Reasoning	0.5104	0.318±0.005	0.706±0.012	0.717±0.013	0.2023±0.0019	0.0460±0.0009	0.419±0.010
GPT-5.1 Reasoning	0.6003	0.429±0.006	0.789±0.002	0.839±0.026	0.1643±0.0018	0.0165±0.0002	0.214±0.003
GPT-5.1 Reasoning + ASG	0.5547	0.420±0.004	0.733±0.007	0.712±0.063	0.1874±0.0022	0.0647±0.0019	0.281±0.003
Gemini-2.5-flash Non-Reasoning [4]	**0.6251**	0.417±0.004	**0.834±0.001**	1.000±0.019	0.2133±0.0030	0.0399±0.0006	0.288±0.003
Gemini-2.5-flash Reasoning	0.6216	**0.430±0.001**	0.813±0.002	0.891±0.006	0.1782±0.0011	0.0245±0.0006	0.252±0.001
Gemini-2.5-flash Reasoning + ASG	0.5257	0.384±0.001	0.737±0.001	0.919±0.006	0.1715±0.0015	0.0286±0.0005	0.241±0.004
Claude-4 Non-Reasoning	0.6216	0.427±0.001	0.831±0.001	0.946±0.005	0.2119±0.0013	0.0526±0.0012	0.297±0.001
Claude-4 Reasoning [3]	**0.6318**	0.421±0.001	0.749±0.004	0.779±0.010	0.2136±0.0023	0.0482±0.0023	0.314±0.003
Claude-4 Reasoning + ASG	0.6031	0.403±0.005	**0.852±0.004**	1.114±0.028	0.2269±0.0022	0.0634±0.0032	0.360±0.002
Qwen2.5-14B-Instruct Non-Reasoning	0.2996	0.295±0.004	0.564±0.002	0.914±0.004	0.2237±0.0012	0.0773±0.0010	0.515±0.002
Qwen2.5-14B-Instruct Reasoning	0.6246	0.423±0.005	0.449±0.006	0.062±0.019	0.1842±0.0012	0.0307±0.0008	0.279±0.003
Qwen2.5-14B-Instruct Reasoning + ASG	0.4331	0.316±0.004	0.385±0.004	0.218±0.020	0.1918±0.0043	0.0432±0.0011	0.359±0.004
Qwen2.5-14B-ProActor-Q4 + Custom RULER [1]	**0.7293**	0.426±0.015	0.484±0.019	0.136±0.048	**0.2347±0.0201**	0.0708±0.0078	**0.546±0.036**
Qwen2.5-14B-ProActor-Q4 + Adaptive RULER [2]	**0.6842**	**0.431±0.022**	0.586±0.044	0.320±0.123	**0.2515±0.0272**	0.1089±0.0083	**0.521±0.052**
Home Loan							
GPT-4.1-mini Non-Reasoning	0.4835	0.237±0.011	0.265±0.010	0.118±0.067	0.0219±0.0015	0.0009±0.0002	0.074±0.006
GPT-4.1-mini Reasoning	0.4882	0.300±0.010	0.329±0.013	0.097±0.057	0.0118±0.0026	**0.0003±0.0002**	0.057±0.002
GPT-4.1-mini Reasoning + ASG	0.4652	0.188±0.002	0.232±0.005	0.234±0.030	0.0281±0.0098	0.0012±0.0010	0.151±0.013
GPT-5.1 Non-Reasoning	0.5047	0.272±0.001	0.467±0.003	0.717±0.013	0.0462±0.0013	0.0049±0.0002	0.116±0.003
GPT-5.1 Reasoning	0.5047	0.363±0.007	**0.579±0.004**	0.595±0.033	0.0186±0.0026	0.0003±0.0003	0.031±0.002
GPT-5.1 Reasoning + ASG	0.5010	0.281±0.007	0.481±0.013	0.712±0.063	0.0374±0.0061	0.0034±0.0004	0.105±0.010
Gemini-2.5-flash Non-Reasoning	0.6165	0.349±0.004	**0.688±0.004**	0.972±0.024	0.0632±0.0011	0.0072±0.0005	0.173±0.005
Gemini-2.5-flash Reasoning [1]	**0.7303**	0.345±0.004	0.527±0.006	0.528±0.024	0.0757±0.0011	**0.0001±0.0001**	0.241±0.002
Gemini-2.5-flash Reasoning + ASG	0.6067	0.283±0.007	0.479±0.002	0.693±0.042	0.0764±0.0039	0.0063±0.0018	0.251±0.012
Claude-4 Non-Reasoning	0.5416	0.224±0.003	0.347±0.004	0.549±0.024	**0.0811±0.0029**	0.0118±0.0008	**0.332±0.002**
Claude-4 Reasoning [3]	**0.7039**	0.332±0.006	0.472±0.003	0.422±0.028	0.0742±0.0027	0.0063±0.0016	0.261±0.003
Claude-4 Reasoning + ASG [2]	**0.7262**	**0.375±0.002**	0.607±0.006	0.619±0.022	0.0760±0.0031	0.0127±0.0013	0.307±0.005
Qwen2.5-14B-Instruct Non-Reasoning	0.4288	0.253±0.002	0.383±0.003	0.514±0.017	0.0120±0.0006	0.0003±0.0004	0.054±0.005
Qwen2.5-14B-Instruct Reasoning	0.4012	0.217±0.005	0.239±0.009	0.101±0.049	0.0037±0.0010	0.0006±0.0001	0.032±0.003
Qwen2.5-14B-Instruct Reasoning + ASG	0.3223	0.113±0.011	0.144±0.006	0.274±0.133	0.0184±0.0083	0.0033±0.0022	0.090±0.014
Qwen2.5-14B-ProActor-Q4 + Custom RULER	0.5603	0.206±0.024	0.234±0.021	0.137±0.161	**0.0846±0.0095**	0.0355±0.0079	**0.465±0.074**
Qwen2.5-14B-ProActor-Q4 + Adaptive RULER [4]	**0.6232**	**0.395±0.029**	0.466±0.038	0.180±0.129	0.0501±0.0055	0.0131±0.0013	0.156±0.009

Table 5: Model performance comparison: Baselines are averaged over 3 runs, while Qwen2.5-14B-ProActor-Q4 are aggregated by last $N = 4$ steps. Adaptive RULER is using Max RAC with $\lambda_{\max} = 0.3$. [1], [2], [3], [4] mark top-1, top-2, top-3, top-4 methods by PRI (Proactiveness Ranking Index, see Section K.4).

- **Action Consistency** (AC, Max AC, Consistency Difference) measures *what*, *how well*, and *how reliably*. AC averages consistency across all predictions, while Max AC captures the best single-action quality per turn, accommodating agents that track multiple hypotheses. Consistency Difference quantifies prediction reliability—the relative gap between best-case and average performance. A high Difference indicates that the model occasionally "gets lucky" with one good action but is unreliable on average—a signature of guesswork rather than genuine understanding. Low Difference signals consistent, trustworthy predictions suitable for production deployment.
- **Timing** (PT, FTR, RAR) measures *when* and *how aggressively*—whether the agent predicts action readiness at appropriate moments. Proactive Timing (PT) rewards predictions aligned with oracle readiness windows. Fault Trigger Rate (FTR) penalizes predictions of non-existent actions. Ready Action Rate (RAR) captures overall proactive engagement—the proportion of predictions marked as ready.

This design reveals the fundamental challenge of proactive agents: an agent optimizing purely for Action Consistency may become overly conservative, waiting until parameters are complete—achieving high precision but low proactiveness. Conversely, aggressive timing may sacrifice parameter accuracy and consistency. Effective proactive agents must balance both dimensions while maintaining reliable predictions, a challenge we find no baseline fully resolves.

K.2 Interpretive Principles

Before analyzing results, we highlight three critical interpretive principles:

On Action Consistency: AC measures alignment with oracle annotations, which represent **one valid timing choice among potentially many**. Oracle annotations are constructed with hindsight: annotators observe the complete conversation and label actions that *were* or *could have been* triggered. A genuinely proactive agent operating in real-time may identify opportunities *earlier* than the oracle, before all parameters are available—valid proactive behavior that nonetheless receives lower

AC scores. We complement AC with PT, which evaluates whether predictions *will become* ready ($\exists \tau \geq t$ in reference range) rather than requiring exact timing match.

