TOOLCOMPASS: GUIDING TOOL TRIALING, NOT SUPPRESSING IT

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents must generalize from tools seen during training to unseen tools at deployment. A key challenge is *tool trialing*, i.e., excessive trials waste the interaction budget, whereas selective trials enable exploration of unfamiliar tools. Existing outcome-based post-training leaves wasteful trials unguided, while turn-level supervision may suppress necessary exploration. We introduce TOOLCOMPASS, a post-training framework that guides tool trialing by organizing tool-call representations according to shared functions. Specifically, TOOLCOMPASS models each function class as a von Mises–Fisher distribution and jointly reduces intra-function variation across domains and increases inter-function separation. This structure transfers experience from seen tools to functionally similar unseen tools, directing exploration away from unrelated alternatives. TOOLCOMPASS requires no ground-truth call traces or unseen-tool access and incurs no inference overhead. Experiments on AppWorld and FTRL show consistent gains across GRPO, RFT, and DMPO. TOOLCOMPASS improves AppWorld OOD task success by up to 10.71 percentage points over vanilla post-training and performs best among competitive baselines on both benchmarks.

1 INTRODUCTION

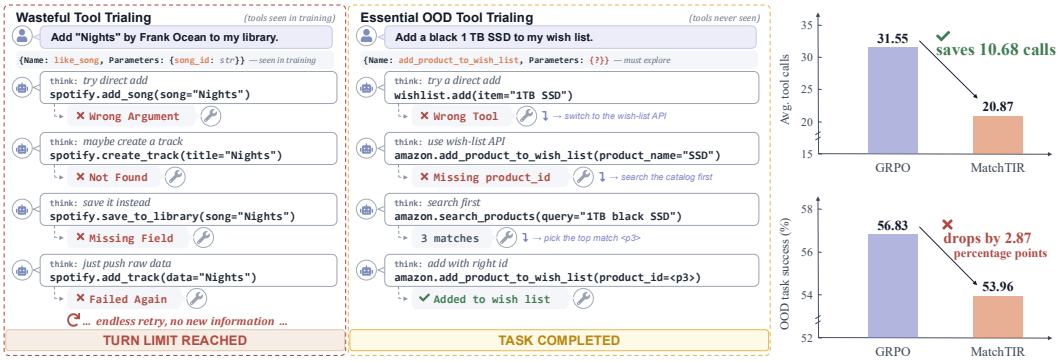
Large language model (LLM) agents interact with external environments through tools such as search engines and application APIs (Yao et al., 2023; Schick et al., 2023; Patil et al., 2024; Qin et al., 2024; Qu et al., 2025). A reliable agent should not only use tools seen during training, but also transfer learned functions to unseen tools in out-of-distribution (OOD) environments (Tang et al., 2023; Mekala et al., 2024; He et al., 2025). For example, experience with `spotify.search_songs` should help the agent invoke `amazon.search_products`. In multi-turn environments, an agent may try a plausible call, observe an error, and revise its tool choice or arguments. We refer to this trial-and-correction behavior as *tool trialing* (Gao et al., 2026). Unnecessary trials consume the interaction budget and can cause solvable tasks to fail (Liu et al., 2025; Mansoor et al., 2026). Yet trialing might be essential for unfamiliar tools, whose behavior must be learned through environment feedback. The agent must therefore reduce wasteful calls without losing the exploration required for OOD transfer (Figure 1a).

Existing post-training methods do not effectively resolve this trade-off. Outcome-based methods for tool use (Qian et al., 2025; Jin et al., 2025; Feng et al., 2026), such as GRPO (Shao et al., 2024), assign the same trajectory-level advantage across tool turns, learning wasteful trials together with useful solution steps. Recent methods instead introduce turn-level supervision (Yu et al., 2025; Zeng et al., 2025a), either by matching predicted calls to a ground-truth trace (MatchTIR (Qu et al., 2026)) or estimating each turn's contribution to the final outcome (TRACE (Tao et al., 2026); StepTool (Yu et al., 2025)). Despite the promise, such supervision can discourage deviations from known trajectories, including exploration that becomes necessary for unseen tools. Our pilot study confirms this limitation on held-out AppWorld applications (Trivedi et al., 2024): MatchTIR substantially reduces tool calls and rarely reaches the turn limit, yet achieves lower task success than GRPO (Figure 1b). This motivates the central question of our work:

How can post-training enable OOD exploration without reinforcing wasteful tool trialing?

To understand this failure, we examine the rollouts and tool-call representations of a Qwen3.5-9B (Team, 2026) agent post-trained with GRPO on AppWorld. When an unseen tool shares a func-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Tool trialing: wasteful on seen tools, essential for OOD exploration **(b) Suppressing trials lowers OOD success**

Figure 1: **Tool trialing should be guided rather than suppressed.** (a) Excessive trialing wastes the interaction budget to find the needed tool "like_song", whereas selective trialing enables exploration of unfamiliar tools of "add_product_to_wish_list". (b) On held-out AppWorld applications with Qwen3.5-4B, MatchTIR makes fewer calls than GRPO but achieves lower task success, revealing the cost of suppressed exploration.

tion with a seen tool, the agent does not exploit this similarity to narrow its exploration. Instead, it trials unrelated tools before reaching a useful call (related examples in Appendix B.5). The representation space exhibits a corresponding pattern that *calls from the same domain lie close together despite performing different functions, whereas calls sharing a function remain far apart across domains* (Figure 2a). The learned representations therefore encode domain identity rather than the function-level commonality needed to transfer experience across tools.

Motivated by this finding, we propose TOOLCOMPASS, a vMF-based post-training framework that shapes tool-call representations by shared function. Our key idea is to promote low intra-function variation across domains and high inter-function separation. Specifically, TOOLCOMPASS maps each tool-call representation onto the unit hypersphere. Each function class can then be naturally modeled as a von Mises–Fisher (vMF) distribution (Mardia & Jupp, 2009) centered at a prototype (Figure 3). A variation loss draws calls toward their function prototype, while a separation loss keeps different prototypes apart. The resulting objective is optimized jointly with the original post-training objective. By organizing the policy's hidden representations around tool functions, TOOLCOMPASS makes experience with seen tools more transferable to functionally similar unseen tools, directing exploration away from unrelated alternatives (Figure 2b).

