TReDS: Trajectory-Grounded Requirement–Capability Modeling for Training Distribution Shaping in Tool-Interactive Tasks

Han Liu
School of Information Science and Technology
Beijing Forestry University
Beijing, China
haizyliu@gmail.com

Abstract

Verifiable tool-interactive agents produce rich trajectories that contain far more information than final success or failure. However, standard RL post-training with verifiable rewards often collapses these trajectories into scalar outcomes and trains over a largely fixed data stream, leaving open the question of which examples are most useful for improving the current policy. We introduce TReDS, a trajectory-grounded framework for capability-conditioned training distribution shaping. TReDS estimates multidimensional task requirements from offline probing trajectories and controlled tool-view evidence, maintains the policy capability state in the same semantic space, and uses reliability-gated online correction to form a policy-conditioned requirement representation. A scheduler then converts requirement–capability mismatch, recent utility, and stabilizing pressure into a time-varying training distribution for GRPO, without modifying the GRPO objective. We evaluate TReDS with a progressive stress-test suite spanning τ^2-Bench official evaluation, τ-Bench native-bridge evaluation, BFCL V3, and API Bank, using only the τ^2-Bench training split for policy optimization. TReDS improves over the GRPO baseline under the same training data and backend, and shows stronger behavior retention under same-family and cross-protocol evaluations. These results suggest that RL post-training for tool-interactive agents should optimize not only how trajectories are rewarded, but also where trajectory-revealed requirements allocate training probability mass.

1 Introduction

Large language models are increasingly evaluated and trained as interactive agents rather than single-turn generators [18, 47, 36, 39, 43]. In verifiable tool-interactive tasks, a policy must invoke tools, observe returns, update states, follow task-specific rules, and produce outputs checked by verifiers or judges [48, 1, 12, 14]. Such tasks produce trajectories containing reasoning steps, tool calls, argument construction, state changes, verifier-observable execution information, and final responses [26, 15, 20, 2]; these trajectories reveal not only success or failure, but also planning, grounding, tool use, state consistency, and long-horizon behavior [18, 51, 5, 15].

Recent benchmarks for tool-use, web, software, and API-use/function-calling agents make this interaction-centric setting increasingly important [15, 26, 25, 19, 6, 28]. Meanwhile, RL from verifiable rewards, including GRPO-style optimization, provides a practical route for improving policies when outcomes can be automatically judged [32, 9, 40]. Yet scalar rewards leave a key distributional question unresolved: after a trajectory is scored, which examples should receive more training probability mass [23, 13, 16]? This is critical because verifiable tool-interactive tasks are not governed

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

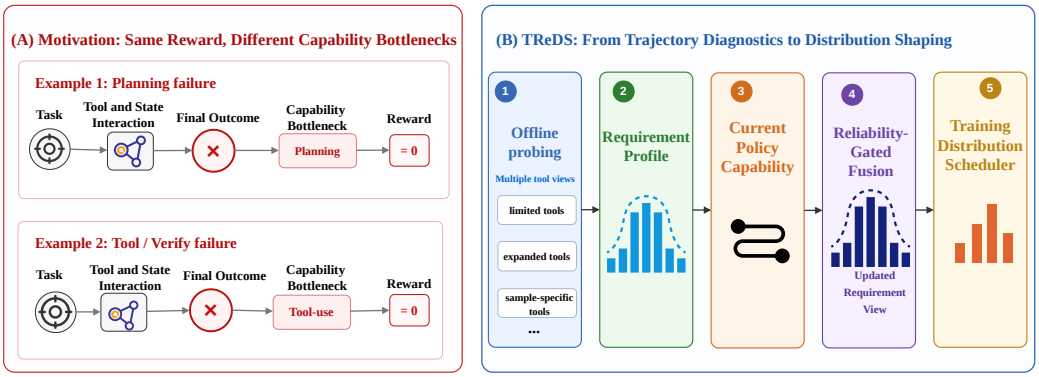

Figure 1: (left) In verifiable tool-interactive tasks, similar final scalar rewards can correspond to different capability bottlenecks. (right) TReDS uses trajectory diagnostics to construct task-side requirement profiles, tracks current policy capability from online behavior, and applies reliability-gated correction before a distribution scheduler allocates training probability mass.

by a single notion of difficulty [18, 42]; similar rewards may correspond to different capability requirements, and tool-related failures are only part of a broader multidimensional requirement space [27]. We therefore argue that trajectories should serve not only as outcomes to be scored, but also as evidence for shaping the training distribution beyond fixed data streams or scalar curricula [23, 30]. Figure 1 summarizes this intuition.

We propose TReDS, short for Trajectory-Grounded Requirement Modeling with Capability-Conditioned Training Distribution Shaping. As shown in Figure 2, TReDS estimates multidimensional requirement priors and reliability scores from multi-view probing trajectories and tool-view performance deltas, maintains policy capability in the same semantic space, and fuses online policy-conditioned evidence through a reliability gate. A scheduler then converts requirement–capability mismatch, recent utility, and stabilizing pressure into a time-varying training distribution, while keeping the GRPO objective unchanged. We evaluate TReDS with a progressive stress-test suite: all

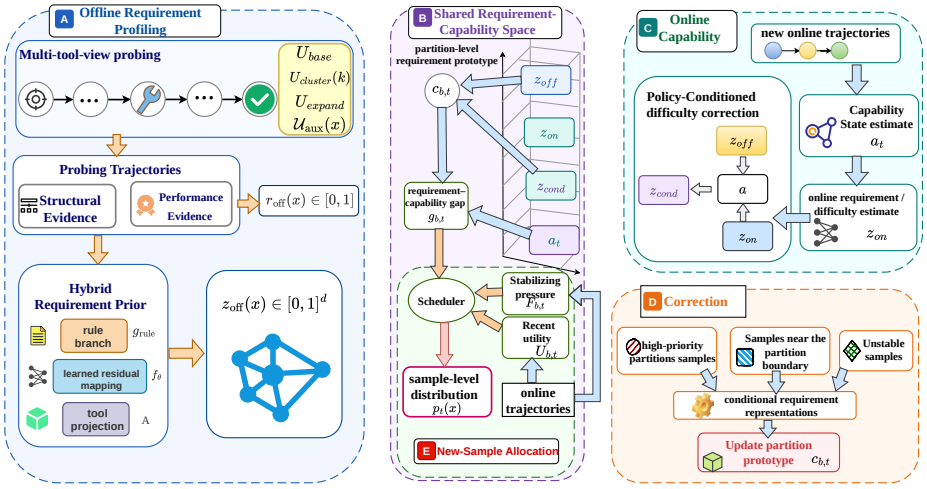

Figure 2: The overall framework of TReDS.

models are trained only on the τ^2-Bench training split and evaluated on the held-out τ^2-Bench official test, τ-Bench native-bridge evaluation, BFCL V3, and API Bank. Empirically, TReDS improves over the GRPO baseline on τ^2-Bench and shows stronger same-family and cross-protocol retention. Ablations and mechanism analyses confirm that online correction, reliability-gated fusion, scheduler-based allocation, offline profiling, multidimensional requirement modeling, and tool-aware evidence

all contribute to performance, and that the scheduler converts controller-side estimates into sampling probabilities rather than degenerating into a static curriculum or hardest-first sampler.

Our contributions: **(1)** We formulate agent RL post-training as trajectory-grounded requirement–capability distribution shaping. **(2)** We propose TReDS, which builds multidimensional requirements from trajectory and tool-view evidence and updates them with reliability-gated online correction. **(3)** We run GRPO over a scheduler-shaped distribution $p_t(x)$, allocating limited training probability mass without changing the objective. **(4)** We validate TReDS with progressive stress tests and link its gains to multidimensional modeling, tool-aware evidence, and scheduler-based allocation.

2 Preliminary

2.1 Verifiable Tool-Interactive Agent Tasks and Trajectories

We consider a collection of verifiable tool-interactive agent tasks $\mathcal{X}$. Each example $x \in \mathcal{X}$ requires an agent policy π_θ to interact with a tool environment U, take intermediate actions, and generate an output checked by a verifier or task-specific judge. Given x and U, a rollout produces a trajectory: $\tau(x;U) = \{o_t, u_t, \eta_t\}_{t=1}^{T}$, where o_t is an observation or intermediate state, u_t is an agent action, and η_t records tool returns, environment state changes, or verifier-observable execution information. Actions may include reasoning steps, tool calls, argument construction, state updates, and the final response. We use $\Phi(\tau)$ as a structured trajectory feature extractor. Unlike the final reward, $\Phi(\tau)$ summarizes trajectory-level behavioral signals such as planning structure, grounding failures, tool-call validity, verification behavior, state consistency, and long-horizon dependencies. Requirement estimation is therefore based on trajectory-level evidence rathefr than only final outcomes.

2.2 Tool Views and Normalized Performance

A tool environment U specifies available tool schemas, tool execution semantics, and verifier-observable state changes. For offline profiling, we define:

$$\mathcal{U}_{\text{probe}}(x) = \{U_{\text{base}}, U_{\text{expand}}\} \cup \{U_{\text{cluster}}(k) : k \in \mathcal{K}\} \cup \mathcal{U}_{\text{aux}}(x), \tag{1}$$

where $\mathcal{K} = \{1, \ldots, K\}$ indexes tool clusters. In our implementation, $K = 5$, with tool clusters corresponding to retrieval, planning, verification, execution, and memory. The number of clusters and their assignments are induced from the τ^2-Bench training tasks through model-assisted grouping, manual consolidation, and held-out validation, rather than specified a priori. The resulting taxonomy is fixed throughout training and evaluation. Details are provided in Appendix B.1. U_{base} covers core reasoning dimensions, U_{expand} provides broader tool access, $U_{\text{cluster}}(k)$ isolates the marginal contribution of the k-th tool cluster, and $\mathcal{U}_{\text{aux}}(x)$ contains optional sample-conditioned views such as sample-specific or evolved views. Given x and U, let $\mathcal{T}(x, U)$ denote the probing trajectories collected under that view. We define:

$$P(x, U) = \text{Agg}_{\tau \in \mathcal{T}(x,U)} \, \text{Perf}(\tau). \tag{2}$$

Here, $\text{Perf}(\tau)$ is calibrated from verifier-derived signals, including task success, verification quality, tool-use precision, state alignment, trajectory consistency, answer fidelity, and confidence.