On Consistency Difference: A high Consistency Difference (¿0.5) is a strong indicator of **unreliable, guess-based predictions**. For example, Qwen2.5-14B-Instruct Non-Reasoning achieves the highest Max AC (0.564) on ABCD+ but also the highest Consistency Difference (0.914), revealing that while it occasionally identifies good actions, its average predictions are poor. Similar cases also appear on Gemini-2.5-flash, Claude-4.0. In contrast, reasoning-based methods dramatically reduce Consistency Difference (Qwen2.5-14B-Instruct: 0.914→0.062), demonstrating that explicit reasoning improves prediction consistency, not just accuracy. For production deployment, a low or moderate Consistency Difference is essential—a model that is "sometimes right" is less valuable than one that is "consistently good."

On Fault Trigger Rate: For proactive agents, **lower FTR is not always better**. FTR≈0 often indicates that a model has abandoned proactiveness—achieving "precision" by rarely predicting actions as ready. The appropriate evaluation considers *net proactive value* (correct actions minus weighted faults) rather than minimizing FTR in isolation.

K.3 On Timing

True proactiveness requires the agent to reason beyond the current dialogue context and schedule action opportunities **as early as possible once the necessary conditions are confidently satisfied**. To assess this capability, we consider: (1) **Proactive Timing (PT)**, measuring the proportion of predicted ready actions that *will* become valid (true positive ready-action rate); (2) **Fault Trigger Rate (FTR)**, quantifying the proportion that *will not* become valid (false positive ready-action rate); and (3) **Ready Action Rate (RAR)**, capturing overall proactive engagement. Effective proactive agents should achieve strong PT and high RAR while keeping FTR within acceptable bounds. Formal definitions can be found in Section 3.2.

K.4 Proactiveness Ranking Index for Model Comparison

While individual proactive metrics (AC, PT, FTR, RAR) provide specific insights into model behavior, comparing models across different datasets and scales requires a unified evaluation framework. We introduce the **Proactiveness Ranking Index (PRI)**, a composite ranking metric that aggregates consistency and timing performance into a single comparable measure for systematic model ordering and selection.

K.4.1 Motivation and Design Principles

Proactive conversational agents must balance two critical aspects: *consistency* in action prediction and *timing* in proactive interventions. A model that achieves high action consistency (AC, Max_AC) but poor timing (high FTR, low PT) may be less effective than one with moderate consistency but excellent timing. The PRI metric addresses this by:

1. **Dimensional Reduction**: Aggregating multiple correlated metrics into interpretable indices
2. **Balanced Evaluation**: Using harmonic mean to prevent one dimension from dominating
3. **Relative Ranking**: Enabling fair comparison across different dataset scales and characteristics

K.4.2 Mathematical Formulation

Given a set of evaluation metrics $\mathbf{M} \equiv \{\text{AC}, \text{Max\_AC}, \text{Difference}, \text{PT}, \text{FTR}, \text{RAR}\}$ and a comparison group $G = \{M_1, M_2, \ldots, M_n\}$ evaluated on the same dataset, we compute PRI through the following steps:

Step 1: Metric Normalization For each metric $m \in \mathbf{M}$, we apply min-max normalization within group G:

$$m_{\text{norm}}^{(i)} = \frac{m^{(i)} - \min_{j \in G} m^{(j)}}{\max_{j \in G} m^{(j)} - \min_{j \in G} m^{(j)}} \tag{35}$$

where $m^{(i)}$ denotes the metric value for model M_i. When $\max = \min$, we set $m_{\text{norm}}^{(i)} = 0.5$.

Step 2: Index Computation We define two composite indices and assume each factor takes an equal weight:

Consistency Index (CI):

$$\text{CI}^{(i)} = \frac{\text{AC}_{\text{norm}}^{(i)} + \text{Max\_AC}_{\text{norm}}^{(i)} + (1 - \text{Diff}_{\text{norm}}^{(i)})}{3} \tag{36}$$

Timing Index (TI):

$$\text{TI}^{(i)} = \frac{\text{PT}_{\text{norm}}^{(i)} + (1 - \text{FTR}_{\text{norm}}^{(i)}) + \text{RAR}_{\text{norm}}^{(i)}}{3} \tag{37}$$

The CI measures action prediction consistency, where higher AC and Max_AC values and lower Difference values indicate better performance. The TI captures timing effectiveness, where higher PT and RAR values and lower FTR values indicate better proactive timing.

Step 3: Harmonic Mean Aggregation The final Proactiveness Ranking Index uses harmonic mean to ensure balanced performance:

$$\text{PRI}^{(i)} = \frac{2 \times \text{CI}^{(i)} \times \text{TI}^{(i)}}{\text{CI}^{(i)} + \text{TI}^{(i)}} \tag{38}$$

To ensure numerical stability, we apply epsilon protection: $\text{CI}^{(i)}, \text{TI}^{(i)} \geq \epsilon = 0.001$.

K.4.3 Interpretation and Properties

Index Range and Meaning PRI values range from 0 to 1, where:

- **PRI $\approx$ 1.0**: Excellent performance in both consistency and timing
- **PRI $\approx$ 0.5**: Moderate performance or strong in one dimension but weak in another
- **PRI $\approx$ 0.0**: Poor performance in at least one critical dimension

Harmonic Mean Properties The harmonic mean penalizes imbalanced performance more severely than arithmetic mean. For example:

- Balanced: CI=0.8, TI=0.8 $\Rightarrow$ PRI=0.8
- Imbalanced: CI=1.0, TI=0.6 $\Rightarrow$ PRI=0.75 (not 0.8)
- Severely imbalanced: CI=1.0, TI=0.1 $\Rightarrow$ PRI=0.18 (not 0.55)

This design encourages models that perform well across both dimensions rather than excelling in only one.

K.4.4 Usage in Model Comparison

We apply the proposed PRI to rank model performance in both the baseline comparison (Table 5) and the reward ablation study (Table 2).

Baseline and ProActor Comparison. We construct two comparison groups corresponding to the ABCD+ and Home Loan datasets. Each group includes all prompting-based baselines as well as two ProActor-Q4 variants, optimized using the Custom RULER and Adaptive RULER rewards, respectively.

ProActor Reward Variation. To analyze the impact of different reward designs, we construct three comparison groups: (1) a small-scale ABCD+ setting with 100 training and 50 test dialogues, used to isolate the effects of single-objective rewards; (2) the full ABCD+ dataset with 5,647 training and 692 test dialogues, used to evaluate all proposed reward variations at scale; and (3) the full Home Loan dataset with 774 training and 97 test dialogues, used to assess the same reward variations, including combinations of RAC and Max RAC with both the Weighted Metric and Adaptive RULER rewards.

K.4.5 Limitations and Considerations

Relative Nature PRI is a *relative ranking metric*, not an absolute performance measure. A model with PRI=0.8 in one group may perform differently than a PRI=0.8 model in another group. Cross-group comparisons should be interpreted with caution.

Normalization Sensitivity In small comparison groups, outlier models can skew normalization. We address this by:

1. Ensuring minimum group size of 3 models when possible
2. Reporting raw metric values alongside PRI scores
3. Documenting outlier cases (e.g., the ABCD+ 3000/600 performance degradation)

Metric Weight Assumptions The equal weighting of AC, Max_AC, and Difference in CI assumes these metrics have similar importance. Similarly, PT, FTR, and RAR receive equal weight in TI. Future work could explore learned or domain-specific weightings.

K.5 Baseline Patterns

(1) Reasoning improves consistency but not timing. Introducing explicit reasoning substantially reduces Consistency Difference (on average 19–20% for non-Qwen baselines, and a much larger 80–93% drop for Qwen), while its effect on AC and Max AC varies across models and settings. In contrast, reasoning does not reliably improve timing-related metrics: Proactive Timing frequently degrades, sometimes sharply (e.g., GPT-5.1 and Qwen on Home Loan), with only a few exceptions (Claude-4 on ABCD+ and Gemini-2.5-flash on Home Loan). These results suggest that while reasoning helps models produce actions more consistently, it does not reliably improve *when* actions should be taken.

Model	ΔAC	ΔMaxAC	ΔDifference	ΔPT	ΔFTR	ΔRAR
ABCD+ (Non-Reasoning → Reasoning)						
GPT-5.1	+34.9%	+11.8%	-31.2%	-18.8%	-64.1%	-48.9%
Gemini-2.5-flash	+3.1%	-2.5%	-10.9%	-16.5%	-38.6%	-12.5%
Claude-4	-1.4%	-9.9%	-17.7%	+0.8%	-8.4%	+5.7%
Qwen2.5-14B-Instruct	+43.4%	-20.4%	-93.3%	-17.7%	-60.3%	-45.8%
Home Loan (Non-Reasoning → Reasoning)						
GPT-5.1	+33.5%	+24.0%	-17.0%	-75.6%	-93.9%	-73.3%
Gemini-2.5-flash	-1.1%	-23.4%	-45.6%	+19.8%	-98.6%	+39.3%
Claude-4	+48.2%	+36.0%	-23.1%	-8.5%	-46.6%	-21.4%
Qwen2.5-14B-Instruct	-14.2%	-37.6%	-80.4%	-69.2%	+100.0%	-40.7%

Table 6: Relative change (%) from Non-Reasoning to Reasoning baselines: $\Delta_{\%} = \frac{\text{R}-\text{NR}}{\text{NR}} \times 100\%$. (Large percentage changes may occur when baseline values are close to zero.)