Importantly, TOOLCOMPASS requires only the function-class assignments of seen tools, without ground-truth call traces, access to unseen tools, or frozen turn-level models during training. It can be readily combined with GRPO (Shao et al., 2024), RFT (Yuan et al., 2023), and DMPO (Shi et al., 2024). The projection head and prototypes are removed after post-training, introducing no additional inference overhead. TOOLCOMPASS therefore guides *where* the agent trials rather than directly penalizing *how much* it trials. Extensive experiments on AppWorld and FTRL (Ye et al., 2026), using Qwen3.5-4B and Qwen3.5-9B, show that TOOLCOMPASS consistently improves task success across all three post-training methods, with strong performance on OOD tasks requiring unseen tools. On AppWorld, TOOLCOMPASS improves OOD task success over vanilla post-training by up to 10.71 percentage points, while reaching 72.36% overall task success with Qwen3.5-9B under GRPO. Further analyses show that TOOLCOMPASS produces shorter trajectories, reduces trials of unrelated tools, increases the adoption of task-relevant unseen tools, and organizes representations by shared function rather than domain (Section 4.3). Our key contributions are summarized as follows:

- We identify a tool-trialing trade-off in OOD tool use, i.e., outcome rewards leave wasteful trials unguided, whereas strict turn-level supervision can suppress necessary exploration. We further connect this failure to representations organized by domain rather than function.
- We propose TOOLCOMPASS, a vMF-based framework that reduces intra-function variation and increases inter-function separation during post-training, guiding exploration without ground-truth call traces, unseen-tool access, or additional inference computation.
- Extensive experiments on AppWorld and FTRL demonstrate consistent improvements across models and post-training objectives, including on OOD tasks, alongside more efficient trialing and stronger adoption of unseen tools.

2 PROBLEM SETUP

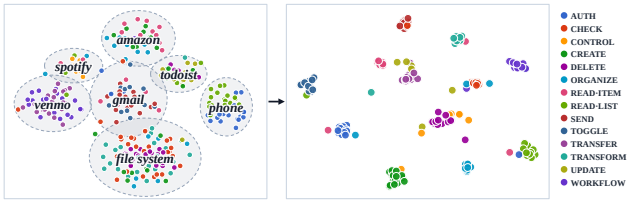
Formally, we describe the tool-use agent, post-training and deployment settings, and learning goal.

Post-training for tool-use agents. A tool-use agent is a policy π_θ that interacts with an environment over a trajectory $\tau = (s_1, a_1, o_1, \ldots, s_T, a_T, o_T)$. At turn t, the state s_t contains the user request, interaction history, and specifications of the available tools; the action a_t is either a language response or a call to tool $u_t \in \mathcal{U}$; and the observation o_t is the resulting environment feedback, such as an execution result or error message.

Tools from different domains may implement the same function. For example, `amazon.search_products` and `spotify.search_songs` both perform SEARCH. We group such tools into *function classes* and denote the function class of tool u by $c(u) \in \mathcal{C}$. The agent is post-trained on tasks involving a fixed set of seen tools $\mathcal{U}_{\text{train}}$ using an objective $\mathcal{L}_{\text{post}}(\theta)$.

OOD tool generalization. At deployment, the agent encounters in-distribution (ID) tasks solvable with tools in $\mathcal{U}_{\text{train}}$, as well as OOD tasks requiring tools from an unseen set $\mathcal{U}_{\text{unseen}}$, where $\mathcal{U}_{\text{train}} \cap \mathcal{U}_{\text{unseen}} = \varnothing$. These unseen tools belong to held-out domains but may share similar functions with the seen tools. Their specifications are provided to the agent only at deployment through s_t; *neither the tools nor their annotations are available* during post-training. Our goal is to transfer function-level knowledge from $\mathcal{U}_{\text{train}}$ to $\mathcal{U}_{\text{unseen}}$ while preserving performance on ID tasks.

Challenge and learning goal. A tool-use trajectory may contain useful trials of functionally plausible tools as well as unproductive trials of unrelated tools. The latter consume the interaction budget and can cause a solvable task to fail (Liu et al., 2025; Mansoor et al., 2026), whereas the former provide feedback needed to use unfamiliar tools. This challenge is reflected in the representation space learned by standard post-training: as shown in Figure 2(a), GRPO organizes tool calls primarily by domain (e.g., different applications) rather than by shared function, providing little structure for transferring experience to unseen tools. Penalties that indiscriminately discourage trialing may instead suppress both useful and unproductive calls. We therefore aim to learn function-level representations, as shown in Figure 2(b), that guide trialing toward functionally relevant tools without suppressing the exploration required for OOD generalization.



(a) GRPO (b) ToolCompass

Figure 2: **Tool-call representations on AppWorld.** After GRPO post-training (a), representations cluster primarily by application domain (dashed ellipses) rather than function class (colors). In contrast, TOOLCOMPASS (b) aligns tool calls sharing the same function across domains.

3 METHOD

Framework overview. Our framework TOOLCOMPASS guides tool trialing by shaping the representations of tool calls according to their functions. Central to our framework is to reduce variation among calls implementing the same function across domains, while separating calls implementing different functions. As illustrated in Figure 3, TOOLCOMPASS first extracts and normalizes tool-call representations (Section 3.1). It then models each function class with a von Mises–Fisher (vMF) distribution and optimizes intra-function variation and inter-function separation (Section 3.2). The resulting objective is trained jointly with the original post-training objective (Section 3.3).

3.1 TOOL-CALL REPRESENTATION

Representation extraction. For each tool call a_t in a sampled trajectory, we extract the hidden representation from the agent (with parameter θ):

$$\mathbf{h}_t = H_\theta^{(l)}(s_t, a_t) \in \mathbb{R}^{d_h}, \tag{1}$$

where $H_\theta^{(l)}$ returns the hidden state at layer l and the final token of a_t, and d_h is the hidden dimension. Since the final token attends to the preceding context, $\mathbf{h}_t$ summarizes the selected tool, its arguments, and the current interaction state. We study the choice of layer l in Section 4.

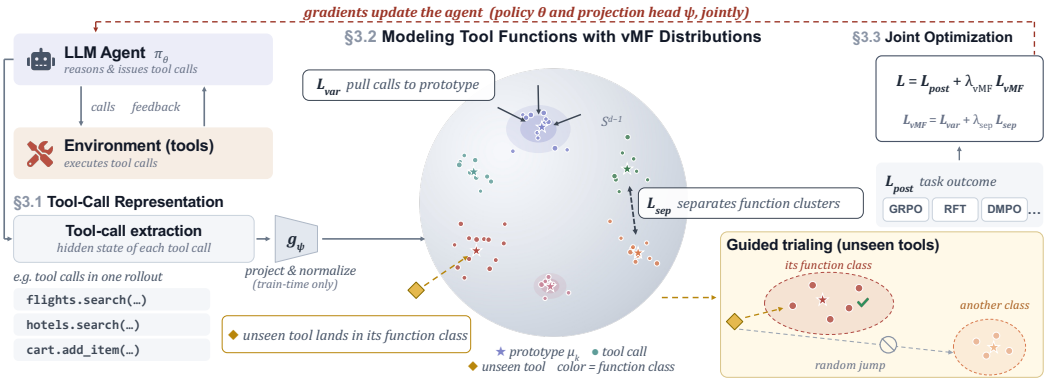


Figure 3: **Overview of TOOLCOMPASS.** For each tool call, TOOLCOMPASS extracts its final-token representation and maps it onto the unit hypersphere. Calls implementing the same function are aligned with a shared vMF prototype, while calls implementing different functions are separated. The resulting objective is optimized jointly with the post-training objective. The projection head and prototypes are not used at deployment.