2.3 A Shared Semantic Space for Requirements and Capabilities

We represent task requirements and policy capabilities in the same d-dimensional semantic space:

$$\begin{aligned} \mathcal{D} = \{&\text{planning}, \text{retrieval/grounding}, \text{tool use}, \text{verification}, \\ &\text{long-horizon consistency}, \text{tool sensitivity}, \text{tool gap}\}, \quad d = 7. \end{aligned} \tag{3}$$

These seven dimensions were not fully prespecified. Instead, we induced the requirement taxonomy from the τ^2-Bench training tasks through model-assisted atomic-requirement extraction, cross-task analysis, and manual consolidation. The candidate dimensions were further examined on held-out training tasks and representative offline probing trajectories, without using the official test set. The complete induction procedure is described in Appendix B.2. For each example x, offline profiling produces an offline requirement prior: $z_{\text{off}}(x) \in [0,1]^d$, and an offline reliability score: $r_{\text{off}}(x) \in [0,1]$. Here, $z_{\text{off}}(x)$ represents the multidimensional capability requirements

of x, while $r_{\text{off}}(x)$ measures the stability and trustworthiness of this profile. During training, we maintain a policy capability state: $a_t \in [0,1]^d$, which represents the capability profile of the trainable policy π_θ in the shared space. Online trajectories provide additional evidence about the current policy, from which we estimate: $z_{\text{on},t}(x) \in [0,1]^d$. The offline evidence and online policy-conditioned evidence are then fused into: $z_{\text{cond},t}(x) \in [0,1]^d$. This shared space allows task-side requirements and policy-side capabilities to be directly compared. To initialize the controller, we use: $s_{\text{off}}(x) = w_{\text{off}}^\top z_{\text{off}}(x), \quad w_{\text{off}} \in \mathbb{R}^d$, as a scalar control interface. This projection is used only for partition initialization; the underlying representation remains multidimensional.

2.4 Time-Varying Training Distribution

At training step t, the controller defines normalized task weights $p_t \in \Delta(\mathcal{X}_{\text{train}})$ to guide task selection. Batch construction uses these weights together with the coverage constraints described in Section 3.3. In our setting, p_t may vary over training and is constructed by the controller in Section 3. Section 3 therefore focuses on estimating z_{off}, r_{off}, a_t, and $z_{\text{cond},t}$, and converting them into the training distribution $p_t(x)$.

3 Method

TReDS combines trajectory-grounded multidimensional requirement modeling with capability-conditioned training distribution shaping. TReDS first uses a probe policy to roll out each training example under multiple tool views, and estimates $z_{\text{off}}(x)$ and $r_{\text{off}}(x)$ from the resulting trajectory diagnostics and tool-view performance evidence. During training, TReDS updates the policy capability state a_t from online trajectories. Online evidence is fused with the offline prior to obtain $z_{\text{cond},t}(x)$, and the scheduler converts requirement–capability mismatch into the GRPO training distribution $p_t(x)$. Thus, TReDS keeps the GRPO objective unchanged while replacing a fixed data stream with a controller-shaped distribution over training examples. The overall procedure is summarized in Algorithm 1 in the Appendix.

3.1 Trajectory-Grounded Offline Requirement Profiling

Trajectory diagnostics and normalized performance. For each training example x, we collect probing trajectories under multiple tool views. Let$\tau^{\text{probe}}(x;U)$ denote a probing trajectory obtained under tool view U, and let $\phi_x^{\text{probe}}(U) = \Phi(\tau^{\text{probe}}(x;U))$ denote its trajectory features. These features provide structural evidence for requirement inference, including action structure, failure stage, tool-use quality, verification behavior, state consistency, and long-horizon dependencies.

In implementation, we aggregate trajectory features over the core view set $\mathcal{U}_\phi(x) = \{U_{\text{base}}, U_{\text{expand}}, U_{\text{evo}}(x)\}$ as

$$\bar{\phi}_x = \text{Agg}\left(\{\phi_x^{\text{probe}}(U) : U \in \mathcal{U}_\phi(x)\}\right). \tag{4}$$

Here, $\bar{\phi}_x$ is the multi-view trajectory summary of example x. The aggregation operator computes feature-wise means and variances, capturing both average behavioral patterns and cross-view instability. Each trajectory is also mapped to a normalized performance value $P(x,U)$ by aggregating $\text{Perf}(\tau)$ as defined in Section 2.2.

Tool-aware performance deltas. To extract tool-aware requirement evidence, we compare normalized performance across tool views. The broad tool-access gain is defined as $\Delta_{\text{tool}}(x) = P(x,U_{\text{expand}}) - P(x,U_{\text{base}})$. For the k-th tool cluster, the marginal gain is $\Delta_k(x) = P(x,U_{\text{cluster}}(k)) - P(x,U_{\text{base}}), \quad k \in \mathcal{K}$. We stack the cluster-level gains into $\Delta_{\text{clus}}(x) = [\Delta_1(x),\ldots,\Delta_K(x)]^\top \in \mathbb{R}^K$. The evolved-view residual diagnostic is $\Delta_{\text{evo}}(x) = P(x,U_{\text{evo}}(x)) - P(x,U_{\text{expand}})$. When a sample-specific view is available, we further define $\Delta_{\text{spec}}(x) = P(x,U_{\text{spec}}(x)) - P(x,U_{\text{expand}})$. These tool-view deltas are not used as isolated heuristics; they are combined with $\bar{\phi}_x$ to construct the offline requirement prior.

Hybrid requirement prior from trajectory and tool evidence. The offline requirement prior combines rule-based trajectory decomposition, optional learned residual mapping, and cluster-level

tool-gain projection. We first construct an uncalibrated prior:

$$\tilde{z}_{\text{off}}(x) = \text{clip}_{[0,1]^d}\left(\lambda_r g_{\text{rule}}(\bar{\phi}_x, \Delta_{\text{tool}}, \Delta_{\text{evo}}, \Delta_{\text{clus}}) + (1-\lambda_r) f_\theta(\bar{\phi}_x) + \lambda_u A \Delta_{\text{clus}}(x)\right). \quad (5)$$

Here, g_{rule} maps interpretable trajectory diagnostics and tool-view evidence into the requirement space; f_θ is an optional learned residual mapper; λ_r balances the rule-based and learned branches; λ_u controls the strength of tool-gain projection; and $A \in \mathbb{R}^{d\times K}$ projects cluster gains into the d-dimensional requirement space. After optional calibration, we denote the final offline requirement prior as $z_{\text{off}}(x) \in [0,1]^d$. This prior is used for subsequent online correction and scheduling.

Offline reliability. Offline reliability measures the trustworthiness of the offline requirement profile. We estimate $r_{\text{off}}(x) \in [0,1]$ from probing stability statistics, including cross-view performance variance and feature-level variance. A compact form is

$$r_{\text{off}}(x) = \sigma\left(q_{\text{rel}}(\bar{\phi}_x, \text{EnvVar}(x))\right), \quad (6)$$

where $\text{EnvVar}(x)$ denotes the variance of normalized performance across core tool views, q_{rel} is a reliability scoring function, and σ is the sigmoid function. This score calibrates the trust placed in the offline prior during online correction.

Requirement-space partitioning for controller initialization. To provide a stable control interface for the scheduler, we initialize partitions using a scalar projection of the offline prior: $s_{\text{off}}(x) = w_{\text{off}}^\top z_{\text{off}}(x)$. Based on the quantiles of $s_{\text{off}}(x)$ over the training set, we divide examples into B partitions and denote $b(x) \in \{1,\dots,B\}$ as the partition index of example x. The set of examples in partition b is $\mathcal{X}_b = \{x \in \mathcal{X}_{\text{train}} : b(x) = b\}$. The corresponding offline partition prototype is $c_b^{\text{off}} = \frac{1}{|\mathcal{X}_b|}\sum_{x\in\mathcal{X}_b} z_{\text{off}}(x)$. This partitioning is used only as a controller interface; the underlying representation remains the multidimensional requirement vector. The offline probing budget and its computational overhead are detailed in Appendix C.

3.2 Online Capability and Requirement Recalibration

Online capability state estimation. At training step t, we collect online evaluation records S_t. For each $x \in S_t$, we extract $\phi_t(x) = \Phi(\tau_t(x))$, and obtain capability evidence $e_t(x) = h_\psi(\phi_t(x), P_t(x), \text{Conf}_t(x)) \in [0,1]^d$, where $P_t(x)$ denotes mean online performance. The estimator combines rule-based and learned evidence; implementation details are provided in Appendix A. The evidence weight is $\omega_t(x) = \exp(-\kappa_v \,\text{Var}_t(x))\cdot \text{Cons}_t(x)\cdot \text{Conf}_t(x)$, where $\text{Var}_t(x)$, $\text{Cons}_t(x)$, and $\text{Conf}_t(x)$ denote reward variance, within-group consistency, and confidence, and κ_v controls the variance penalty. With normalized weight $\tilde{\omega}_t(x)$, the capability state is updated by EMA:

$$a_{t+1} = (1-\beta)a_t + \beta \sum_{x\in S_t} \tilde{\omega}_t(x) e_t(x), \quad (7)$$

where $\beta \in [0,1]$ is the update rate.