(2) ASG destabilizes reasoning policies with model-specific fault amplification. Adding ASG on top of reasoning baselines consistently amplifies fault risks and destabilizes consistency, with similar failure patterns observed across model families. For example, on ABCD+, **GPT-5.1** exhibits only modest gains in proactive timing (+14.1%) and ready action rate (+31.3%), but incurs a sharp

increase in Fault Trigger Rate (+292.1%) alongside declines in AC and Max AC. **Gemini-2.5-flash** shows no meaningful timing improvement (–3.8% PT) while still increasing FTR (+16.7%) and consistency difference (+3.1%). **Claude-4** improves Max AC (+13.7%), yet this comes at the cost of substantially higher inconsistency (+43.0% Difference) and elevated FTR (+31.5%). The degradation is most severe for **Qwen2.5-14B-Instruct**, where ASG causes a large consistency collapse (Difference +251.6%) and a significant rise in FTR (+40.7%) despite marginal timing gains.

A similar trend appears in the Home Loan domain, where ASG leads to extreme fault amplification across all models (e.g., FTR +1033% for GPT-5.1 and +6200% for Gemini-2.5-flash), even when proactive timing or action coverage improves. Together, these results indicate that ASG primarily increases action firing frequency without stabilizing decision quality, resulting in inconsistent and unsafe behavior when layered onto reasoning policies.

Model	ΔAC	ΔMaxAC	ΔDifference	ΔPT	ΔFTR	ΔRAR
ABCD+ (Reasoning → Reasoning + ASG)						
GPT-5.1	-2.1%	-7.1%	-15.1%	+14.1%	+292.1%	+31.3%
Gemini-2.5-flash	-10.7%	-9.3%	+3.1%	-3.8%	+16.7%	-4.4%
Claude-4	-4.3%	+13.7%	+43.0%	+6.2%	+31.5%	+14.6%
Qwen2.5-14B-Instruct	-25.3%	-14.3%	+251.6%	+4.1%	+40.7%	+28.7%
Home Loan (Reasoning → Reasoning + ASG)						
GPT-5.1	-22.6%	-16.9%	+19.7%	+101.1%	+1033.3%	+238.7%
Gemini-2.5-flash	-18.0%	-9.1%	+31.3%	+0.9%	+6200.0%	+4.1%
Claude-4	+13.0%	+28.6%	+46.7%	+2.4%	+101.6%	+17.6%
Qwen2.5-14B-Instruct	-47.9%	-39.7%	+171.3%	+397.3%	+450.0%	+181.3%

Table 7: Relative change (%) from Reasoning to Reasoning + ASG baselines: $\Delta_{\%} = \frac{\text{R+ASG}-\text{R}}{\text{R}} \times 100\%$. (Large percentage changes may occur when baseline values are close to zero.)

(3) No baseline resolves the consistency–timing trade-off. Across both ABCD+ and Home Loan, existing baselines fail to jointly achieve stable action selection and timely execution.

On **ABCD+**, strong baselines frequently attain high Max AC (up to 0.85), but this comes with substantially inflated Consistency Difference (typically 0.6–1.1), indicating unstable action selection. To control this instability, many models effectively behave conservatively, resulting in limited proactive timing and ready action rates (PT ≤ 0.22, RAR ≤ 0.32). While a few baselines achieve PT values closer to ProActor (e.g., Claude-4 and Gemini-2.5-flash), these gains are consistently accompanied by severe consistency degradation.

On **Home Loan**, the trade-off is more pronounced. Baselines that maintain lower Consistency Difference exhibit extremely conservative behavior, yielding very low proactive timing (typically PT < 0.08) and ready action rates (RAR < 0.33). Conversely, models that improve PT do so at the cost of sharply increased inconsistency, suggesting that timing gains arise from aggressive over-triggering rather than reliable temporal decision-making.

In contrast, ProActor-Q4 uniquely maintains low Consistency Difference (≈ 0.14) while simultaneously achieving the highest proactive timing and ready action rates across both datasets, demonstrating that existing baselines do not resolve the consistency–timing trade-off.

(4) Near-zero FTR on baselines reflects conservative failure, not precision. A model that never makes mistakes is often a model that never attempts to be proactive. This failure mode is most evident on Home Loan, where GPT-5.1 Reasoning achieves a high Max AC (0.579±0.004) but exhibits extremely low proactive behavior (PT=0.0186±0.0026, RAR=0.031±0.002) alongside a near-zero Fault Trigger Rate (FTR=0.0003±0.0003). Combined with its large Consistency Difference (0.595±0.033), this pattern reveals a model that can identify well-matched actions but systematically avoids committing to them. The resulting low FTR is therefore achieved through excessive conservatism rather than precise temporal decision-making.

This behavior recurs across conservative baselines on Home Loan, where near-zero FTR values (typically 0.01–0.3%) consistently coincide with severely diminished proactive capacity (RAR 3–16%). Similar observations occur on ABCD+, where reasoning-based baselines maintain low FTR (typically below 2%) yet fail to translate high Max AC into timely or reliable readiness decisions. Although these models occasionally act, their proactive timing and ready action rates remain capped (PT ≤ 0.17, RAR ≤ 0.32), indicating hesitation rather than confidence. Among baselines, **Gemini-2.5-flash** shows relatively stronger proactive timing (PT ≈ 0.21 on ABCD+ and up to 0.076 on Home Loan). However, this improvement comes with poor action consistency, reflected by a high Consistency Difference (often > 0.9) and unstable AC/Max AC alignment, suggesting that better timing is achieved at the expense of reliable action selection.

In contrast, ProActor-Q4 adopts a recall-oriented but controlled strategy across both datasets. On Home Loan, it achieves substantially higher readiness (RAR=46.5%) with a bounded FTR of 3.6%, producing nearly $3\times$ more correct ready actions than conservative baselines. These results suggest that minimizing FTR in isolation is a misleading objective; instead, effective proactive agents should optimize *net proactive value*, balancing correct ready actions against manageable fault costs.

(6) No prompting strategy resolves the precision–consistency–timing trade-off. Across both ABCD+ and Home Loan, even the strongest prompting-based baselines consistently optimize one objective at the expense of others. Non-reasoning variants of state-of-the-art models (e.g., Gemini-2.5-flash and Claude-4) achieve very high Max AC on ABCD+ (up to 0.83–0.85), but suffer from severe inconsistency, with Consistency Difference frequently exceeding 0.9 and unstable alignment between AC and Max AC.

Introducing explicit reasoning substantially improves action consistency and average AC across models, but leads to conservative decision policies that suppress proactive behavior. As a result, reasoning-based baselines exhibit low proactive timing and ready action rates (PT ≤ 0.17, RAR ≤ 0.32 on ABCD+; PT < 0.02, RAR ≈ 0.03 on Home Loan), despite maintaining strong Max AC.

Augmenting reasoning with ASG partially improves timing awareness, yielding the highest PT among prompting strategies, but does so at a high cost to reliability: AC and Max AC degrade, Consistency Difference increases, and fault trigger rates rise sharply. Consequently, no prompting configuration simultaneously achieves high action quality, low inconsistency, timely execution, and meaningful readiness under acceptable fault rates. These systematic limitations motivate our reinforcement learning approach, which learns adaptive policies that balance precision, consistency, and timing beyond the limits of static prompting.

K.6 Detailed ProActor-Q4 Analysis

In contrast, ProActor-Q4 models achieve a better balance between reference action alignment and proactive timing. They maintain performance comparable to strong baselines on action consistency, while substantially improving proactive timing and ready action rates, without incurring an unacceptable increase in fault trigger rates.

Q1: Does ProActor-Q4 preserve robustness in action consistency? ProActor-Q4 maintains **strong and well-balanced action consistency** across domains. We evaluate robustness using *Consistency Difference*, which measures the gap between average and best-case action alignment and reflects susceptibility to sporadic or "lucky" matches.