Hyperspherical normalization. A lightweight projection head $g_\psi : \mathbb{R}^{d_h} \to \mathbb{R}^d$ maps $\mathbf{h}_t$ to $\mathbf{z}_t = \frac{g_\psi(\mathbf{h}_t)}{\|g_\psi(\mathbf{h}_t)\|_2} \in \mathbb{S}^{d-1}$. The unit norm places all tool-call representations on the unit hypersphere, where angular similarity characterizes their relationships and the vMF distribution is naturally defined (Mardia & Jupp, 2009).

3.2 Modeling Tool Functions with vMF Distributions

Our core idea is to model each function class as a compact directional distribution, allowing calls from different domains to share a common direction whenever they implement the same function.

vMF model. For each function class $k \in \mathcal{C}$, we model its normalized representations with a vMF distribution, which is analogous to spherical Gaussian distributions for embeddings with unit norms:

$$p(\mathbf{z} \mid c = k) = C_d(\beta) \exp\left(\beta \boldsymbol{\mu}_k^\top \mathbf{z}\right), \tag{2}$$

where $\boldsymbol{\mu}_k \in \mathbb{S}^{d-1}$ is the prototype direction, $\beta \geq 0$ is the concentration, and $C_d(\beta)$ is the normalizing constant. A larger β concentrates the distribution more tightly around $\boldsymbol{\mu}_k$, whereas $\beta = 0$ yields a uniform distribution on the hypersphere.

Under this probability model, the posterior probability of function class k is

$$p(c = k \mid \mathbf{z}) = \frac{C_d(\beta) \exp\left(\beta \boldsymbol{\mu}_k^\top \mathbf{z}\right)}{\sum_{j \in \mathcal{C}} C_d(\beta) \exp\left(\beta \boldsymbol{\mu}_j^\top \mathbf{z}\right)} = \frac{\exp\left(\boldsymbol{\mu}_k^\top \mathbf{z}/\tau\right)}{\sum_{j \in \mathcal{C}} \exp\left(\boldsymbol{\mu}_j^\top \mathbf{z}/\tau\right)}, \qquad \beta = \frac{1}{\tau},$$

where $\tau > 0$ is the temperature. Since $\boldsymbol{\mu}_k$ and $\mathbf{z}$ have unit norm, their inner product is the cosine similarity. A smaller τ, equivalently a larger β, produces a more concentrated function distribution.

Reducing intra-function variation. Consider a minibatch of tool calls $\mathcal{B} = \{(\mathbf{z}_i, c_i)\}_{i=1}^N$, where $c_i = c(u_i)$ is the function class of the called training tool. We minimize the negative log-likelihood of the observed function assignments:

$$\mathcal{L}_{\text{var}} = -\frac{1}{N} \sum_{i=1}^{N} \log p(c = c_i \mid \mathbf{z}_i). \tag{3}$$

Minimizing $\mathcal{L}_{\text{var}}$ aligns each tool call with its function prototype, thereby reducing intra-function variation across tools, domains, and interaction contexts.

Increasing inter-function separation. The softmax denominator in $\mathcal{L}_{\text{var}}$ already contrasts calls with competing prototypes; $\mathcal{L}_{\text{sep}}$ adds class-level separation. For each function class represented in the minibatch, we compute its normalized batch direction $\bar{\mathbf{z}}_k = \sum_{i \in \mathcal{I}_k} \mathbf{z}_i / \left\|\sum_{i \in \mathcal{I}_k} \mathbf{z}_i\right\|_2, \mathcal{I}_k = \{\, i : c_i = k \,\}, k \in \mathcal{C}_\mathcal{B}$, where $\mathcal{C}_\mathcal{B} = \{c_i : (\mathbf{z}_i, c_i) \in \mathcal{B}\}$ denotes the function classes appearing in the minibatch. We then define

$$\mathcal{L}_{\text{sep}} = \frac{1}{|\mathcal{C}_\mathcal{B}|} \sum_{k \in \mathcal{C}_\mathcal{B}} \log \left[\frac{1}{|\mathcal{C}| - 1} \sum_{j \in \mathcal{C}, j \neq k} \exp\left(\frac{\bar{\mathbf{z}}_k^\top \boldsymbol{\mu}_j}{\tau}\right) \right]. \tag{4}$$

Minimizing $\mathcal{L}_{\text{sep}}$ pushes each batch direction away from the prototypes of other function classes, increasing their angular separation. The prototypes are treated as stop-gradient targets when computing both losses, so $\mathcal{L}_{\text{var}}$ and $\mathcal{L}_{\text{sep}}$ update the policy and projection head through the current representations.

Prototype estimation. We maintain each function prototype using an exponential moving average. After processing a minibatch, the prototype of every observed class is updated as

$$\boldsymbol{\mu}_k \leftarrow \text{Normalize}\left(\alpha\boldsymbol{\mu}_k + (1-\alpha)\bar{\mathbf{z}}_k\right), \qquad k \in \mathcal{C}_{\mathcal{B}}, \tag{5}$$

where $\alpha \in [0, 1)$ is the update factor. The prototypes track the evolving tool-call representations.

Representation-shaping objective. The complete objective is

$$\mathcal{L}_{\text{vMF}} = \mathcal{L}_{\text{var}} + \lambda_{\text{sep}}\mathcal{L}_{\text{sep}}, \tag{6}$$

where λ_{sep} is the weight coefficient modulating the relative importance of the two losses. The two losses organize tool-call representations into compact and separated function clusters. This structure makes knowledge learned from a seen tool transferable to unseen tools implementing the same function, directing trialing toward functionally plausible tools rather than unrelated alternatives.