Policy-conditioned requirement correction. Given the current capability state a_t, we estimate the policy-conditioned online requirement representation: $z_{\text{on},t}(x) = u_\xi(\phi_t(x), a_t)$. Here, $\phi_t(x)$ describes online trajectory behavior and a_t provides policy capability context. We define the offline-to-online trust gate: $\alpha_t(x) = \text{clip}_{[0,1]}\left(\alpha_0 + \alpha_1(1 - r_{\text{off}}(x)) + \alpha_2 \,\text{Instab}_t(x)\right)$. Here, $r_{\text{off}}(x)$ calibrates trust in the offline prior, while $\text{Instab}_t(x)$ captures online evidence instability. The scheduler-facing conditional representation is

$$z_{\text{cond},t}(x) = (1-\alpha_t(x)) z_{\text{off}}(x) + \alpha_t(x) z_{\text{on},t}(x). \quad (8)$$

Budgeted local recalibration. To avoid recalibrating all examples at every step, the controller updates only a local subset $R_t \subseteq \mathcal{X}_{\text{train}}$, including high-priority candidates, unstable examples, and boundary-near examples when enabled. For each partition b, let $R_{b,t} = R_t \cap \mathcal{X}_b$. If $R_{b,t} \neq \emptyset$, we average the online representations: $\hat{c}_{b,t}^{\text{on}} = \frac{1}{|R_{b,t}|}\sum_{x\in R_{b,t}} z_{\text{on},t}(x)$. The current partition prototype is

$$c_{b,t} = (1-\bar{\alpha}_{b,t}) c_b^{\text{off}} + \bar{\alpha}_{b,t} \hat{c}_{b,t}^{\text{on}}, \quad (9)$$

where $\bar{\alpha}_{b,t} = \frac{1}{|R_{b,t}|}\sum_{x\in R_{b,t}} \alpha_t(x)$. If $R_{b,t} = \emptyset$, we retain $c_{b,t} = c_b^{\text{off}}$.

3.3 Scheduler-Shaped Training Distribution and GRPO

New-sample allocation via partition-level distribution shaping. The scheduler converts controller state into a training distribution. For partition b, the requirement–capability gap is $g_{b,t} = \|a_t - c_{b,t}\|_W$, where W defines the weighted L_2 distance. Recent utility and stabilizing pressure are $U_{b,t} = P_{b,t} - P_{b,t-\Delta}, \qquad F_{b,t} = \max(P_b^{\text{best}} - P_{b,t}, 0)$. Here, $P_{b,t}$ is the average verifier-derived performance of partition b in the current online evaluation, Δ is the evaluation interval, and P_b^{best} is the best historical partition performance. The scheduler score is $S_{b,t} = -(g_{b,t} - g_t^\star)^2 + \lambda_U U_{b,t} - \lambda_F F_{b,t}$, where $g_t^\star$ avoids always selecting the largest-gap partition, and λ_U, λ_F weight utility and stabilization. The partition-level distribution is $p_t^{\text{new}}(b) = \frac{\exp(S_{b,t}/\tau_b)}{\sum_{b'=1}^{B} \exp(S_{b',t}/\tau_b)}$. The sample-level distribution

$$p_t(x) = p_t^{\text{new}}(b(x)) q_t(x \mid b(x)). \tag{10}$$

By default, $q_t(x \mid b) = \frac{1}{|\mathcal{X}_b|} \quad \text{if } x \in \mathcal{X}_b$.

GRPO over the shaped distribution. At training step t, the controller assigns task weights $p_t(x)$ to guide batch construction. The batch constructor prioritizes higher-weight tasks within each domain while maintaining task coverage through coverage cycles and interleaving tasks across domains. Weighted random sampling is used as a fallback when needed. For each selected task x, the policy generates G rollouts: $\{\tau_g(x)\}_{g=1}^G$, where G is the group size. The verifier maps each rollout to a training score: $R_g(x)$. The group-relative advantage is

$$A_g(x) = \frac{R_g(x) - \frac{1}{G}\sum_{g'=1}^{G} R_{g'}(x)}{\text{Std}_{g'=1}^{G}(R_{g'}(x)) + \epsilon}. \tag{11}$$

The subsequent update uses the standard clipped GRPO objective. The contribution of TReDS is not to redefine GRPO, but to run GRPO over the controller-shaped distribution $p_t(x)$. In implementation, we use fallback ranking signals and zero-signal filtering to improve training stability in sparse-reward settings. Algorithm 2 summarizes the controller-side update used in Sections 3.2–3.3.

4 Experiments

Training Data and Evaluation. TReDS uses only the τ^2-Bench training split for policy optimization, which contains 178 tasks across airline, retail, and telecom. We evaluate on a progressive protocol-shift suite: the τ^2-Bench official test, τ-Bench native-bridge evaluation, BFCL V3, and API Bank. We employ greedy decoding. The suite ranges from target-protocol evaluation to same-family behavior retention and broader changes in tool schema, calling format, and evaluation logic. We use the corresponding official evaluator or evaluation protocol for τ^2-Bench, BFCL V3, and API Bank, while the τ-Bench native-bridge maps model-generated trajectories into the τ-Bench evaluation interface. No benchmark-specific fine-tuning is performed, so gains over GRPO test whether TReDS preserves reusable tool-use behavior beyond τ^2-specific adaptation. More details are provided in the Appendix Section A.

Training Design. We use a limited interaction budget to evaluate the effectiveness of training distribution shaping. All RL-trained methods use 120 optimizer updates, with eight task groups per update and four trajectories per group, yielding 960 retained groups and 3,840 retained training trajectories per run. The retained-update budget is matched across methods, while additional trajectories generated by rejection-based sampling are accounted for separately. We evaluate Qwen3-8B and Qwen3-4B using GRPO optimization. Training rollouts use stochastic sampling, whereas evaluation uses greedy decoding. Results are averaged over three random seeds, with mean and standard deviation reported. Training configurations are detailed in Appendix A, while the implementations and budget accounting of additional sampling baselines are provided in Appendix E.

4.1 Main Results

TReDS turns distribution shaping into official gains. Table 1 shows TReDS consistently improves over GRPO across both 4B and 8B model scales. For the 4B base model, GRPO improves τ^2 Official Overall from 16.19% to 16.59%, while TReDS further reaches 17.28%. The gain is larger

Table 1: τ^2-Bench results. We report standalone official test performance and project-internal native-bridge diagnostics. Best results within each base model group are highlighted in **bold**.

Model	Method	Official Test (%)				Native-Bridge (%)			
		Overall	Airline	Retail	Telecom	Bridge Verified	Airline	Retail	Telecom
Qwen3-4B	Base	16.19 0.43	22.34 0.67	3.05 0.47	**25.94** 0.57	30.93 1.07	52.63 0.14	35.00 0.58	15.79 0.79
	GRPO	16.59 0.52	26.72 1.33	**3.98** 0.54	24.06 0.48	47.31 0.97	73.68 0.09	53.85 0.41	25.71 0.94
	TReDS	**17.28** 0.46	**27.03** 0.82	3.91 0.61	25.86 0.66	**53.76** 1.10	**84.21** 0.05	**64.86** 0.35	**27.03** 1.03
Qwen3-8B	Base	19.72 0.25	17.97 1.15	21.25 0.33	19.06 0.44	36.83 0.42	61.67 0.74	52.91 0.41	8.33 0.54
	GRPO	20.09 0.47	16.88 1.52	23.34 0.49	19.45 0.51	53.76 1.01	84.21 0.45	61.54 0.87	28.57 0.98
	GRPO + Static Curriculum	20.22 0.34	17.50 0.84	22.66 0.36	19.14 0.48	–	–	–	–
	GRPO + Learning-Progress	20.19 0.61	18.13 0.58	21.95 0.47	19.45 0.69	–	–	–	–
	GRPO + DAPO-style	20.47 0.48	17.89 0.36	23.71 0.26	18.48 0.41	–	–	–	–
	GRPO + ToolSample/RDS-style	20.92 0.49	**18.83** 0.91	21.80 0.54	21.02 0.53	–	–	–	–
	TReDS	**22.03** 0.53	16.41 0.80	**24.30** 1.08	**22.58** 0.97	**58.06** 0.74	**85.00** 0.69	**73.53** 0.73	**30.77** 0.76

at the 8B scale, where GRPO improves the base model from 19.72% to 20.09%, and TReDS further improves it to 22.03%. These results suggest that trajectory-grounded requirement modeling and capability-conditioned scheduling make the limited RL budget more effective by allocating updates to more informative regions of the training distribution.

TReDS improves along learnable capability gaps. As Table 1 shows, Domain-level results show that TReDS does not act as a uniform domain-level booster. On the 8B model, TReDS improves retail from 23.34% to 24.30% and telecom from 19.45% to 22.58% over GRPO, while airline remains close to GRPO with a mild decrease from 16.88% to 16.41%. This pattern is consistent with requirement-based allocation: because requirement regions do not perfectly align with dataset domains, gains may concentrate in some domains while producing small domain-level fluctuations.