1. **ABCD+.** ProActor-Q4 + Custom RULER achieves the lowest Consistency Difference (0.136 ± 0.048) with solid alignment (AC $= 0.426 \pm 0.015$, Max AC $= 0.484 \pm 0.019$). The Adaptive RULER variant increases action quality substantially (AC $= 0.431 \pm 0.022$, Max AC $= 0.586 \pm 0.044$) while keeping the Consistency Difference at a moderate level (0.320 ± 0.123), far below those of strong prompting baselines such as Gemini-2.5-flash Non-Reasoning (Difference $= 1.000 \pm 0.019$) and Claude-4 Reasoning + ASG (Difference $= 1.114 \pm 0.028$). This indicates a favorable trade-off: Adaptive RULER sacrifices some consistency to obtain markedly stronger action alignment, without entering the unstable regime observed in prompting methods.
2. **Home Loan.** Under domain shift, Adaptive RULER exhibits an even clearer balance. Compared to Custom RULER (AC $= 0.206 \pm 0.024$, Max AC $= 0.234 \pm 0.021$, Difference

$= 0.137 \pm 0.161$), Adaptive RULER nearly doubles alignment quality (AC $= 0.395 \pm 0.029$, Max AC $= 0.466 \pm 0.038$) while incurring only a modest increase in Consistency Difference (0.180 ± 0.129). Both variants remain substantially more stable than strong prompting baselines such as GPT-5.1 Reasoning (Difference $= 0.595 \pm 0.033$) and Gemini-2.5-flash Non-Reasoning (Difference $= 0.972 \pm 0.024$).

Overall, Adaptive RULER achieves a **more favorable consistency–alignment balance**, trading a controlled increase in variance for large gains in action quality, while avoiding the severe instability that characterizes prompting-based approaches.

Q2: ProActor-Q4 exhibits higher FTR due to increased action recall rather than over-triggering. To better understand the elevated Fault Trigger Rate (FTR) of ProActor-Q4 relative to more conservative baselines, we analyze the distribution of predicted ready actions per dialogue turn on the test sets (Table 8). Despite higher FTR, the absolute number of actions predicted per turn remains small: on ABCD+, the model predicts fewer than one action per turn on average (mean 0.96, median 1), and on Home Loan the mean is 0.65 with a median of 0.5. In both datasets, 75% of turns contain at most one predicted action, indicating that ProActor-Q4 does not excessively over-trigger actions.

Statistic	ABCD+	Home Loan
Count	4,533	806
Mean	0.96	0.65
Std. Dev.	0.61	0.53
Minimum	0.00	0.00
25th Percentile	1.00	0.00
Median (50%)	1.00	0.50
75th Percentile	1.00	1.00
Maximum	10.00	3.00

Table 8: Ready action number per dialogue turn delivered by the RL-trained model on test sets

The observed increase in FTR therefore reflects **a deliberate shift toward higher action recall**, rather than uncontrolled over-triggering. This pattern holds for both ProActor-Q4 variants: Custom RULER maintains the lowest inconsistency with moderate FTR, while Adaptive RULER further increases recall with a controlled rise in fault rate. In both cases, the absolute number of predicted actions per turn remains small, indicating that higher FTR arises from acting more often when appropriate, not from action spamming.

Compared to aggressive ASG-based variants, which exhibit unstable consistency and sharply inflated fault rates, both ProActor-Q4 variants maintain substantially lower FTR while achieving stronger proactive timing and ready action rates. This suggests that the additional fault cost introduced by recall-oriented optimization is bounded and acceptable given the overall gains in proactive effectiveness.

Q3: Why does ProActor-Q4 succeed where prompting-based methods fall short? ProActor-Q4 succeeds by optimizing proactive behavior through **learning-based exploration of the action space**, rather than relying on static oracle examples or hand-crafted prompting rules. **The Custom RULER reward** explicitly encodes proactiveness requirements via rubric-based evaluation, providing dense, turn-level feedback that directly incentivizes timely and appropriate action readiness. This design leads to substantially stronger improvements in proactive timing and ready action rates than metric-only or prompting-based alternatives.

By adopting **composite rewards** that jointly account for action consistency and timing, ProActor-Q4 learns a calibrated trade-off between acting early and acting reliably. As demonstrated in our analysis, this allows ProActor-Q4 to significantly improve timing performance while keeping consistency degradation bounded. In contrast, prompting-based baselines exhibit polarized failure modes: reasoning-based methods tend toward excessive conservatism (e.g., GPT-5.1 Reasoning), while structure-driven prompting such as ASG induces uncalibrated aggressiveness with elevated fault rates and instability. These results highlight the advantage of reinforcement learning with ex-