Comparison with existing methods. Outcome-based methods such as GRPO apply the same trajectory-level advantage to the tool calls within a rollout, indicating whether the trajectory succeeds but not what function each call performs. In contrast, $\mathcal{L}_{\text{vMF}}$ provides function-level supervision: calls implementing the same function share a directional target even when they involve different tools or domains. Unlike trace-matching supervision (Qu et al., 2026), TOOLCOMPASS does not prescribe a gold sequence of calls; unlike turn-contribution methods (Tao et al., 2026; Xie et al., 2026), it does not require estimating the contribution of individual turns. It therefore complements outcome supervision by guiding where the agent trials without directly penalizing exploration. Figure 4 visualizes the normalized representation space, with the spherical surface showing the kernel density of a representative function class; compared with GRPO's diffuse distribution, TOOLCOMPASS forms a compact cluster across tools (see Appendix B.9).

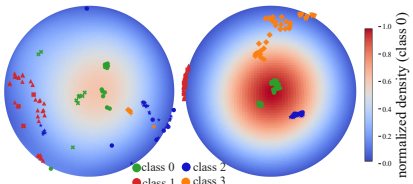


Figure 4: The surface shows the spherical kernel density of one function class (class 0) under GRPO (a) and TOOLCOMPASS (b), on a shared color scale. We showcase four classes for visual clarity; colors denote function classes and markers denote tools.

3.3 JOINT OPTIMIZATION WITH POST-TRAINING

We optimize the representation-shaping objective jointly with the original post-training objective:

$$\mathcal{L}(\theta, \psi) = \mathcal{L}_{\text{post}}(\theta) + \lambda_{\text{vMF}}\mathcal{L}_{\text{vMF}}(\theta, \psi). \tag{7}$$

Here, $\mathcal{L}_{\text{post}}$ uses task feedback to learn which tool-use behaviors help complete the task. $\mathcal{L}_{\text{vMF}}$ aligns call representations across tools that perform the same function. Joint training encourages the policy to transfer useful tool-use experience from seen tools to unseen tools with matching functions. The host objective can be instantiated with GRPO (Shao et al., 2024), RFT (Yuan et al., 2023), or DMPO (Shi et al., 2024) without changing its original formulation. The overall procedure is provided in Algorithm 1 in Appendix A.

Our training components consist only of the projection head g_ψ and one prototype per function class. Both are not utilized at deployment. Since $\mathcal{L}_{\text{vMF}}$ has already shaped the policy parameters θ, the agent uses its original policy without an auxiliary scoring module or additional inference pass.

4 EXPERIMENTS

4.1 SETUP

Benchmarks, models, and evaluation. We evaluate TOOLCOMPASS on AppWorld (Trivedi et al., 2024) and FTRL (Ye et al., 2026), two multi-turn tool-use benchmarks with executable tools and verifiable task outcomes. We additionally evaluate semantic OOD transfer to three jointly held-out

Table 1: **TOOLCOMPASS across post-training objectives.** Results are task success rate (%) on AppWorld and Solve-F1 (%) on FTRL. Each objective is compared with its TOOLCOMPASS-augmented counterpart under the same backbone and training setup.

Method	Variant	AppWorld			FTRL		
		ID	OOD	Total	ID	OOD	Total
		Qwen3.5-4B					
GRPO (Shao et al., 2024)	Original	58.33	56.83	57.26	34.82	60.40	38.91
	+TOOLCOMPASS	**70.63**	**64.75**	**66.44**	**46.97**	**64.01**	**49.70**
RFT (Yuan et al., 2023)	Original	44.64	35.01	37.78	34.93	52.17	37.69
	+TOOLCOMPASS	**46.43**	**39.09**	**41.20**	**40.98**	**55.78**	**43.35**
DMPO (Shi et al., 2024)	Original	39.29	30.94	33.33	36.79	45.68	38.21
	+TOOLCOMPASS	**44.44**	**33.97**	**36.98**	**42.37**	**52.92**	**44.06**
		Qwen3.5-9B					
GRPO (Shao et al., 2024)	Original	70.83	61.15	63.93	38.70	61.20	42.30
	+TOOLCOMPASS	**78.17**	**70.02**	**72.36**	**48.99**	**64.98**	**51.55**
RFT (Yuan et al., 2023)	Original	55.36	40.05	44.44	35.94	46.70	37.66
	+TOOLCOMPASS	**60.12**	**50.76**	**53.45**	**40.70**	**57.39**	**43.37**
DMPO (Shi et al., 2024)	Original	60.71	49.16	52.48	37.63	53.93	40.24
	+TOOLCOMPASS	**65.67**	**55.32**	**58.29**	**46.90**	**55.73**	**48.31**

function classes on FTRL (Appendix B.1). In the main experiments, we partition the available tools into a seen set $\mathcal{U}_{\text{train}}$ and a held-out set $\mathcal{U}_{\text{unseen}}$. ID tasks can be completed using tools from $\mathcal{U}_{\text{train}}$, whereas OOD tasks require at least one tool from $\mathcal{U}_{\text{unseen}}$. The unseen tools and their function annotations are inaccessible during post-training; their specifications are provided to the agent only at evaluation time. The AppWorld test set contains 168 ID and 417 OOD tasks, and the FTRL test set contains 168 ID and 32 OOD tasks. We use Qwen3.5-4B and Qwen3.5-9B (Team, 2026) as policy backbones. Throughout all experiments, we adopt the ReAct interaction scaffold (Yao et al., 2023), in which the agent interleaves language reasoning with executable tool calls and revises its subsequent decisions based on environment observations, including execution results and error messages. Following AppWorld (Trivedi et al., 2024), we use its state-based evaluator and report Task Success Rate, defined as the percentage of tasks for which all task-specific evaluation tests are passed. Following MatchTIR (Qu et al., 2026), we report Solve-F1 on FTRL, which balances tool-invocation precision and task-completion recall. For both benchmarks, we report performance on the ID subset, the OOD subset, and the entire test set. Detailed benchmark versions, post-training data, ID/OOD split construction, function-class annotations, prompts, interaction protocols, metric implementations, training and inference configurations, and computational resources are provided in Appendix A.

Baselines. We compare TOOLCOMPASS against six categories of baselines. First, prompting-based agents include GPT-5.5 (OpenAI, 2026), Claude Opus 4.8 (Anthropic, 2026), GLM-5.2 (Zeng et al., 2026), and DeepSeek V4 Pro (Xu et al., 2026). These agents use the same ReAct scaffold (Yao et al., 2023), tool specifications, interaction budget, and evaluation protocol described above. Second, general post-training objectives include GRPO (Shao et al., 2024), RFT (Yuan et al., 2023), and DMPO (Shi et al., 2024); we additionally augment each objective with TOOLCOMPASS to evaluate its compatibility with different host objectives. Unless otherwise stated, TOOLCOMPASS uses GRPO as the base objective in all remaining experiments. Third, turn-level tool-use post-training methods include StepTool (Yu et al., 2025), FTRL-M (Ye et al., 2026), MatchTIR (Qu et al., 2026), SOAR (Li et al., 2026b), and TRACE (Tao et al., 2026). Fourth, tool-use RL methods include SimpleTIR (Xue et al., 2026), ToolMaster (Gao et al., 2026), and LOOP (Chen et al., 2025). Fifth, we include SEAL (Li et al., 2026c) as a representation-learning baseline. Finally, the OOD-generalization comparison includes CORAL (Sun & Saenko, 2016), Group DRO (Sagawa et al., 2020), ToolRL (Qian et al., 2025), and PAFT (Lv et al., 2026). All trainable baselines use the same policy initialization, post-training data, tool specifications, and interaction budget whenever applicable. Baseline-specific adaptations and reproduction details are provided in Appendix A.6.