TReDS improves verifier-facing trajectory outcomes. The native-bridge diagnostics in Table 1 further provide a trajectory-oriented view. Compared with GRPO on the 8B model, TReDS improves Internal Official Verified performance and also improves the bridge diagnostics on retail and telecom. This indicates that TReDS improves not only the final official aggregate score, but also verifier-facing trajectory outcomes.

Table 2: τ-Bench zero-shot transfer result: For this experiment, model-generated trajectories on τ-Bench are mapped into the τ-Bench evaluation interface to produce the evaluation report.

Method	Overall local reward	Airline reward	Retail reward
Qwen3-8B Base	20.92 0.71	28.01 0.35	17.84 0.38
Qwen3-8B + GRPO	21.46 0.59	28.79 0.42	18.27 1.17
GRPO + Static Curriculum	21.21 0.42	27.00 0.33	18.70 0.17
GRPO + Learning-Progress	21.06 0.78	27.75 0.75	18.15 0.84
GRPO + DAPO-style	23.05 0.64	**30.26** 0.61	18.91 0.61
GRPO + ToolSample/RDS-style	22.42 0.75	26.26 0.37	20.77 0.45
Qwen3-8B + TReDS	**23.45** 0.64	23.89 0.41	**23.26** 0.81

Stronger same-family transfer. Table 2 shows that GRPO brings only a modest improvement over the base model, increasing the overall local reward from 20.92% to 21.46%. TReDS yields a larger gain, reaching 23.45% overall local reward. This indicates that the gains of TReDS are not tied solely to the τ^2 official evaluator: under τ-native scoring, its distribution shaping still translates into stronger performance, suggesting better retention of tool-use behavior under a related but different protocol.

TReDS preserves cross-protocol competence. Table 3 shows that GRPO does not automatically transfer τ^2-training gains to BFCL V3 and API Bank. Compared with the base model, GRPO only marginally improves Non-Live AST Accuracy from 87.96% to 87.98%, while decreasing Multi-Turn from 37.50% to 37.12% and Live from 80.01% to 79.16%. In contrast, TReDS outperforms

Table 3: Zero-shot transfer performance on BFCL V3 and API Bank benchmarks.

Method	BFCL V3 zero-shot transfer (%)			API Bank zero-shot transfer (%)		
	Multi-Turn	Non-Live AST Acc	Live Acc	L1 API Call Acc	L2 API Call Acc	L3 API Call Acc
Qwen3-8B Base	37.50 0.09	87.96 0.79	80.01 0.04	65.30 0.17	32.77 0.51	34.29 0.61
Qwen3-8B + GRPO	37.12 0.19	87.98 0.32	79.16 0.25	64.01 0.06	32.77 1.04	34.29 0.20
GRPO + Static Curriculum	37.54 0.16	88.20 0.21	79.33 0.11	63.87 0.21	32.59 0.04	34.18 0.46
GRPO + Learning-Progress	38.28 0.12	88.16 0.32	80.15 0.12	65.33 0.15	32.59 0.13	34.55 0.29
GRPO + DAPO-style	38.33 0.26	88.23 0.15	80.12 0.32	65.50 0.13	32.74 0.46	34.18 0.31
GRPO + ToolSample/RDS-style	38.15 0.15	**88.33** 0.17	80.11 0.15	**65.58** 0.23	32.88 0.27	34.29 0.15
Qwen3-8B + TReDS	**38.62** 0.28	88.23 0.10	**80.38** 0.17	65.30 0.27	**43.70** 0.64	**52.24** 1.12

GRPO across all BFCL V3 groups, reaching 38.62% on Multi-Turn. On API Bank, TReDS preserves L1 API Call Accuracy at 65.30% while substantially improving harder levels: L2 rises from 32.77% to 43.70%, and L3 from 34.29% to 52.24%. Since all checkpoints are trained only on τ^2, these results suggest that TReDS does not merely reinforce τ^2-specific templates. Instead, requirement-conditioned distribution shaping mitigates GRPO's cross-protocol drift and directs limited updates toward tool-use and API-use bottlenecks that remain useful under protocol shifts.

4.2 Ablation Study

Ablation on the closed-loop controller. Table 4 supports the central claim that TReDS gains arise from the closed-loop combination of online correction, reliability-gated fusion, and scheduler-based allocation. This ablation tests whether the gains of TReDS depend on the full closed-loop distribution-shaping mechanism. Removing online correction, the reliability gate, or the scheduler consistently lowers performance on τ^2-Bench, BFCL V3, and API Bank. The results show that a static offline prior cannot fully adapt to changing policy capability, online evidence must be calibrated by r_{off}, z_{off}, a_t, and $z_{\text{cond},t}$ affect GRPO only when converted into the training distribution $p_t(x)$.

Ablation on requirement modeling assumptions. Table 4 shows that TReDS does not benefit from an arbitrary curriculum signal, but from a structured requirement space that directs limited training budget toward the current policy's learnable bottlenecks.This ablation tests whether TReDS relies on a trajectory-grounded, multidimensional, and tool-aware requirement space. Removing offline profiling, using a scalar-only representation, or removing the tool-aware branch consistently lowers performance on τ^2-Bench, BFCL V3, and API Bank. The results show that online evidence alone cannot bootstrap an effective controller from a flat prior, a single difficulty score cannot capture differences across planning, retrieval, tool use, verification, and long-horizon control, and structured tool views with tool-gain signals provide important evidence for requirement modeling.

Table 4: Ablation study of TReDS. We report the overall performance across τ^2-Bench, BFCL V3, and API Bank. Best results are highlighted in **bold**.

Category	Method	τ^2-Bench Official Overall (%)	BFCL V3 Overall (%)	API Bank Overall (%)
	Qwen3-8B + TReDS	**22.03** 0.53	**75.31** 0.14	**57.64** 0.39
Closed-loop Controller	w/o Online Correction	20.17 0.11	74.83 0.08	54.36 0.15
	w/o Reliability Gate	18.25 1.08	74.91 0.16	56.51 0.57
	w/o Scheduler	18.38 0.14	74.82 0.05	49.71 0.63
Requirement Modeling Assumptions	w/o Offline Profiling	20.28 0.73	74.96 0.24	55.78 0.11
	Scalar-only	20.86 0.44	74.87 0.02	55.64 0.08
	w/o Tool-aware Branch	20.06 0.51	74.91 0.31	53.52 0.17

5 Analysis

We analyze project-internal diagnostics collected during TReDS training on the τ^2-Bench training split, using Qwen3-8B as the base model. More detailed results are provided in Appendix Section G.

5.1 Multidimensional Requirement Structure Beyond Scalar Difficulty

TReDS initializes examples into coarse requirement regions using the scalar projection $s_{\text{off}}(x)$; here we use 4 regions and average the sample-level $z_{\text{off}}(x)$ vectors to obtain region-level requirement prototypes. Figure 3 (left) shows that these regions capture distinct multidimensional capability-demand profiles rather than scalar difficulty levels. By comparing them with the current capability state a_t, TReDS identifies capability gaps that remain learnable for the current policy. Figure 3 (middle) further shows that two examples with nearly identical s_{off} values, 0.5372 and 0.5376, can have substantially different 7-dimensional requirement vectors, with $\|z_A - z_B\|_2 = 1.162$. Thus, a one-dimensional curriculum would obscure these differences, whereas TReDS converts them into scheduling signals for allocating training budget toward learnable bottlenecks.

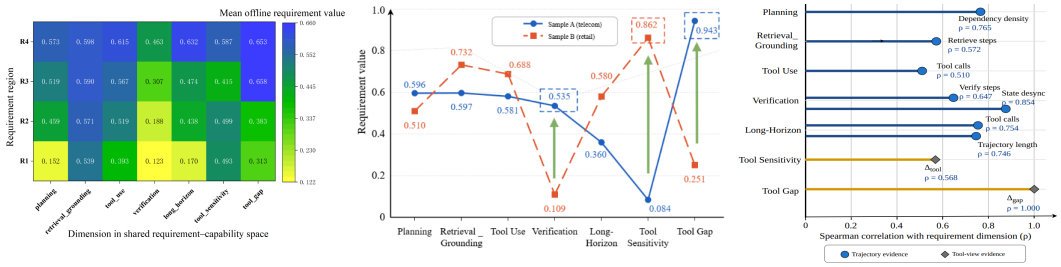

Figure 3: Analysis of the Multidimensional Requirement Structure and the Requirement Dimension

5.2 Requirement Dimensions Align with Behavioral and Tool-View Evidence

Figure 3(right) validates whether the constructed requirement dimensions are grounded in their intended evidence sources. We compute Spearman correlations between each dimension of $z_{\text{off}}(x)$ and trajectory- or tool-view evidence features, and report the strongest interpretable anchor for each dimension. The results show that structural dimensions align with trajectory-side behavioral signals, while tool-aware dimensions align with controlled tool-view deltas. This confirms that z_{off} is not an arbitrary representation, but a trajectory-grounded and tool-aware requirement space.

5.3 Scheduler Targets Learnable Gaps

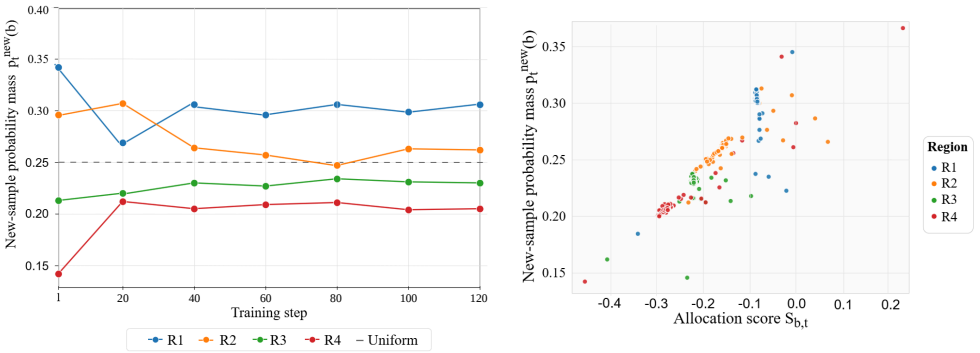

Figure 4: Analysis of Dynamic Scheduling Behavior.