#Dialogs Train/Test	Reward Type	PRI ↑	AC ↑		Max AC ↑		Proactive Timing ↑		Fault Trigger Rate ↓		Ready Action Rate ↑	
			Statistics	Δ	Statistics	Δ	Statistics	Δ	Statistics	Δ	Statistics	Δ
ABCD+												
100/50	RAC	0.2596	0.3239±0.01606	-0.1528	0.3881±0.02590	-0.1348	0.1223±0.02364	-0.1030	0.0640±0.01960	0.9636	0.3002±0.08088	0.2770
100/50	Max RAC	0.2264	0.3263±0.02468	-0.4740	0.4384±0.02264	-0.1587	0.1643±0.00207	-0.0616	0.0886±0.01212	2.6500	0.3862±0.01111	0.2262
100/50	General Ruler	0.4918	0.3494±0.02714	-0.2750	0.3978±0.02984	-0.2436	0.2324±0.01984	0.9056	0.1019±0.01910	3.7188	0.5945±0.05405	1.4867
100/50	Custom Ruler †	**0.6140**	0.3850±0.02544	-0.1995	0.4203±0.03200	-0.2476	0.2315±0.02393	0.6755	0.1068±0.02327	4.8358	0.5707±0.05717	1.2846
100/50	Weighted Metric (Max RAC)	0.5800	0.3929±0.03797	-0.0381	0.5099±0.06267	0.17834	0.2030±0.02414	0.55646	0.0818±0.01114	1.74	0.4860±0.03368	0.6535
100/50	Adaptive Metric ⋆	**0.8160**	0.4429±0.02656	-0.0291	0.4931±0.02914	-0.01426	0.1989±0.01303	0.18889	0.0631±0.01157	0.6667	0.4642±0.01688	0.5163
5647/692	RAC	0.4352	0.3368±0.02401	-0.1478	0.3857±0.01949	-0.1327	0.1799±0.00452	0.0128	0.0621±0.00765	0.1760	0.4148±0.02354	0.1818
5647/692	Max RAC	0.3737	0.3745±0.03439	-0.0926	0.5014±0.06809	0.0492	0.1172±0.03605	-0.0351	**0.0365±0.01013**	0.0919	0.2426±0.06160	-0.0008
5647/692	Custom Ruler ⋆	**0.7217**	0.4257±0.01469	-0.4791	0.4837±0.01848	-0.4731	0.2347±0.02014	0.56561	0.0708±0.00784	2.4654	0.5456±0.04169	1.8482
5647/692	Weighted Metric (RAC) †	**0.7078**	**0.4547±0.01284**	0.0016	0.5130±0.01260	0.0932	0.1920±0.00905	0.2096	0.0503±0.00149	0.5371	0.4329±0.01393	0.4391
5647/692	Weighted Metric (Max RAC)	0.2169	0.3068±0.02606	-0.2975	0.4399±0.05253	0.0011	0.1465±0.03091	0.0338	0.0700±0.01253	1.5204	0.3769±0.05893	0.3607
5647/692	Adaptive Metric	0.5159	0.3537±0.02264	-0.1950	0.3922±0.03489	-0.2524	0.2268±0.04591	0.2966	0.1137±0.01925	2.5661	0.6332±0.11113	1.0600
5647/692	Adaptive RULER ($\lambda_{\max} = 0.3$, RAC)	0.5921	0.4515±0.01795	0.1139	0.5134±0.01762	0.1487	0.2076±0.01269	0.3667	0.1004±0.00314	1.4526	0.4612±0.05153	0.4523
5647/692	Adaptive RULER ($\lambda_{\max} = 0.3$, Max RAC)	0.6026	0.4314±0.02240	-0.05045	**0.5861±0.04335**	0.1150	**0.2515±0.02722**	0.4410	0.1089±0.00832	1.9003	0.5212±0.05153	0.5818
Home Loan												
774/97	RAC	0.5668	**0.4701±0.03285**	0.8483	**0.5195±0.03016**	0.8462	0.0264±0.00590	-0.2502	0.0041±0.00245	-0.7494	0.0612±0.01856	-0.5365
774/97	Max RAC	0.4311	0.3894±0.01327	0.4891	0.4780±0.02065	0.6927	0.0093±0.00378	-0.8214	**0.0015±0.00072**	-0.9409	0.0177±0.00787	-0.9178
774/97	Custom Ruler	0.4220	0.2057±0.02409	-0.3760	0.2338±0.02114	-0.3329	**0.0846±0.00951**	2.5245	0.0355±0.00785	5.2040	0.4653±0.07412	4.8508
774/97	Weighted Metric (RAC)	0.5376	0.4359±0.01376	0.6695	0.4805±0.02399	0.7160	0.0205±0.00349	-0.2859	0.0020±0.00066	-0.8762	0.0481±0.00722	-0.6039
774/97	Weighted Metric (Max RAC)	0.4809	0.4133±0.02947	0.6461	0.5346±0.03620	0.9601	0.0270±0.00730	-0.4212	0.0049±0.00090	-0.4268	0.0506±0.02000	-0.7042
774/97	Adaptive Metric	0.5742	0.4677±0.01760	0.8090	0.5184±0.02184	0.8390	0.0282±0.00552	-0.3170	0.0042±0.00121	-0.7683	0.0653±0.01628	-0.5919
774/97	Hybrid RULER ($\lambda = 0.3$, RAC)	0.6672	0.4418±0.02502	0.7347	0.4632±0.02591	0.6569	0.0492±0.00122	0.7427	0.0107±0.00091	0.4341	0.1725±0.00620	0.6519
774/97	Adaptive RULER ($\lambda_{\max} = 0.3$, RAC)	0.6154	0.4173±0.00768	0.5111	0.4403±0.01019	0.4510	0.0397±0.00528	0.4382	0.0071±0.00070	-0.1682	0.1231±0.01985	0.4329
774/97	Hybrid RULER ($\lambda = 0.5$, RAC) ⋆	**0.6836**	0.4233±0.02586	0.6608	0.4501±0.02397	0.5636	0.0613±0.00751	1.2351	0.0154±0.00184	1.4814	0.2236±0.03846	1.4644
774/97	Adaptive RULER ($\lambda_{\max} = 0.5$, RAC)	0.5416	0.3919±0.01226	0.4050	0.4483±0.01509	0.4791	0.0325±0.00553	0.3638	0.0063±0.00225	-0.1435	0.0946±0.01411	0.1204
774/97	Hybrid RULER ($\lambda = 0.75$, RAC) †	**0.6711**	0.3584±0.01861	0.3707	0.3786±0.01823	0.2833	0.0720±0.01011	1.5767	0.0181±0.00349	2.5504	0.2931±0.05276	2.2065
774/97	Adaptive RULER ($\lambda_{\max} = 0.75$, RAC)	0.6227	0.4019±0.02993	0.5041	0.4258±0.02664	0.4238	0.0447±0.00239	0.5757	0.0105±0.00150	0.3950	0.1708±0.01436	0.8176
774/97	Hybrid RULER ($\lambda = 0.3$, Max RAC)	0.6112	0.3868±0.02672	0.4823	0.4329±0.04006	0.5458	0.0595±0.00465	0.9393	0.0210±0.00250	1.9204	0.2540±0.01615	1.5509
774/97	Adaptive RULER ($\lambda_{\max} = 0.3$, Max RAC)	0.5734	0.3950±0.02936	0.5366	0.4658±0.03821	0.6764	0.0501±0.00547	1.0451	0.0131±0.00131	1.5299	0.1561±0.00930	0.5190
774/97	Hybrid RULER ($\lambda = 0.5$, Max RAC)	0.6533	0.4114±0.02512	0.5827	0.4491±0.02862	0.5771	0.0580±0.00210	0.9779	0.0169±0.00252	0.8783	0.2385±0.01071	1.4148
774/97	Adaptive RULER ($\lambda_{\max} = 0.5$, Max RAC)	0.5849	0.3983±0.01492	0.4339	0.4536±0.01820	0.5338	0.0436±0.00421	0.9312	0.0086±0.00157	0.3971	0.1295±0.01488	0.5244
774/97	Hybrid RULER ($\lambda = 0.75$, Max RAC)	0.5431	0.2795±0.01821	-0.0153	0.3095±0.01610	-0.0287	0.0711±0.01088	1.9046	0.0264±0.00743	2.5972	0.3683±0.07846	3.0949
774/97	Adaptive RULER ($\lambda_{\max} = 0.75$, Max RAC)	0.3357	0.2448±0.01290	-0.1612	0.3382±0.01703	0.0401	0.0715±0.01286	2.2181	0.0362±0.01368	6.5137	0.2977±0.09482	3.1734

Table 9: Performance given different reward settings. #Dialogs Train/Test denotes the dialogue number for training and testing. Statistics report mean±std on the test set. ⋆, † mark top-1, top-2 methods by PRI (Proactiveness Ranking Index, see Section K.4).

plicit proactiveness supervision for resolving the precision–consistency–timing trade-off in proactive decision-making.

L More Ablations

This section provides comprehensive ablation study results, including reward type comparisons, data scaling effects, and additional analysis tables.

L.1 Reward Type Comparison

The full ablation results are presented in Table 9. Here we provide a detailed analysis.

Reward Types. Among all evaluated reward types, **Custom RULER** achieves the most balanced performance, which supports our reward decision to leverage it when further scaling up data volumes.

In terms of Action Consistency, Custom RULER attains the highest AC score (0.5540±0.0160), outperforming RAC, Max RAC, and General RULER, while maintaining competitive Max AC (0.6004±0.0243). In contrast, Adaptive RULER substantially harms action consistency (AC drops to 0.232±0.0089), suggesting that overly aggressive emphasis on timing can destabilize action alignment at small data scales.

From a timing perspective, Custom RULER improves Proactive Timing (0.1548±0.0164) over RAC and General RULER, while keeping the Fault Trigger Rate at the lowest level among all reward variants (0.0106±0.0037). Although Weighted Metric and Adaptive Metric rewards achieve higher Proactive Timing, they do so at the cost of significantly increased Fault Trigger Rates (e.g., 0.0778 and 0.0631, respectively), indicating overly aggressive action triggering and reduced reliability.

We further compute the **Consistency Difference** for each reward on **ABCD+** (training 100 + test 50), as shown in Table 10:

Reward Type	Consistency Difference (↓)
Custom RULER	**0.084 ± 0.054**
RAC	0.088 ± 0.088
General RULER	0.111 ± 0.063
Adaptive Metric	0.113 ± 0.094
Weighted Metric (Max RAC)	0.323 ± 0.078
Max RAC	0.540 ± 0.132
Adaptive RULER	0.664 ± 0.101

Table 10: Consistency Difference on ABCD+ (training 100/test 50) for reward type decision.

Custom RULER achieves the lowest AC Difference among all reward types, indicating the most consistent alignment between AC and Max AC and supporting its selection as a safe and robust reward choice before scaling.

Data Scaling Effects. Next, we analyze the effect of scaling training data for Custom RULER on the ABCD+ dataset, ranging from training 300 + test 100 to the full dataset (training 5647 + test 692).

Action Consistency: As training data on ABCD+ increases, Action Consistency remains largely stable. AC improves from 0.554 (100 samples) to 0.597 (1k samples), and while it decreases at larger scales (e.g., 0.320 at 3k, 0.426 at full scale), the corresponding AC Difference stays moderate (e.g., 0.079–0.136), suggesting that the gap between AC and Max AC does not widen significantly. This indicates that the model does not collapse into optimizing a narrow subset of actions; this behavior aligns with our design goal: Custom RULER prioritizes proactive decision-making over strictly maintaining alignment with reference actions at every turn.

Timing Behavior: Proactive Timing consistently improves with more data, increasing from 0.155 (100 samples) to 0.235 at full scale, demonstrating stronger anticipation of future-ready actions. This improvement is accompanied by a moderate increase in Fault Trigger Rate (from 0.010 to 0.071), but the values remain controlled. Overall, scaling data strengthens proactive timing while preserving reasonable action consistency, aligning with the intended design of Custom RULER.

M Further Discussions

This section elaborates on our RL design choices and presents additional analyses of factors that impact RL performance. Specifically, Section M.1 examines our reward granularity decisions through a set of controlled experiments, while Section M.2 discusses the rationale behind selecting the importance sampling ratio cap $\bar{r}_u^{(k)}$.

M.1 Reward Granularity Decision

To assess the effectiveness of trajectory-level versus turn-level reward formulations, we construct two comparative experimental groups. Each group incorporates both metric-based rewards (RAC and Max RAC) and RULER-based rewards (General RULER and Custom RULER). Experiments are conducted on a reduced ABCD+ subset comprising 100 training dialogues and 50 validation dialogues. We evaluate each reward design by tracking **four metadata-derived metrics** (Section G.3) on the training set throughout the learning process.