4.2 MAIN RESULTS

TOOLCOMPASS benefits across post-training objectives. Table 1 compares each post-training objective with its TOOLCOMPASS-augmented counterpart. TOOLCOMPASS improves every reported ID, OOD, and total score across both benchmarks and model scales. On Qwen3.5-4B, it improves the AppWorld and FTRL total scores by 9.18 and 10.79 percentage points with GRPO, 3.42 and 5.66

Table 2: **Main comparison on AppWorld and FTRL.** Results are task success rate (%) on AppWorld and Solve-F1 (%) on FTRL. All prompting models use the same ReAct scaffold (Yao et al., 2023). Rep. learning abbreviates representation learning, and FTRL-M is the multi-turn variant of the FTRL method. For brevity, we report mean ± standard deviation over the same three seeds for LOOP, SEAL, and TOOLCOMPASS.

		AppWorld			FTRL		
Method	Category	ID	OOD	Total	ID	OOD	Total
Prompting-based							
GPT-5.5 (OpenAI, 2026)	Prompting	70.24	63.07	65.13	33.06	54.32	36.46
Claude Opus 4.8 (Anthropic, 2026)	Prompting	70.83	58.51	62.05	44.18	49.64	45.05
GLM-5.2 (Zeng et al., 2026)	Prompting	58.33	54.92	55.90	40.64	43.85	41.15
DeepSeek V4 Pro (Xu et al., 2026)	Prompting	70.24	55.64	59.83	35.52	52.23	38.20
Qwen3.5-4B							
GRPO (Shao et al., 2024)	Base RL	58.33	56.83	57.26	34.82	60.40	38.91
DMPO (Shi et al., 2024)	Base RL	39.29	30.94	33.33	36.79	45.68	38.21
StepTool (Yu et al., 2025)	Turn-level RL	65.48	52.76	56.41	40.11	49.59	41.63
FTRL-M (Ye et al., 2026)	Turn-level RL	65.48	53.00	56.58	36.81	59.01	40.36
MatchTIR (Qu et al., 2026)	Turn-level RL	69.05	53.96	58.29	39.91	50.41	41.59
SOAR (Li et al., 2026b)	Turn-level RL	64.88	50.12	54.36	35.55	57.64	39.08
TRACE (Tao et al., 2026)	Turn-level RL	69.05	55.16	59.15	39.26	55.94	41.93
SimpleTIR (Xue et al., 2026)	Tool RL	63.69	55.64	57.95	42.94	55.45	44.95
ToolMaster (Gao et al., 2026)	Tool RL	60.12	57.55	58.29	42.85	48.26	43.71
LOOP (Chen et al., 2025)	Tool RL	$65.87_{\pm 0.34}$	$58.99_{\pm 0.86}$	$60.97_{\pm 0.65}$	$41.83_{\pm 1.03}$	$57.72_{\pm 2.14}$	$44.37_{\pm 1.07}$
SEAL (Li et al., 2026c)	Rep. learning	$67.26_{\pm 1.19}$	$59.23_{\pm 0.96}$	$61.54_{\pm 1.03}$	$38.02_{\pm 1.49}$	$57.94_{\pm 2.51}$	$41.21_{\pm 1.63}$
TOOLCOMPASS (ours)	**Rep. learning**	$\mathbf{70.63}_{\pm 0.91}$	$\mathbf{64.75}_{\pm 1.27}$	$\mathbf{66.44}_{\pm 1.09}$	$\mathbf{46.97}_{\pm 1.76}$	$\mathbf{64.01}_{\pm 2.39}$	$\mathbf{49.70}_{\pm 1.53}$
Qwen3.5-9B							
GRPO (Shao et al., 2024)	Base RL	70.83	61.15	63.93	38.70	61.20	42.30
DMPO (Shi et al., 2024)	Base RL	60.71	49.16	52.48	37.63	53.93	40.24
StepTool (Yu et al., 2025)	Turn-level RL	74.40	59.95	64.10	46.50	44.23	46.13
FTRL-M (Ye et al., 2026)	Turn-level RL	72.02	61.63	64.62	42.79	51.36	44.16
MatchTIR (Qu et al., 2026)	Turn-level RL	75.60	58.27	63.25	48.41	45.36	47.93
SOAR (Li et al., 2026b)	Turn-level RL	62.50	53.24	55.90	42.85	52.28	44.36
TRACE (Tao et al., 2026)	Turn-level RL	74.40	57.79	62.56	48.89	49.92	49.05
SimpleTIR (Xue et al., 2026)	Tool RL	72.02	65.71	67.52	46.04	58.27	47.99
ToolMaster (Gao et al., 2026)	Tool RL	71.43	61.87	64.62	44.67	54.96	46.31
LOOP (Chen et al., 2025)	Tool RL	$71.03_{\pm 0.69}$	$67.15_{\pm 0.96}$	$68.26_{\pm 0.71}$	$43.61_{\pm 1.27}$	$58.55_{\pm 1.41}$	$46.00_{\pm 0.99}$
SEAL (Li et al., 2026c)	Rep. learning	$76.79_{\pm 1.19}$	$63.47_{\pm 0.97}$	$67.29_{\pm 1.03}$	$46.92_{\pm 1.48}$	$58.53_{\pm 2.46}$	$48.78_{\pm 1.57}$
TOOLCOMPASS (ours)	**Rep. learning**	$\mathbf{78.17}_{\pm 1.37}$	$\mathbf{70.02}_{\pm 1.87}$	$\mathbf{72.36}_{\pm 1.55}$	$\mathbf{48.99}_{\pm 1.21}$	$\mathbf{64.98}_{\pm 2.18}$	$\mathbf{51.55}_{\pm 1.07}$

points with RFT, and 3.65 and 5.85 points with DMPO, respectively. The improvements remain consistent on Qwen3.5-9B, reaching up to 9.01 points on AppWorld and 9.25 points on FTRL. These results demonstrate that TOOLCOMPASS is consistently effective across different post-training objectives and model scales.