To test whether controller-side estimates are converted into training behavior, we read scheduler snapshots and extract $p_t^{\text{new}}(b)$, $S_{b,t}$, and $g_{b,t}$ for each requirement region at every training step. Fig-

ure 4 (left) shows that TReDS dynamically reallocates probability mass: R1 later stabilizes around 0.30, while R4 increases from 0.142 to around 0.21 but remains below the uniform baseline of 0.25. Thus, TReDS is not a fixed easy-to-hard curriculum, but adjusts training exposure as the controller state changes. Figure 4 (right) shows monotonic alignment between $S_{b,t}$ and $p_t^{\mathrm{new}}(b)$, indicating that the score is the interface through which controller estimates become sampling probabilities. The scheduler is also not hardest-first: R4 has the largest mean raw gap (0.708) but the lowest mean mass (0.216), while R1 has a lower mean gap (0.548) but the highest mean mass (0.296). This suggests that the scheduler combines gap, utility, and stabilizing signals to prioritize currently more valuable training regions.

6 Related Work

Recent benchmarks move LLM evaluation from single-turn generation to verifiable interactive agents that use tools, update state, and are judged by executable or task-specific evaluators [49, 22, 35, 52, 10, 11, 38, 46, 33], including ToolLLM [26], WebArena [51, 4], SWE-bench [12], AppWorld [34], τ-Bench [48], τ^2-Bench [1], BFCL [25], and API-Bank [15]. TReDS builds on this setting but studies the training problem: how trajectory diagnostics should shape the RL training distribution. RL from verifiable, process-level, or execution feedback improves reward signals or optimization, as in GRPO, Math-Shepherd, and RLEF [32, 37, 8]. TReDS is orthogonal: it keeps the standard GRPO objective and changes the distributional interface over which GRPO is applied [45]. TReDS is also related to tool learning and data scheduling [31, 24]. Unlike tool-only adaptation, influence-based selection, or scalar curricula [21, 41, 3, 44], TReDS constructs a shared requirement–capability space from trajectory and tool-view evidence, updates policy capability online, and converts requirement–capability mismatch into a time-varying training distribution $p_t(x)$ [29, 23, 17].

7 Conclusion

This paper presented TReDS, a trajectory-grounded framework for capability-conditioned training distribution shaping in verifiable tool-interactive agent tasks. TReDS estimates multidimensional task requirements from offline probing trajectories and tool-view evidence, tracks policy capability in the same semantic space, and converts requirement–capability mismatch into a scheduler-shaped GRPO training distribution. Across τ^2-Bench, τ-Bench native-bridge evaluation, BFCL V3, and API Bank, TReDS outperforms GRPO under the same training data and backend, improving target-protocol performance while preserving reusable tool-use behavior under protocol shifts. These results suggest that agent RL post-training should consider not only how trajectories are rewarded, but also where trajectory-revealed requirements allocate training probability mass.

Acknowledgments and Disclosure of Funding

This research was self-funded by the author. The author received no third-party funding or in-kind support for this work and declares no competing interests.

References

[1] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

[2] Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, and Kai Chen. T-eval: Evaluating the tool utilization capability of large language models step by step. 2023.

[3] Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Yuchen Zhang, Jiacheng Chen, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.

[4] Thibault Le Sellier De Chezelles, Maxime Gasse, Alexandre Drouin, Massimo Caccia, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, and Lawrence Keunho Jang. The browsergym ecosystem for web agent research. 2024.

[5] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

[6] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024.

[7] Zihao Feng, Xiaoxue Wang, Bowen Wu, Hailong Cao, Tiejun Zhao, Qun Yu, and Baoxun Wang. Toolsample: Dual dynamic sampling methods with curriculum learning for rl-based tool learning, 2025. URL `https://arxiv.org/abs/2509.14718`.

[8] Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Quentin Carbonneaux, Taco Cohen, and Gabriel Synnaeve. Rlef: Grounding code llms in execution feedback with reinforcement learning. *arXiv preprint arXiv:2410.02089*, 2024.

[9] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[10] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. 2024.

[11] Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, et al. Metatool benchmark for large language models: Deciding whether to use tools and which to use. *arXiv preprint arXiv:2310.03128*, 2023.

[12] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

[13] Pascal Klink, Carlo D'Eramo, Jan R Peters, and Joni Pajarinen. Self-paced deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:9216–9227, 2020.

[14] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905, 2024.

[15] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 3102–3116, 2023.

[16] Renda Li, Hailang Huang, Fei Wei, Feng Xiong, Yong Wang, and Xiangxiang Chu. Adacurl: Adaptive curriculum reinforcement learning with invalid sample mitigation and historical revisiting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 23123–23131, 2026.

[17] Zhenghao Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, et al. Rho-1: Not all tokens are what you need. *arXiv preprint arXiv:2404.07965*, 2024.

[18] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

[19] Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1160–1183, 2025.

[20] Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. 2024.

[21] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *IEEE Transactions on Neural Networks and Learning Systems*, PP(99), 2017.

[22] Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann Lecun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. 2023.

[23] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50, 2020.

[24] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37: 126544–126565, 2024.

[25] Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, 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.

[26] 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. *arXiv preprint arXiv:2307.16789*, 2023.

[27] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343, 2025.

[28] Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024.

[29] Ayrton San Joaquin, Bin Wang, Zhengyuan Liu, Nicholas Asher, Brian Lim, Philippe Muller, and Nancy Chen. In2core: Leveraging influence functions for coreset selection in instruction finetuning of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10324–10335, 2024.

[30] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[31] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

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

[33] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

[34] Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16022–16076, 2024.

[35] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. 2023.

[36] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

[37] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, 2024.

[38] Wenxuan Wang, Shi Juluan, Zixuan Ling, Yuk-Kit Chan, Chaozheng Wang, Cheryl Lee, Youliang Yuan, Jen-tse Huang, Wenxiang Jiao, and Michael R Lyu. Learning to ask: When llm agents meet unclear instruction. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 21784–21795, 2025.

[39] Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691*, 2023.

[40] Xumeng Wen, Zihan Liu, Shun Zheng, Shengyu Ye, Zhirong Wu, Yang Wang, Zhijian Xu, Xiao Liang, Junjie Li, Ziming Miao, et al. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base llms. *arXiv preprint arXiv:2506.14245*, 2025.

[41] Lijun Wu, Fei Tian, Yingce Xia, Yang Fan, Tao Qin, Jianhuang Lai, and Tie Yan Liu. Learning to teach with dynamic loss functions. 2018.

[42] Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. Less: Selecting influential data for targeted instruction tuning. 2024.

[43] Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*, 2024.

[44] Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy S Liang, Quoc V Le, Tengyu Ma, and Adams Wei Yu. Doremi: Optimizing data mixtures speeds up language model pretraining. *Advances in Neural Information Processing Systems*, 36:69798–69818, 2023.

[45] Sang Michael Xie, Shibani Santurkar, Tengyu Ma, and Percy Liang. Data selection for language models via importance resampling. 2023.

[46] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

[47] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

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

[49] Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416*, 2025.

[50] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL `https://arxiv.org/abs/2503.14476`.

[51] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

[52] Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. Toolqa: A dataset for llm question answering with external tools. 2023.

A Parameters

Table 5 summarizes the main experimental settings.

Category	Configuration
Policy model	Qwen3-4B / Qwen3-8B
Probe model	Qwen3-30B-A3B
Training backend	Local HuggingFace backend
Probe backend	Remote OpenAI-compatible backend
Precision	float16
Training max completion tokens	320
Training temperature	0.7
Training top-p	0.8
Thinking mode	Disabled
GPUs	$4\times$ RTX 5090, 32 GB each
Training time per run	Within 10 hours for Qwen3-8B
RL objective	GRPO
Learning rate	2×10^{-6}
Optimizer updates	120
Task groups per update	8
Rollouts per group	4
Evaluation group size	4
Checkpoint interval	6
Fine-tuning method	LoRA
LoRA rank	8
LoRA alpha	16
LoRA dropout	0.05
Evaluation max completion tokens	320
Evaluation temperature	0.0
Evaluation top-p	1.0

Table 5: Main configurations for experiments and ablation studies.

Capability estimator and controller settings. The capability estimator takes a 37-dimensional trajectory feature vector as input and uses a shared MLP with three hidden layers of widths 128, 128, and 64. Layer normalization follows the first linear layer, and GELU activations are applied after each hidden layer. Two output heads with sigmoid activations produce seven-dimensional vectors v_ψ and c_ψ, yielding $\hat{e}_\psi = v_\psi \odot (0.65 + 0.35 c_\psi)$. The estimator contains 30,798 trainable parameters. The final capability evidence is an equally weighted combination of the learned prediction and rule-based evidence.

The capability state is initialized to zero, and the predictor uses the default parameter initialization in PyTorch. The predictor is trained on rule-generated targets using Adam with a learning rate of

10^{-3}, with 60 full-batch optimization steps for offline fitting and 20 steps for each online refit. The training objective combines the mean squared error between the predicted and target vectors with an additional term, weighted by 0.1, measuring the mean squared error between their averages across capability dimensions.