We collect RL training statistics (training reward, loss, entropy) together with metadata-derived metrics (proactive timing, AC, Max AC) for each run. Each comparison group organizes these six metric diagrams in this way: The top row displays timing reward and training reward, demonstrating proactive timing and training performance stability differences. The middle row shows the RAC score and the Max RAC score evolution, indicating the effectiveness of proactive action consistency. Bottom row presents training loss and entropy metrics.

M.1.1 RAC GROUP

Besides Figure 26, a brief summary in Table 11 shows that the **turn-wise advances in exploration diversity and reward progression**, while the **trajectory-level** demonstrates superior **better convergence** (64% improvement) and **AC consistency** with 37% lower variance.

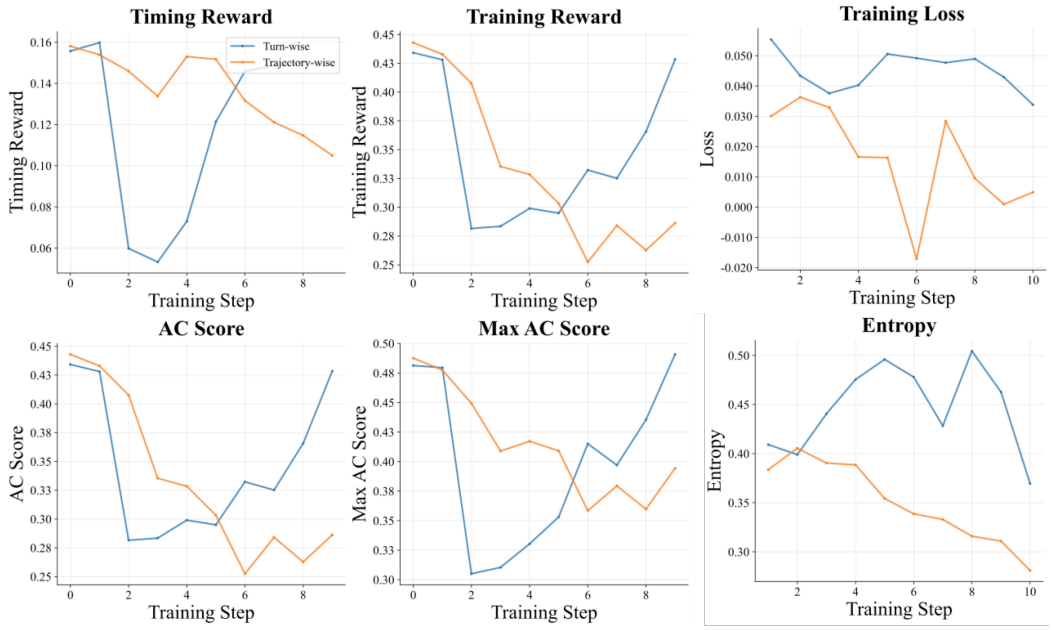


Figure 26: Turn-level(blue) vs Trajectory-level(orange) **RAC** Training Comparison.

Metric	Turn-level	Trajectory-level	Better
Loss Stability	0.0450±0.0066	**0.0159±0.0168**	Trajectory-level
Entropy Diversity	**0.4463±0.0445**	0.3502±0.0411	Turn-level
Reward Progression	**Positive trend**	Declining trend	Turn-level
Timing Reward	0.121±0.0427	0.1369±0.0186	Trajectory-level
AC Consistency	0.3473±0.0625	0.3336±0.0703	Turn-level
Max AC Performance	0.3998±0.0721	0.4142±0.0452	Trajectory-level

Table 11: Training stability comparison between turn-level and trajectory-level RAC reward computation.

M.1.2 MAX RAC GROUP

When it comes to Max RAC as shown in Figure 27, **the turn-level reward demonstrates superior performance across 4 out of 5 metrics**, including 17.5% higher Max AC performance, better training rewards, superior exploration stability, and more consistent AC scores, making it the optimal choice for Max RAC training despite trajectory-level showing better loss convergence.

M.1.3 GENERAL RULER GROUP

In our experiments, we observe that the trajectory-level context length grows rapidly as training progresses. As shown in Figure 28, even at this smaller scale, the context length explodes around training step 12, causing training to fail due to **out-of-memory (OOM) errors** on a 1×H200 GPU. Although **trajectory-level rewards achieve better performance** in Table 13, we adopt **a conservative choice in favor of turn-level modeling for its stability and feasibility**.

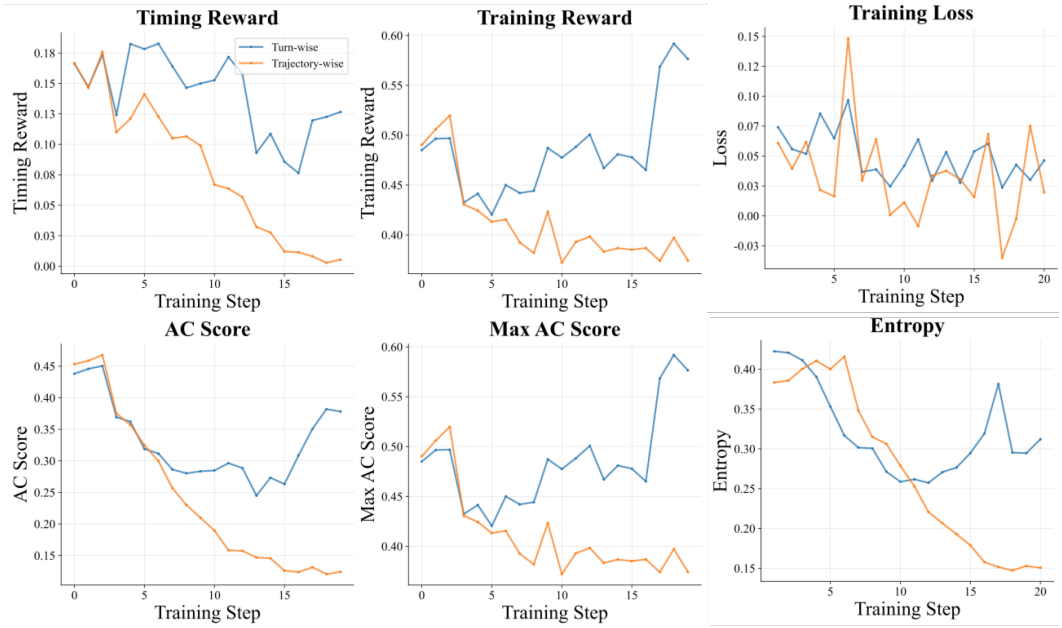


Figure 27: Turn-level(blue) vs Trajectory-level(orange) **Max RAC** Training Comparison.

Metric	Turn-level	Trajectory-level	Better
Loss Stability	0.0501±0.0201	**0.0344±0.0390**	Trajectory
Training Reward	**0.4845±0.0469**	0.4124±0.0439	Turn-level
Entropy Diversity	**0.3205±0.0558**	0.2728±0.1029	Turn-level
Timing Reward	**0.1415±0.0328**	0.0791±0.0576	Turn-level
AC Consistency	**0.3308±0.0628**	0.2428±0.1233	Turn-level
Max AC Performance	**0.4845±0.0469**	0.4124±0.0439	Turn-level

Table 12: Training stability comparison between turn-level and trajectory-level Max RAC reward.

Figure 28: Turn-wise(blue) vs Trajectory-wise(orange) **General RULER** Training Comparison.

Metric	Turn-level	Trajectory-level	Better
Loss Performance	0.0232±0.0162	**0.0016±0.0242**	Trajectory
Training Reward	0.7560±0.0338	0.7750±0.0273	Trajectory
Entropy Diversity	0.3482±0.0468	**0.4034±0.0289**	Trajectory
Timing Reward	**0.2649±0.0591**	0.1477±0.0206	Turn-level
AC Consistency	0.3265±0.0532	0.3593±0.0657	Trajectory
Max AC Performance	0.3788±0.0513	**0.4102±0.0631**	Trajectory

Table 13: Training stability comparison between turn-level and trajectory-level General RULER reward computation.

M.1.4 CUSTOM RULER GROUP

Similar to the General RULER group, Figure 29 shows that training terminates abruptly around step 16 due to context explosion. Further analysis in Table 14 indicates that turn-level rewards achieve superior entropy diversity and timing reward optimization, motivating our recommendation of turn-level Custom RULER despite the advantages of trajectory-level rewards on other metrics.