Comparison with competitive baselines. Table 2 compares TOOLCOMPASS with prompting-based agents and trainable tool-use methods. TOOLCOMPASS achieves the best OOD and total scores across both benchmarks and model scales. On AppWorld with Qwen3.5-9B, it reaches a total task success rate of 72.36%, compared with 68.26% for LOOP (Chen et al., 2025), the strongest baseline on this metric, a gain of 4.10 percentage points. We note that a few proprietary models remain competitive on the AppWorld ID split, likely because *their training corpora may already cover related applications and tool-use data*, yet TOOLCOMPASS surpasses all prompting-based agents on every OOD and total score.

Comparison with OOD-generalization methods. Table 3 compares TOOLCOMPASS with general and agent-oriented OOD-generalization methods under GRPO with Qwen3.5-4B. On AppWorld, TOOLCOMPASS achieves an OOD score of 64.75% and a total score of 66.44%, exceeding CORAL, the strongest baseline on both metrics, by 6.48 and 6.27 percentage points, respectively. On FTRL, TOOLCOMPASS reaches an OOD score of 64.01% and a total score of 49.70%, exceeding GRPO by 3.61 points on OOD and ToolRL by 7.12 points on total performance. TOOLCOMPASS thus generalizes to tasks requiring unseen tools more effectively than the compared baselines.

4.3 ANALYSIS

Generalization across datasets. We train GRPO and TOOLCOMPASS on a source dataset and directly evaluate them on a different target dataset. As shown in Table 4, TOOLCOMPASS outperforms vanilla GRPO in both transfer directions. For example, FTRL-trained TOOLCOMPASS achieves 60.51% task success on AppWorld, versus 53.03% for GRPO; training TOOLCOMPASS on AppWorld itself yields 66.44%. These results suggest that the benefits of TOOLCOMPASS extend to cross-dataset transfer.

Table 3: **Comparison with OOD generalization methods.** All methods use GRPO with Qwen3.5-4B. Results are task success rate (%) on AppWorld and Solve-F1 (%) on FTRL.

Method	Category	AppWorld ID	AppWorld OOD	AppWorld Total	FTRL ID	FTRL OOD	FTRL Total
GRPO (Shao et al., 2024)	Vanilla RL	58.33	56.83	57.26	34.82	60.40	38.91
CORAL (Sun & Saenko, 2016)	General OOD	64.88	58.27	60.17	33.73	58.98	37.77
Group DRO (Sagawa et al., 2020)	General OOD	67.86	54.68	58.46	35.55	58.98	39.30
ToolRL (Qian et al., 2025)	Agentic OOD	68.45	56.12	59.66	39.20	60.32	42.58
PAFT (Lv et al., 2026)	Agentic OOD	60.12	51.56	54.02	36.47	57.84	39.89
TOOLCOMPASS (ours)	**Agentic OOD**	**70.63**	**64.75**	**66.44**	**46.97**	**64.01**	**49.70**

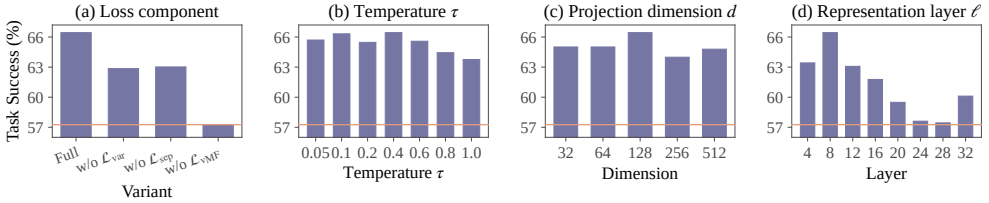


Figure 5: **Ablation studies.** (a) Effect of loss components, (b) effect of the vMF temperature τ, (c) effect of projection dimension d, and (d) effect of representation layer l. Results are AppWorld task success rates using Qwen3.5-4B with GRPO and are averaged over three seeds. The orange horizontal line denotes vanilla GRPO.

Effect of loss components. We ablate each loss component by removing it from the full objective. As shown in Figure 5 (a), removing any component degrades performance. Removing $\mathcal{L}_{\text{vMF}}$ causes the largest drop, from 66.44% to 57.26%, as the policy no longer receives function-level representation shaping. Removing $\mathcal{L}_{\text{var}}$ allows tool calls implementing the same function to spread apart, while removing $\mathcal{L}_{\text{sep}}$ reduces the separation between different function prototypes. Both losses contribute to the performance of the full objective.

Table 4: **Generalization across datasets.** Qwen3.5-4B results (%) on the full test sets: AppWorld task success and FTRL Solve-F1.

Test dataset	GRPO	TOOLCOMPASS
Training dataset: AppWorld		
AppWorld	57.26	**66.44**
FTRL	35.72	**42.31**
Training dataset: FTRL		
AppWorld	53.03	**60.51**
FTRL	38.91	**49.70**

Effect of prototype estimation. We compare EMA prototypes with learnable and batch-estimated prototypes. As shown in Figure 6(a), EMA prototype estimation achieves the best performance, suggesting that smoothly tracking the evolving representation space provides more stable function-level targets. We further ablate the EMA factor α in Appendix B.4.

Effect of the temperature τ. We vary the temperature τ of the vMF objective. Figure 5 (b) shows that performance peaks at $\tau = 0.4$ and remains competitive across a wide range, suggesting that TOOLCOMPASS is robust to the temperature setting.

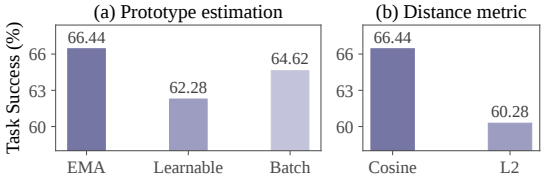


Figure 6: **Analysis of design choices.** (a) Effect of prototype estimation and (b) effect of distance metric. Results are AppWorld task success rates using Qwen3.5-4B with GRPO and are averaged over three seeds.

Effect of the output dimension d. We vary the output dimension of the projection head. As shown in Figure 5 (c), performance peaks at $d = 128$ and remains competitive across a wide range, showing that TOOLCOMPASS is not sensitive to a specific projection dimension.

How do different layers impact TOOLCOMPASS? In Figure 5 (d), we extract tool-call representations from different layers. Performance peaks at layer $l = 8$, suggesting that early-to-intermediate layers preserve the most transferable function-level information. TOOLCOMPASS remains above vanilla GRPO across all tested layers, indicating that it does not rely on a specific layer depth to be effective. Performance at the top layers degrades, likely because their representations become specialized for next-token prediction and drift away from the function-level semantics of the call.