The controller uses gate coefficients $(\alpha_0, \alpha_1, \alpha_2) = (0.15, 0.55, 0.45)$, an EMA update rate of $\beta = 0.35$, a variance penalty coefficient of $\kappa_v = 2.5$, and scheduler parameters $(\lambda_U, \lambda_F, \tau_b) = (0.5, 0.7, 0.5)$. The target gap adapts to the current capability state:

$$g_t^\star = \mathrm{clip}_{[0,1.5]}\left(0.22 + 0.28 w_s^\top a_t\right), \tag{12}$$

where $w_s = (0.18, 0.16, 0.16, 0.14, 0.12, 0.12, 0.12)$. The weighted distance is defined as $\|a - c\|_W = \sqrt{(a-c)^\top W (a-c)}$, with $W = \mathrm{diag}(1, 1, 1, 1, 0.9, 0.8, 0.8)$. Replay is disabled.

B Offline Taxonomy Construction

B.1 Number and Partitioning of Tool Clusters

Initial Induction. We divide the τ^2-Bench training tasks into an 80% discovery set and a 20% internal validation set through domain-stratified sampling, excluding the official test set. DeepSeek-V3.2 extracts atomic tool attributes from function schemas, parameters, return values, state access, invocation prerequisites, side effects, policy constraints, and usage contexts. Without predefined cluster labels or a predefined K, it groups tools by functional similarity, state interaction, execution role, and cross-task usage patterns.

Consolidation and Held-Out Review. We manually examine cluster coherence and overlap, merging, splitting, or reassigning tools with ambiguous boundaries. The preliminary taxonomy is then reviewed by DeepSeek-V3.2 on held-out training tasks without access to previous assignments or induction rationales. The review examines coverage, multiple membership, uncovered functions, redundancy, and boundary clarity.

Final Taxonomy. We resolve the remaining ambiguities by introducing clusters for repeatedly uncovered functions or consolidating persistently overlapping clusters. This procedure yields $K = 5$ functional clusters: retrieval, planning, verification, execution, and memory. The resulting taxonomy is fixed for all subsequent training and evaluation. Neither K nor the cluster labels were prespecified, and downstream test performance was not used to select them. We did not evaluate alternative values of K through retraining and therefore do not claim optimality or insensitivity to K.

B.2 Induction of Requirement-Space Dimensions

Candidate Dimension Induction. We use the same domain-stratified 80/20 discovery-validation protocol on the τ^2-Bench training tasks. On the discovery set, DeepSeek-V3.2 extracts evidence-backed atomic requirements from user goals, domain policies, tool schemas, initial states, interaction roles, information dependencies, action prerequisites, and state transitions. Candidate dimensions are induced from semantic similarity and cross-task co-occurrence without predefined dimension names or a predefined dimension count.

Consolidation and Validation. We merge synonymous requirements, remove superficial domain-specific features, and distinguish task requirements from model-specific errors. Co-occurrence patterns are used to identify redundant or excessively fragmented dimensions. DeepSeek-V3.2 then reviews candidate definitions on held-out training tasks without prior assignment information, using different prompts, randomized input orders, and multiple runs. We further examine representative Qwen3-30B-A3B offline probing trajectories to identify recurring behavioral requirements and uncovered failure modes.

Final Requirement Space. Based on the combined evidence, we finalize seven dimensions: planning, retrieval/grounding, tool use, verification, long-horizon consistency, tool sensitivity, and tool gap. The names and number of dimensions were not fully prespecified but emerged from training-data analysis, manual consolidation, and validation. The resulting taxonomy is fixed before subsequent policy optimization and evaluation.

C Offline Profiling and Computational Overhead

C.1 Offline Probing Budget

Each of the 178 training tasks produces ten offline probing trajectories: one coarse probe, three view-specific probes, one evolution probe, and five cluster-specific probes. With $K = 5$, the total number of offline trajectories is

$$N(1 + 3 + 1 + K) = 178 \times 10 = 1,780. \tag{13}$$

With at most five interaction steps per trajectory, offline trajectory generation has an upper bound of 8,900 chat completions, plus at most 178 additional tool-evolution calls. Cached feature extraction requires no additional probe-model calls.

D Algorithm

The overall procedure of TReDS is illustrated in Algorithm 1, and Algorithm 2 summarizes the controller-side update used in Sections 3.2–3.3.

E Additional Baseline Implementations

E.1 Comparison Protocol and Budget Accounting

All RL-trained methods use 120 optimizer updates, retaining eight task groups per update and four trajectories per group, for a total of 960 retained groups and 3,840 retained training trajectories per run. They use the same GRPO optimization objective unless otherwise specified. This protocol matches the retained training budget, not necessarily the total trajectory-generation cost: preliminary difficulty-labeling probes and rejected candidate groups are accounted for separately. The DAPO-style and ToolSample/RDS-style baselines isolate their sampling components rather than reproducing all components of the original methods.

E.2 Sampling Baselines

Static Curriculum. Qwen3-30B-A3B generates ten pre-training rollouts per task to estimate difficulty. Within each domain, tasks are divided into fixed easy, medium, and hard thirds. The 120 optimizer updates are divided into three consecutive 40-update phases with easy/medium/hard group-sampling ratios of 5/2/1, 2/4/2, and 1/2/5, respectively. Each difficulty level receives 320 retained task groups (1,280 trajectories) across the run. The preliminary rollouts are used only for difficulty labeling and are not part of the retained RL training trajectories.

Learning-Progress. Previously unseen tasks are sampled first. For each task i, competence is estimated using the mean reward of four rollouts. Fast and slow competence estimates use exponential moving average (EMA) coefficients of 0.5 and 0.1, respectively. Learning progress is $LP_i = |c_i^{\text{fast}} - c_i^{\text{slow}}|$. The subsequent task-sampling probability is

$$p_i = 0.8 \frac{LP_i + 10^{-3}}{\sum_k (LP_k + 10^{-3})} + \frac{0.2}{N}, \tag{14}$$

where N is the number of training tasks. This baseline changes task sampling without modifying reward definitions, group-relative advantages, or loss weights.

DAPO-style Dynamic Sampling. The full DAPO [50] method also modifies clipping, loss reduction, and overlong-reward shaping. To isolate dynamic sampling under the common GRPO objective, a candidate task group is rejected when its within-group reward standard deviation is below 10^{-6}. Sampling continues until eight effective groups have been retained for each optimizer update. Each run therefore retains 3,840 training trajectories, while any additional trajectories generated for rejected groups are recorded separately and are not included in the retained count.

Algorithm 1 TReDS: Scheduler-Shaped GRPO Training

Require: Training set $\mathcal{X}_{\text{train}}$, evaluation set $\mathcal{X}_{\text{eval}}$, probing tool views $\mathcal{U}_{\text{probe}}$;
probe policy π_{probe}, trainable policy π_θ, verifier $\mathcal{V}$;
number of partitions B, training steps N_{train}, group size G.

Ensure: Trained policy π_θ.

Offline profiling.

for all $x \in \mathcal{X}_{\text{train}}$ **do**
  Roll out π_{probe} under multiple tool views $U \in \mathcal{U}_{\text{probe}}(x)$.
  Extract trajectory diagnostics and tool-view performance evidence.
  Estimate offline requirement prior $z_{\text{off}}(x)$ and reliability $r_{\text{off}}(x)$.
end for
Partition examples by $s_{\text{off}}(x) = w_{\text{off}}^\top z_{\text{off}}(x)$ into $\{\mathcal{X}_b\}_{b=1}^B$.
Initialize offline prototypes $c_b^{\text{off}} = \frac{1}{|\mathcal{X}_b|} \sum_{x \in \mathcal{X}_b} z_{\text{off}}(x)$.
Initialize capability state a_0 and online cache $\mathcal{C}_{\text{on}}$.

Scheduler-shaped GRPO training.

for $t = 1, \ldots, N_{\text{train}}$ **do**
  Propose recalibration set R_t and capability-update set S_t from the current controller state and online cache $\mathcal{C}_{\text{on}}$.
  Collect online trajectories for $R_t \cup S_t$ using current policy π_θ, and score them with verifier $\mathcal{V}$.
  Extract online features $\phi_t(x)$, compute capability evidence $e_t(x)$, and update capability state a_t by weighted EMA over S_t.
  for all $x \in R_t$ **do**
    Estimate online requirement representation $z_{\text{on},t}(x)$.
    Compute trust gate $\alpha_t(x)$ from $r_{\text{off}}(x)$ and online instability.
    Form conditional requirement representation:

$$z_{\text{cond},t}(x) = (1 - \alpha_t(x)) z_{\text{off}}(x) + \alpha_t(x) z_{\text{on},t}(x).$$

  end for
  for $b = 1, \ldots, B$ **do**
    if $R_t \cap \mathcal{X}_b \neq \emptyset$ **then**
      Average $z_{\text{on},t}(x)$ and $\alpha_t(x)$ over $R_t \cap \mathcal{X}_b$, and fuse the online mean with c_b^{off} as specified in Section 3.2.
    else
      Set $c_{b,t} = c_b^{\text{off}}$.
    end if
    Compute $g_{b,t}$, $U_{b,t}$, $F_{b,t}$, and scheduler score $S_{b,t}$.
  end for
  Convert $\{S_{b,t}\}_{b=1}^B$ into partition-level new-sample distribution $p_t^{\text{new}}(b)$, inducing sample-level distribution $p_t(x)$.
  Construct training batch $\mathcal{B}_t$ using task weights $p_t(x)$ subject to the coverage constraints in Section 3.3.
  for all $x \in \mathcal{B}_t$ **do**
    Generate G rollouts $\{\tau_g(x)\}_{g=1}^G$ using π_θ.
    Score rollouts with verifier $\mathcal{V}$ and compute group-relative advantages.
  end for
  Update π_θ with the standard GRPO objective.
  Log scheduler snapshots, online diagnostics, and training statistics.
end for
Evaluate final policy π_θ on $\mathcal{X}_{\text{eval}}$.
return π_θ.