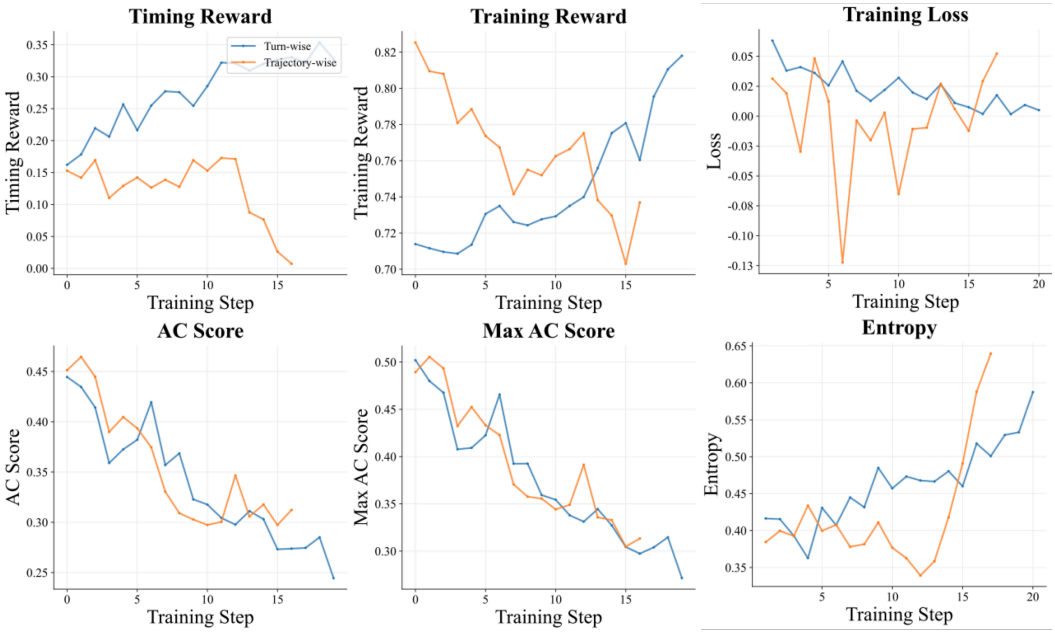


Figure 29: Turn-wise(blue) vs Trajectory-wise(orange) **Custom RULER** Training Comparison.

Metric	Turn-level	Trajectory-level	Better
Loss Performance	0.023±0.016	**0.003±0.043**	Trajectory
Training Reward	0.745±0.034	0.766±0.032	Trajectory
Entropy Diversity	0.463 ± 0.054	0.421±0.080	Turn-level
Timing Reward	**0.276±0.056**	0.124±0.049	Turn-level
AC Consistency	0.338±0.059	0.355±0.059	Trajectory
Max AC Performance	0.374±0.068	0.393±0.065	Trajectory

Table 14: Training stability comparison between turn-level and trajectory-level Custom RULER reward computation.

Considering the importance of training stability in 4×H200 and 8×H100 environments, together with the clear performance advantages of turn-level rewards, we adopt **turn-level modeling** as our initial strategy. This choice allows us to systematically study effective proactive timing optimiza-

tion and establish a solid foundation for future extensions to **trajectory-guided multi-turn reward learning**.

M.2 GRPO PARAMETERS

This section focuses on hyperparameter tuning on the GRPO algorithm.

M.2.1 IMPORTANCE SAMPLING RATIO CAP $\bar{r}_u^{(k)}$

Experiment To identify an appropriate value for the gamma clamp parameter γ without degrading model performance consistency, we conduct a controlled comparison study using two configurations: $\gamma \in \{10, 100\}$. We employ the Home Loan dataset at full scale with the Qwen2.5-14B-Instruct model using distributed data parallel (DDP) training PyTorch Team (2025). Each γ configuration is evaluated through two independent training runs to assess run-to-run consistency, resulting in four total experimental runs. We monitor eight key training metrics categorized into two groups: (1) *ML statistics*, including training loss, entropy, reward, reward standard deviation, and policy loss; and (2) *Proactive statistics*, including timing reward, AC score, and Max AC score. These metrics provide comprehensive coverage of both general machine learning training behavior and proactive agent-specific performance indicators.

We further define Combined Consistency Score to measure the experiment that better captures the balance between performance value consistency and trajectory correlation.

Final Value Consistency Score how similar the final training values are between runs within each group:

$$C_{\text{final}}(\text{metric}) = 1 - \frac{|v_{1,\text{final}} - v_{2,\text{final}}|}{|v_{1,\text{final}}| + |v_{2,\text{final}}|} \tag{39}$$

Where final values are extracted from the training time series:

$$v_{i,\text{final}} = x_{i,U_i} \quad \text{where } U_i = \max\{u : x_{i,u} \text{ exists}\} \tag{40}$$

and $v_{1,\text{final}}, v_{2,\text{final}}$ are the last recorded training values from run 1 and run 2, $x_{i,u}$ is the metric value at training step u for run i, U_i is the final training step for run i.

Trajectory correlation measures how well the training curves follow similar patterns over time using

$$C_{\text{trajectory}}(\text{metric}) = \rho(\mathbf{x}_1, \mathbf{x}_2) = \frac{\sum_{u=1}^{n}(x_{1,u} - \bar{x}_1)(x_{2,u} - \bar{x}_2)}{\sqrt{\sum_{u=1}^{n}(x_{1,u} - \bar{x}_1)^2 \sum_{u=1}^{n}(x_{2,u} - \bar{x}_2)^2}} \tag{41}$$

where ρ is the **Pearson correlation coefficient** Pearson (1895), $\mathbf{x}_1, \mathbf{x}_2$ are complete time series trajectories from run 1 and run 2, $x_{i,u}$ is the metric value at training step u for run i, $\bar{x}_i = \frac{1}{n}\sum_{u=1}^{n} x_{i,u}$ is the mean value across all time steps for run i, n is the number of overlapping training steps between runs.

Combined Consistency Score averages final value consistency and trajectory correlation across all metrics:

$$C_{\text{combined}}(\text{group}) = \frac{1}{2}\left(\frac{1}{m}\sum_{i=1}^{m} C_{\text{final}}(\text{metric}_i) + \frac{1}{m}\sum_{i=1}^{m} C_{\text{trajectory}}(\text{metric}_i)\right) \tag{42}$$

Where m is the total number of metrics evaluated for the group, metric_i represents the i-th training metric (loss, entropy, reward, etc.)

Performance Analysis Table 15 summarizes the consistency results across all monitored metrics. Overall, $\gamma = 10$ demonstrates stable and reliable training behavior, characterized by uniformly high trajectory correlations ($\rho \in [0.650, 0.974]$) and strong final value alignment (consistency scores between 0.974 and 0.988). No anti-correlated behaviors are observed, and run-to-run variance remains low across metrics (average 1.4%), indicating reproducible optimization dynamics. In contrast, $\gamma = 100$ exhibits less predictable training behavior, with substantially more variable trajectory

correlations ($\rho \in [0.362, 0.964]$) and an anti-correlated timing reward trajectory ($\rho = -0.586$), suggesting instability in timing-sensitive optimization. Although $\gamma = 100$ achieves higher absolute performance on certain RAC-related metrics, its increased variance (2.4%) and inconsistent dynamics reduce reliability, making $\gamma = 10$ a more dependable choice for stable training.

Metric	**Final Consistency**	**Trajectory Correlation**	**Variance(%)**
$\gamma = 10$ *Group*			
Training Loss	0.974	0.899	–
Training Entropy	0.978	**0.650**	–
Training Reward	**0.985**	**0.948**	**1.5**
RAC Score	0.984	0.753	1.6
Max RAC Score	**0.988**	0.828	**1.2**
Timing Reward	**0.962**	**0.974**	–
Average	**0.975**	**0.840**	**1.4**
$\gamma = 100$ *Group*			
Training Loss	**0.979**	0.887	–
Training Entropy	0.978	0.362	–
Training Reward	0.952	0.945	4.8
RAC Score	**0.998**	**0.964**	**0.2**
Max RAC Score	0.978	**0.951**	2.2
Timing Reward	0.522	-0.586	–
Average	0.886	0.669	2.4
Overall Comparison			
Combined Score	$\gamma = 10$**: 0.907** vs $\gamma = 100$: 0.778		
Anti-Correlated Metrics	$\gamma = 10$**: 0** vs $\gamma = 100$: 1		
Reproducibility	$\gamma = 10$**: High** vs $\gamma = 100$: Moderate		

Table 15: Training consistency comparison between $\gamma = 10$ and $\gamma = 100$ groups arranged sequentially for single-column format. $\gamma = 10$ achieves 16.6% higher combined consistency score with no anti-correlated behaviors.