Effect of distance metric. We compare cosine similarity with negative unsquared L2 scores at a fixed temperature (Appendix A.3). Figure 6(b) shows that cosine achieves 66.44% task success, compared with 60.28% for L2 under the same configuration.

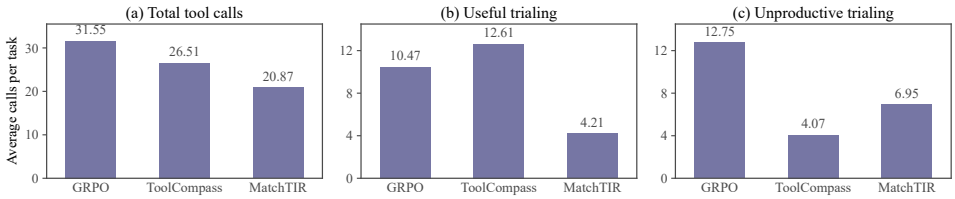


Figure 7: **AppWorld OOD tool-call analysis with Qwen3.5-4B.** (a) Total calls, (b) useful trialing, and (c) unproductive trialing. TOOLCOMPASS has more useful and fewer unproductive trials than GRPO.

Does TOOLCOMPASS guide rather than suppress tool trialing? We categorize tool calls following Appendix B.8. Figure 7 shows that TOOLCOMPASS makes fewer calls than GRPO (26.51 vs. 31.55) but more than MatchTIR (20.87). It retains more useful trials than GRPO and MatchTIR (12.61 vs. 10.47 and 4.21) and fewer unproductive trials than GRPO (4.07 vs. 12.75). These results are consistent with guided tool trialing: TOOLCOMPASS retains more useful trials while reducing unproductive ones.

Additional analysis. We provide further analyses in the Appendix: (1) semantic OOD analysis on FTRL with three jointly held-out function classes (Appendix B.1); (2) comparisons with alternative training objectives (Appendix B.2); (3) ablations on loss weights (Appendix B.3) and the EMA factor α in Equation 5 (Appendix B.4); (4) qualitative case studies on AppWorld OOD tasks (Appendix B.5); and (5) AppWorld metrics on Scenario Goal Completion (Appendix B.6).

5 RELATED WORK

OOD generalization. Conventional OOD generalization improves robustness through domain alignment, robust optimization, or representation learning (Sun & Saenko, 2016; Sagawa et al., 2020). These methods study fixed prediction, whereas tool-use agents face shifts in tasks, tools, observations, and interactions. Recent studies expose weak transfer across agent environments (Xi et al., 2026; Lv et al., 2026), while tool-specific work uses synthetic environments (Tang et al., 2023; He et al., 2025; Sullivan et al., 2025; Du et al., 2026), self-verification or interface optimization (Mekala et al., 2024; Guo et al., 2026), open-world retrieval (Wu et al., 2025; Molfetta et al., 2025; Zhuang et al., 2026; Huang et al., 2026), generalization-aware rewards (Zeng et al., 2025b; Qian et al., 2025), and documentation adaptation for evolving tools (Wu et al., 2026). TOOLCOMPASS instead organizes tool-call representations by shared function during post-training, enabling transfer to unseen tools without accessing them or adding test-time adaptation.

Tool-use agents. LLM agents use external tools through prompting, pretraining, and instruction tuning (Yao et al., 2023; Schick et al., 2023; Shen et al., 2023; Liang et al., 2024; Li et al., 2023; Patil et al., 2024; Qin et al., 2024). Post-training methods optimize trajectory outcomes (Jin et al., 2025; Qian et al., 2025; Feng et al., 2026; Chen et al., 2025; Xue et al., 2026; Zeng et al., 2025b), while environment-feedback and teacher-guided methods learn self-correction and trial-and-error (Gou et al., 2024; Shinn et al., 2023; Song et al., 2024; Su et al., 2026). Sparse long-horizon rewards motivate turn-level and error-localized supervision (Yu et al., 2025; Zeng et al., 2025a; Luo et al., 2025; Li et al., 2026a; Qu et al., 2026; Tao et al., 2026; Xie et al., 2026; Liang et al., 2026), which matches gold traces, estimates turn contributions, or localizes irrecoverable actions. ToolMaster (Gao et al., 2026) is a related approach to tool trialing; however, it requires an additional teacher model to generate trialing trajectories. TOOLCOMPASS instead guides trialing through shared-function representations, without additional trajectory supervision, frozen turn scorers, or inference overhead.

6 CONCLUSION

We propose TOOLCOMPASS, a novel framework for guiding tool trialing to improve OOD tool use. Our framework leverages function-class assignments and a vMF objective to shape the tool-call representation space, enabling experience to transfer from seen tools to functionally related unseen tools. TOOLCOMPASS requires neither ground-truth call traces nor unseen-tool access, introduces no inference overhead, and can be readily plugged into diverse post-training methods. Extensive experiments show that TOOLCOMPASS consistently improves tool-use performance across benchmarks, model scales, and post-training methods. We hope our work can inspire future research on guided tool trialing and more broadly on building generalizable tool-use agents in the wild.

AI USE STATEMENT

We used generative AI tools to annotate tool domains and function classes on benchmarks and to categorize every tool call in the behavior analyses (Appendix A.5 and Appendix B.8). The LLM-generated annotations are independently reviewed by human annotators, and we report Fleiss's kappa and raw agreement rates in Appendix A.5 and Appendix B.8. Generative AI models also serve as prompting-based baselines in our experiments (Appendix A.6). Generative AI is also used to polish the manuscript. We reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects, personal data, or privacy-sensitive information. The datasets used are established public benchmarks (AppWorld and FTRL) with no personally identifiable information; our annotations describe tool functions and tool-use behavior rather than people. We do not foresee harmful applications or misuse scenarios beyond standard software tool use, and we report no conflicts of interest or sponsorship. We therefore consider that this work raises no ethical concerns requiring further discussion.

REPRODUCIBILITY STATEMENT

Our method is fully described in Section 2 and Section 3, and the experimental setup, benchmarks, and evaluation protocol are given in Section 4.1. Detailed information to reproduce our results is provided in the appendix: Appendix A.2 gives the complete training prompts used for both benchmarks; Appendix A.3 describes the model configurations, hyperparameters, and training procedure, including the temperature, loss weights, and EMA factor used by our method; Appendix A.4 specifies how the tool-call representations are extracted from the policy backbone; Appendix A.6 and Appendix A.8 give the baseline implementations and the full training algorithm; and Appendix A.7 reports the software and hardware used. In addition, we will release the code, together with the annotation protocol and statistics, upon acceptance.