Algorithm 2 Budgeted Local Recalibration and Scheduler

Require: Partitions $\{\mathcal{X}_b\}_{b=1}^{B}$, offline priors z_{off}, reliabilities r_{off}, offline prototypes c_b^{off}, capability state a_t, online cache $\mathcal{C}_{\text{on}}$, performance history $P_{b,\cdot}$.

Ensure: Updated prototypes $c_{b,t}$, partition distribution $p_t^{\text{new}}(b)$, sample distribution $p_t(x)$.

Select recalibration set R_t and capability-update set S_t from the current controller state and online cache.
Collect or retrieve online trajectories for $R_t \cup S_t$, score them with the verifier, and update $\mathcal{C}_{\text{on}}$.
Update capability state a_t from weighted online evidence on S_t.
for all $x \in R_t$ **do**
    Estimate $z_{\text{on},t}(x)$, compute reliability-gated weight $\alpha_t(x)$, and form

$$z_{\text{cond},t}(x) = (1 - \alpha_t(x))z_{\text{off}}(x) + \alpha_t(x)z_{\text{on},t}(x).$$

end for
for $b = 1, \ldots, B$ **do**
    if $R_t \cap \mathcal{X}_b \neq \emptyset$ **then**
        Average $z_{\text{on},t}(x)$ and $\alpha_t(x)$ over $R_t \cap \mathcal{X}_b$, and fuse the online mean with c_b^{off} as specified in Section 3.2.
    else
        Set $c_{b,t} = c_b^{\text{off}}$.
    end if
    Compute $g_{b,t}$, $U_{b,t}$, $F_{b,t}$, and scheduler score $S_{b,t}$.
end for
Convert $\{S_{b,t}\}_{b=1}^{B}$ into $p_t^{\text{new}}(b)$.
Induce sample-level training distribution

$$p_t(x) = p_t^{\text{new}}(b(x))q_t(x \mid b(x)).$$

return $\{c_{b,t}\}_{b=1}^{B}$, $p_t^{\text{new}}(b)$, $p_t(x)$.

ToolSample/RDS-style Weighting. The full DSCL framework [7] additionally includes TCDL; here we isolate its RDS weighting component while preserving the common reward definition. After seven consecutive optimizer-step batches with mean reward above 0.25, groups with mean reward at least 0.999999 receive weight 0. For groups with mean reward below 0.125, the weight is 1 if either the current or historical reward variance exceeds 0.00625, and 0 otherwise. All remaining groups receive weight 1 if both variances exceed 0.00625, and 0.5 otherwise. No replacement groups are generated for downweighted groups. Thus, each run generates 960 task groups and 3,840 trajectories, although their effective gradient weights differ.

Fairness and Additional Sampling Costs. Matching 3,840 retained trajectories does not imply identical total generation or optimization signal across baselines. In particular, Static Curriculum uses separate pre-training difficulty probes, DAPO-style sampling may generate rejected groups beyond the retained budget, and ToolSample/RDS-style weighting changes the effective contribution of generated groups without refilling them. TReDS also incurs the offline probing workload reported in Appendix C. We therefore interpret these comparisons as matched retained-training-budget evaluations rather than equal-total-compute comparisons.

F Analysis of Recovery Warm-Start SFT

F.1 Construction of Recovery Warm-Start SFT Data

We additionally explored a supervised warm-start stage before RL. This stage was not used in the final main results, but we describe it here for completeness. The goal of the warm-start data construction was to synthesize high-quality tool-use trajectories that teach the base policy to follow strict action contracts, perform repair-and-verification behavior, and produce conservative final responses.

As Figure 5 shows, the pipeline consists of four stages. First, we select seed tasks from the τ^2-Bench training split and auxiliary training pools. Tasks are filtered by domain, capability family,

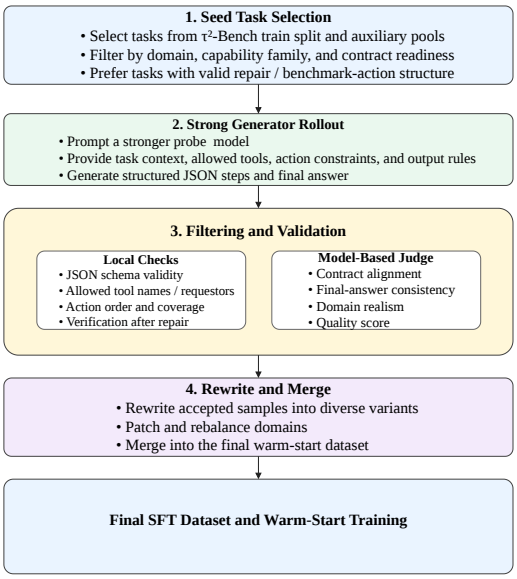

Figure 5: SFT Data Construction Pipeline.

and contract readiness. In particular, the selection logic favors examples for which the available tools and annotations define a valid repair or benchmark-action trajectory. Second, for each selected task, a stronger probe/generator model is prompted with the task context, allowed tools, required action constraints, verification requirements, and domain-specific output rules. The generator is required to return a compact JSON object with a sequence of structured steps and a final answer. Each step specifies the tool name, requestor, arguments, and notes.

Third, generated candidates are filtered by local contract checks and a model-based judge. The local checks verify JSON schema validity, explicit requestors, allowed tool names, action order, required repair or benchmark-action coverage, verification-after-repair behavior, and whether the final answer makes only supported success claims. The judge further evaluates contract alignment, closed-loop follow-up behavior, final-answer consistency, domain realism, usefulness for warm-starting, and a quality score. Only candidates that pass both local checks and the judge threshold are retained. Accepted examples are deduplicated and split into training and development sets.

Finally, the accepted data are further augmented and balanced through a rewrite-and-merge stage. Rewriting produces structurally distinct variants while preserving the underlying capability skeleton and avoiding leakage from the source instance. Additional patching and merging steps rebalance domains and capability families, especially for backend-state and explicit-recovery cases. In the final merged warm-start dataset used in our SFT exploration, the training split contains 54 examples and the development split contains 32 examples, spanning airline, retail, and telecom. The SFT model is trained with a masked completion loss over the target completions, using LoRA adaptation on the local Qwen3-8B policy.

F.2 SFT Warm-Start as an Auxiliary Negative Result

Although the recovery warm-start SFT stage improved several project-internal diagnostics, it did not improve the primary official evaluation. In our experiments, SFT produced small positive changes on internal proxy metrics such as schema validity, contract alignment, and recovery-style generation quality. However, when evaluated on the τ^2-Bench official test split, the SFT warm-start checkpoint slightly decreased the official overall reward compared with directly applying RL from the base policy. The domain-level official scores were also either slightly lower or approximately unchanged. Since the main objective of this paper is to improve official τ^2-Bench performance and

cross-protocol tool-use behavior, we exclude SFT from the final mainline and use direct GRPO post-training under the controller-shaped distribution.

This choice also keeps the experimental comparison cleaner. The final method therefore isolates the effect of trajectory-grounded requirement modeling and capability-conditioned distribution shaping, rather than mixing it with an additional supervised imitation stage whose gains are not reflected in the official benchmark. We keep the SFT pipeline as an auxiliary implementation detail and report it as a negative result rather than a main component.

F.3 Limited Effect of Warm-Start SFT

We hypothesize that the limited effect of warm-start SFT comes from a mismatch between supervised imitation and the final verifier-driven agent objective. The SFT data are generated and filtered to satisfy local trajectory contracts, JSON schemas, repair-verification patterns, and conservative final-answer rules. These signals improve local format and contract-following diagnostics, but they remain proxy objectives for official τ^2-Bench success. Consequently, optimizing token-level likelihood on these completions can improve internal metrics without improving the official task reward.

A second factor is the scale and structure of the warm-start dataset. The final merged dataset contains only dozens of accepted examples and is concentrated on recovery-style tool-use patterns and selected capability families. This makes SFT prone to overfitting to local templates, action orders, and conservative response styles. Such overfitting may reduce rollout diversity, which is undesirable for subsequent group-based RL where useful learning signals often depend on variation among sampled rollouts.

Third, the SFT trajectories are off-policy demonstrations produced by a generator and filtered by proxy judges. They may not match the trajectory distribution produced by the trainable policy during RL, nor expose the same requirement–capability gaps that TReDS identifies online. In contrast, TReDS uses online trajectories and verifier-derived performance signals to update policy capability and allocate training probability mass dynamically. Direct RL with distribution shaping is therefore better aligned with the final optimization target than a static supervised warm-start stage.

This observation is consistent with recent RL post-training results suggesting that supervised cold-start data can improve readability or formatting, but is not always necessary for eliciting useful reasoning or tool-use behavior. For example, DeepSeek-R1-Zero shows that large-scale RL without an initial SFT stage can elicit strong reasoning behavior, while the later DeepSeek-R1 pipeline uses cold-start data mainly to improve readability and reduce language-mixing before further RL. Similarly, recent agent RL studies report that supervised fine-tuning alone may struggle with robustness and generalization in dynamic interactive settings, whereas verifier- or rule-based RL can better optimize action correctness under interactive evaluation. These findings support our empirical choice: because warm-start SFT does not improve the official τ^2-Bench objective in our setting, we retain the RL-only TReDS mainline and treat SFT as an auxiliary negative result.