Training Dynamics Visualization Figure 30 and 31 further illustrates the training dynamics under different importance sampling ratio caps. For $\gamma = 10$, independent training runs closely follow similar trajectories across all metrics, exhibiting smooth convergence and consistent optimization patterns throughout training. By contrast, $\gamma = 100$ shows larger fluctuations and increasingly divergent trajectories as training progresses, with pronounced instability in timing-related signals. These fluctuations indicate that higher ratio caps require more updates to stabilize and may amplify sensitivity to stochastic effects. Overall, the qualitative trends in figures align with the quantitative consistency scores in Table 15, reinforcing $\gamma = 10$ as the more stable and reproducible configuration.

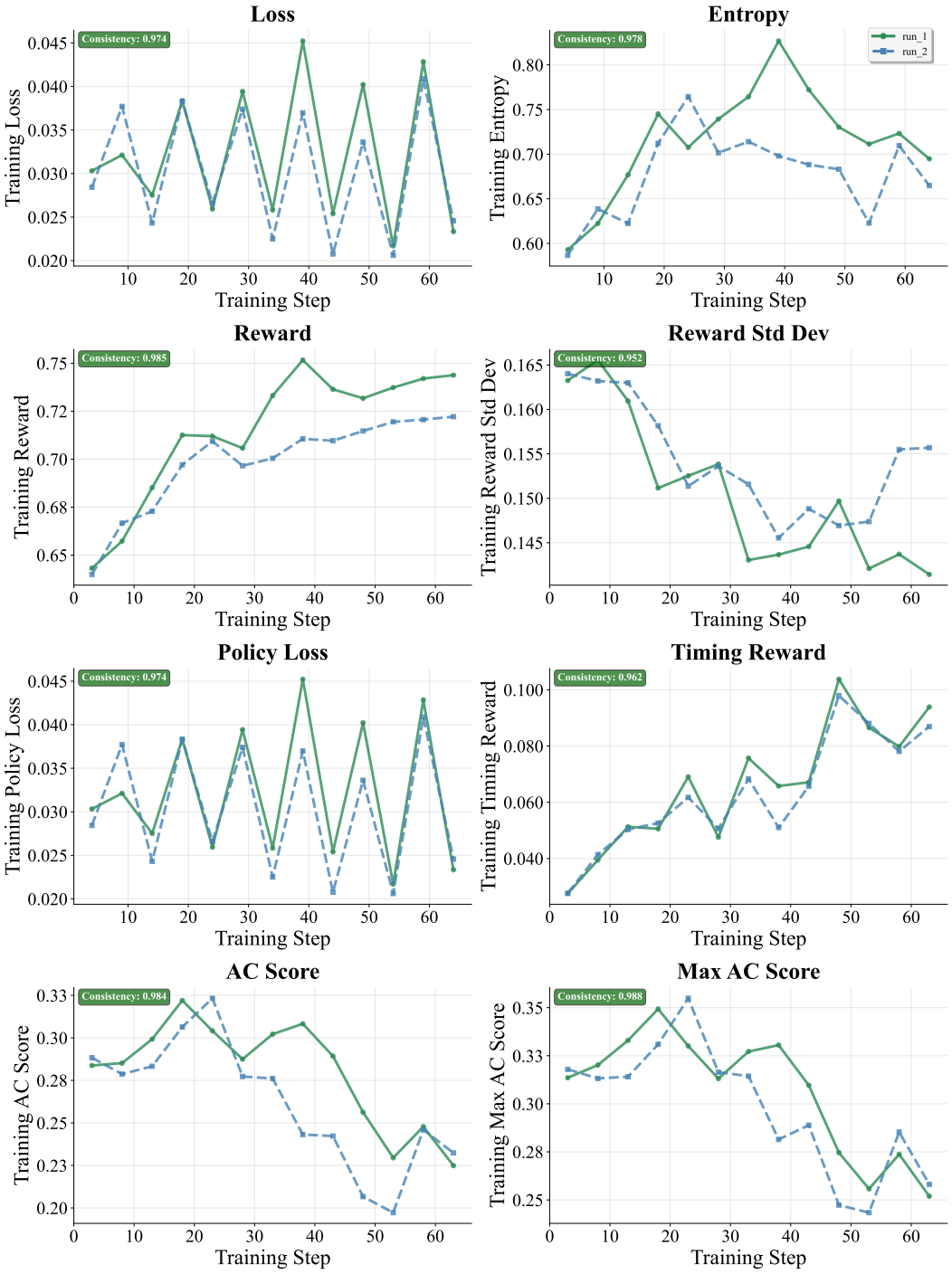


Figure 30: Training curve consistency analysis across two independent runs ($\bar{r}_u^{(k)} = 10$).

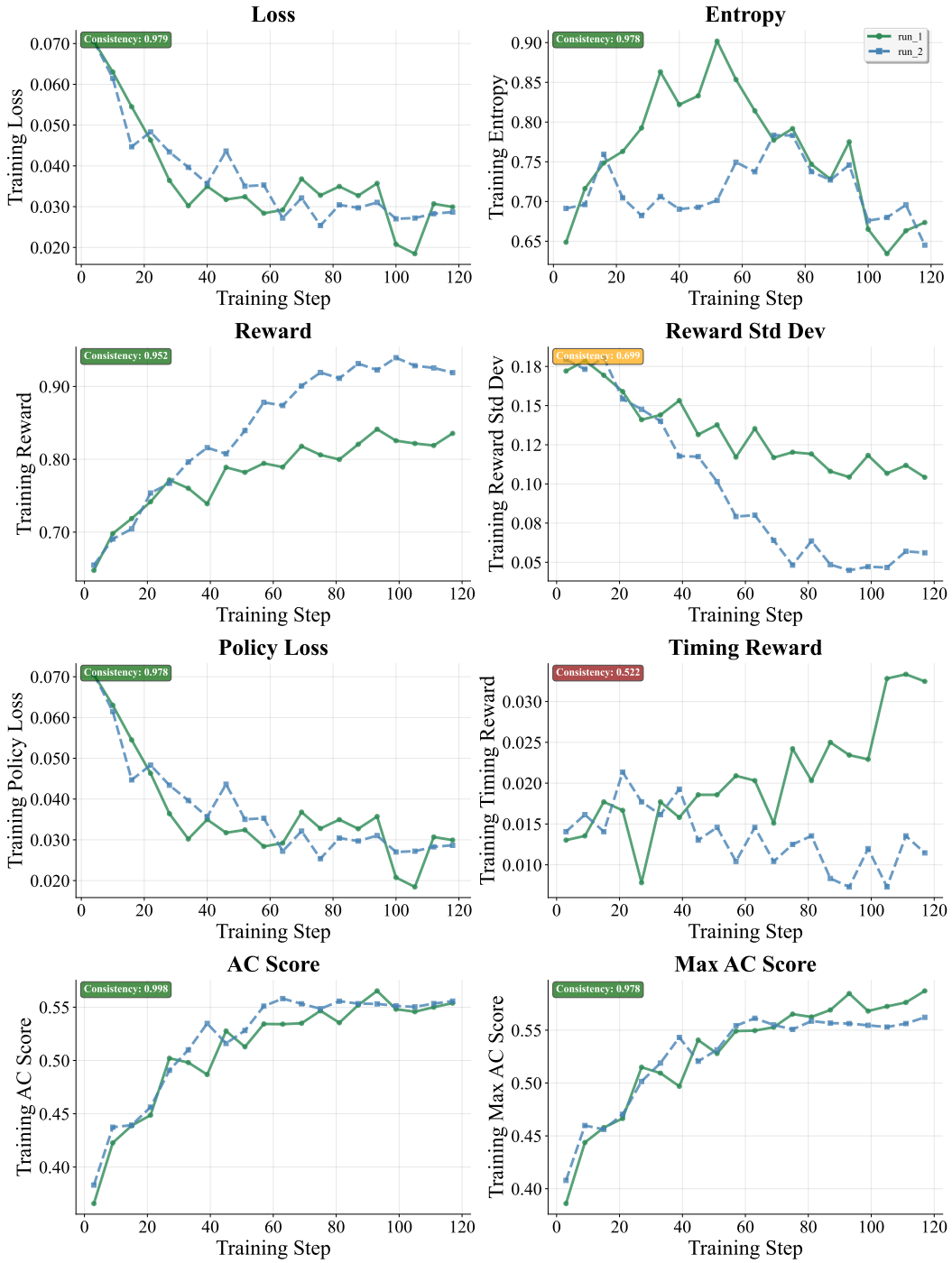


Figure 31: Training curve consistency analysis across two independent runs ($\bar{r}_u^{(k)} = 100$).