G Analysis: Centered Region Prototypes

G.1 Centered Region Prototypes

To further inspect the multidimensional structure of the initialized requirement regions, we additionally visualize centered region-level prototypes. For each region b, we first compute its region-level requirement prototype by averaging the sample-level offline requirement vectors:

$$c_b^{\text{off}} = \frac{1}{|\mathcal{X}_b|} \sum_{x \in \mathcal{X}_b} z_{\text{off}}(x). \tag{15}$$

We then subtract the global average requirement profile

$$\bar{z}_{\text{off}} = \frac{1}{|\mathcal{X}_{\text{train}}|} \sum_{x \in \mathcal{X}_{\text{train}}} z_{\text{off}}(x) \tag{16}$$

from each region prototype, obtaining the centered prototype

$$\Delta c_b = c_b^{\text{off}} - \bar{z}_{\text{off}}. \tag{17}$$

Figure 6 shows the resulting centered prototypes. Compared with the raw prototype visualization in the main text, this centered view removes the global average requirement trend and highlights which dimensions distinguish each region from the overall training set. The figure shows that different regions deviate from the global average along different requirement dimensions, confirming that the initialized regions are not merely ordered by scalar difficulty.

This analysis further supports the role of $s_{\text{off}}(x)$ in TReDS. The scalar projection is used only as a compact interface for initializing coarse requirement regions, while the information relevant to scheduling is retained in the multidimensional prototypes c_b^{off}. Therefore, even though regions are initialized through a scalar score, the scheduler-facing structure remains multidimensional.

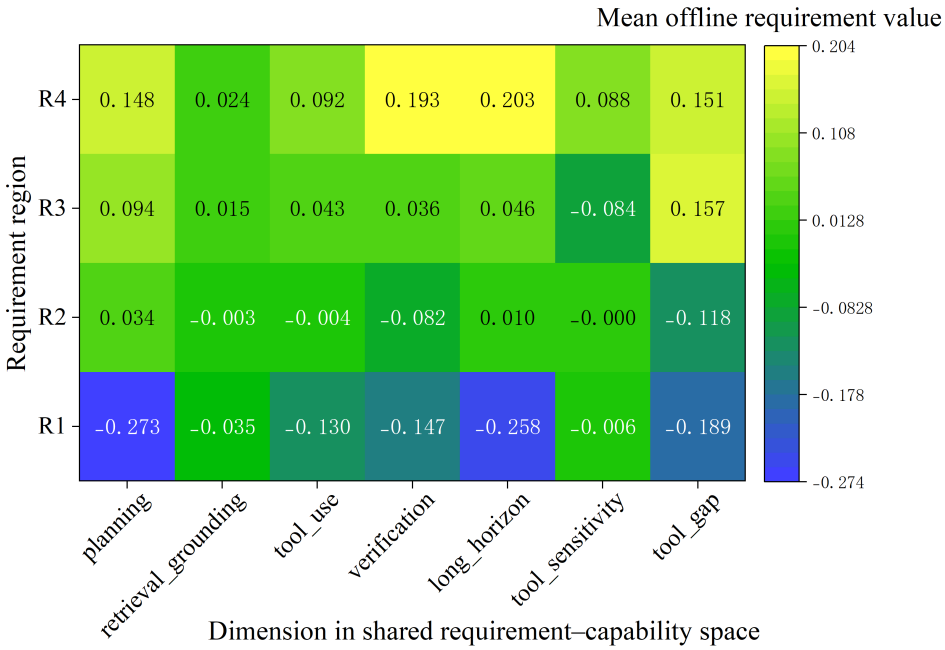

Figure 6: Analysis of Centered Region Prototypes.

G.2 Policy-Conditioned Scalar Calibration

As Table 6 shows, we analyze the final online-cache snapshot to examine whether online evidence changes the scheduler-facing scalar view used by the controller. For each active record, we compare the offline scalar score $s_{\text{off}}(x)$ with the online-corrected score $s_{\text{cond},t}(x)$, and keep only records with non-empty sample sets and $\alpha_{x,t} > 0$.

Across all filtered active records, the mean correction is

$$\Delta s_t = s_{\text{cond},t} - s_{\text{off}}, \qquad \mathbb{E}[\Delta s_t] = -0.0695. \tag{18}$$

This corresponds to about 15.3% of the mean offline score (0.4541), indicating a moderate but non-trivial calibration rather than a negligible perturbation. The correction is also highly directional: 96.55% of records have $\Delta s_t < 0$, and 72.41% have $|\Delta s_t| > 0.05$. Thus, after observing online policy behavior, the controller-facing scalar view is systematically adjusted rather than copied from the static offline estimate.

Importantly, the correction does not destroy the offline structure. The Spearman correlation between s_{off} and $s_{\text{cond},t}$ remains high (0.9379), suggesting that online correction preserves the coarse offline ordering while calibrating scores within that structure. Region-level statistics further show that corrections appear across active regions, with mean Δs_t values of -0.0210, -0.0748, -0.0829, and -0.1235 for R1–R4, respectively.

This diagnostic should not be interpreted as full evidence of per-step seven-dimensional requirement drift. It is a scalar-level final-cache analysis. Nevertheless, it supports the intended role of

online correction: TReDS does not simply reuse a fixed offline difficulty estimate, but calibrates the scheduler-facing view according to observed policy behavior.

Statistic	Value
Mean s_{off}	0.4541
Mean $s_{\text{cond},t}$	0.3846
Mean Δs_t	-0.0695
Std. Δs_t	0.0371
Median Δs_t	-0.0781
Fraction $\Delta s_t < 0$	96.55%
Fraction $\lvert\Delta s_t\rvert > 0.05$	72.41%
Mean $\alpha_{x,t}$	0.2582
Spearman corr. $(s_{\text{off}}, s_{\text{cond},t})$	0.9379

Table 6: Policy-Conditioned scalar calibration diagnostics.

H Limitations

TReDS has several limitations. First, our experiments focus on verifiable tool-interactive agent tasks where trajectories can be evaluated by executable verifiers, task-specific judges, or benchmark protocols. This setting is important for controlled RL post-training, but it does not cover open-ended agent tasks where success is ambiguous, feedback is delayed, or reliable verification is unavailable. Extending requirement–capability distribution shaping to such settings would require more robust reward modeling and uncertainty estimation.

Second, TReDS depends on multi-view offline probing and online trajectory diagnostics. Although this provides richer evidence than scalar rewards, it introduces additional computational overhead and assumes that meaningful tool views can be constructed for the training tasks. In domains where tool access cannot be systematically varied, or where probing trajectories are expensive to collect, the offline requirement profiling stage may become less practical.

Third, although the five tool clusters and seven requirement dimensions were induced from the training data rather than fully prespecified, their construction still involves model-assisted induction and manual adjudication. The resulting taxonomies remain fixed throughout training and evaluation. We did not retrain models using alternative values of K or alternative requirement-space taxonomies, and therefore do not claim that the current choices are optimal or insensitive to these design decisions. Future work may investigate alternative taxonomies, more automated induction, and their transfer across agent domains.

Fourth, our empirical claims are bounded by the evaluated models, benchmarks, and training budget. We evaluate TReDS on τ^2-Bench, τ-Bench native-bridge evaluation, BFCL V3, and API Bank, using limited interaction budgets and a small number of random seeds. These results support the effectiveness of distribution shaping under the tested protocol shifts, but they do not establish universal generalization to all tool-use benchmarks, larger-scale agents, or substantially different tool ecosystems.

Finally, TReDS changes the training distribution rather than the underlying policy optimizer. This design isolates the effect of requirement-conditioned allocation, but it also means that performance remains dependent on the quality of the verifier, the stability of rollout scores, and the expressiveness of the base policy. When verifier signals are noisy or the base model lacks the prerequisite capabilities, scheduler-shaped sampling alone may not be sufficient.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state that TReDS performs trajectory-grounded requirement–capability modeling and uses the resulting controller state to shape the GRPO training distribution. The claims are empirical and are evaluated on τ^2-Bench, τ-Bench native-bridge evaluation, BFCL V3, and API Bank, without claiming a new GRPO objective or benchmark-specific fine-tuning.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We explicitly discuss the limitations of our paper in Appendix H.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present formal theoretical results, theorems, or proofs. The mathematical expressions define the TReDS controller, requirement estimation, online correction, scheduler, and GRPO interface.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Sections 3 and 4 and Appendices A–E describe the method, training data, model configurations, GRPO optimization settings, evaluation protocols, and baseline implementations. Appendix A provides the main experimental configuration and the capability estimator and controller settings, while Appendix C details the offline probing budget.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We have provided the relevant implementation code.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: In Section 4 and the Appendix of the paper, we have already provided the experimental details and settings.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report the average results and the standard deviation of the experiment results in Section 4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The computing resources for the experiments have been provided in Appendix Section A.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This work conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

Justification: Our work does not involve any social impact. There is no necessary societal impact of our work that should be specified.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not introduce a new high-risk pre-trained model, image generator, or scraped dataset release. If checkpoints are released, they should inherit the usage restrictions and safety considerations of the underlying base model and benchmark assets.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper properly cites the original sources for datasets, models, and frameworks used, including Qwen3, τ^2-Bench, τ-Bench, BFCL V3, and API-Bank.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.

- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: This paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs are central to the research: Qwen3 models are used as trainable policies, a stronger LLM is used for offline probing, and model-based user simulation is used in the τ^2-Bench official evaluation. The method and appendix distinguish training, probing, and evaluation-time usage.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.