REFERENCES

Anthropic. Claude Opus 4.8 system card. Technical report, Anthropic, May 2026.

Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. Reinforcement learning for long-horizon interactive llm agents. *arXiv preprint arXiv:2502.01600*, 2025.

Weihua Du, Hailei Gong, Zhan Ling, Kang Liu, Lingfeng Shen, Xuesong Yao, Yufei Xu, Dingyuan Shi, Yiming Yang, and Jiecao Chen. Generalizable end-to-end tool-use RL with synthetic codegym. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=QRSeFZfu8E.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. ReTool: Reinforcement learning for strategic tool use in LLMs. In *International Conference on Learning Representations*, volume 2026, pp. 37909–37926, 2026.

Xingjie Gao, Pengcheng Huang, Zhenghao Liu, Yukun Yan, Shuo Wang, Zulong Chen, Chen Qian, Ge Yu, and Yu Gu. Teaching LLMs to learn tool trialing and execution through environment interaction. *arXiv preprint arXiv:2601.12762*, 2026.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, volume 2024, pp. 57734–57811, 2024.

Ruocheng Guo, Kaiwen Dong, Xiang Gao, and Kamalika Das. Learning to rewrite tool descriptions for reliable llm-agent tool use. *arXiv preprint arXiv:2602.20426*, 2026.

Jie He, Jennifer Neville, Mengting Wan, Longqi Yang, Hui Liu, Xiaofeng Xu, Xia Song, Jeff Z Pan, and Pei Zhou. GenTool: Enhancing tool generalization in language models through zero-to-one and weak-to-strong simulation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 1097–1122, 2025.

Shouzheng Huang, Meishan Zhang, Baotian Hu, and Min Zhang. Toolomni: Enabling open-world tool use via agentic learning with proactive retrieval and grounded execution. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 37421–37439, 2026.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in neural information processing systems*, 33:18661–18673, 2020.

Junbo Li, Peng Zhou, Rui Meng, Meet P Vadera, Lihong Li, and Yang Li. Turn-ppo: Turn-level advantage estimation with ppo for improved multi-turn rl in agentic llms. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 6227–6243, 2026a.

Meng Li, Lei Li, Xiting Wang, Yi Yuan, Zheng Wei, Zang Li, et al. Soar: Supervision from observation for agentic reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 35175–35197, 2026b.

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*, pp. 3102–3116, 2023.

Wendi Li, Shawn Im, and Sharon Li. Cyclical entropy eruption: Entropy dynamics in agent reinforcement learning. *arXiv preprint arXiv:2605.27954*, 2026c.

Qiao Liang, Yuke Zhu, Chao Ge, Lei Yang, Ying Shen, Bo Zheng, and Sheng Guo. Learning from the irrecoverable: Error-localized policy optimization for tool-integrated llm reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11008–11028, 2026.

Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *Intelligent Computing*, 3:0063, 2024.

Tengxiao Liu, Zifeng Wang, Jin Miao, I Hsu, Jun Yan, Jiefeng Chen, Rujun Han, Fangyuan Xu, Yanfei Chen, Ke Jiang, et al. Budget-aware tool-use enables effective agent scaling. *arXiv preprint arXiv:2511.17006*, 2025.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K Qiu, and Yuqing Yang. Agent lightning: Train any ai agents with reinforcement learning. *arXiv preprint arXiv:2508.03680*, 2025.

Song-Lin Lv, Weiming Wu, Rui Zhu, Zi-Jian Cheng, and Lan-Zhe Guo. Can agents generalize to the open world? unveiling the fragility of static training in tool use. *arXiv preprint arXiv:2607.01084*, 2026.

Isham Kalappurackal Mansoor, Abhishek Phadke, and Pratip Rana. Verified tool calls improve llm agent reliability under non-atomic failures. *arXiv preprint arXiv:2608.02645*, 2026.

Kanti V Mardia and Peter E Jupp. *Directional statistics*. John Wiley & Sons, 2009.

Dheeraj Mekala, Jason E Weston, Jack Lanchantin, Roberta Raileanu, Maria Lomeli, Jingbo Shang, and Jane Dwivedi-Yu. ToolVerifier: Generalization to new tools via self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 5026–5041, 2024.

Lorenzo Molfetta, Giacomo Frisoni, Nicolò Monaldini, and Gianluca Moro. PORTS: Preference-optimized retrievers for tool selection with large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 10007–10030, 2025.

OpenAI. GPT-5.5 system card. Technical report, OpenAI, April 2026.

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.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru WANG, Xiusi Chen, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. ToolRL: Reward is all tool learning needs. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 105523–105553, 2025. doi: 10.52202/085713-3524.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024.

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.

Changle Qu, Sunhao Dai, Hengyi Cai, Jun Xu, Shuaiqiang Wang, and Dawei Yin. MatchTIR: Fine-grained supervision for tool-integrated reasoning via bipartite matching. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11953–11968, 2026.

Shiori Sagawa, Pang Wei Koh, Tatsunori B. Hashimoto, and Percy Liang. Distributionally robust neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ryxGuJrFvS.

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.

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.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023.

Wentao Shi, Mengqi Yuan, Junkang Wu, Qifan Wang, and Fuli Feng. Direct multi-turn preference optimization for language agents. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7584–7600, 2024.

Junhao Su, Yuanliang Wan, Junwei Yang, Hengyu Shi, Tianyang Han, Yurui Qiu, and Junfeng Luo. Failure makes the agent stronger: Enhancing accuracy through structured reflection for reliable tool interactions. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 12712–12734, 2026.

Michael Sullivan, Mareike Hartmann, and Alexander Koller. Procedural environment generation for tool-use agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 18544–18562, 2025.

Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *European conference on computer vision*, pp. 443–450. Springer, 2016.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. ToolAlpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*, 2023.

Leitian Tao, Baolin Peng, Wenlin Yao, Tao Ge, Hao Cheng, Mike Hang Wang, Jianfeng Gao, and Sharon Li. TRACE: Turn-level reward assignment via credit estimation for long-horizon agents. *arXiv preprint arXiv:2607.13988*, 2026.

Qwen Team. Qwen3.5: Accelerating productivity with native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

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)*, pp. 16022–16076, 2024.

Bin Wu, Edgar Meij, and Emine Yilmaz. Beyond static toolsets: Self-evolving llm tool agents via continual documentation adaptation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 21519–21539, 2026.

Mengsong Wu, Tong Zhu, Han Han, Xiang Zhang, Wenbiao Shao, and Wenliang Chen. Chain-of-tools: Utilizing massive unseen tools in the cot reasoning of frozen language models. *arXiv preprint arXiv:2503.16779*, 2025.

Zhiheng Xi, Xin Guo, Jiaqi Liu, Jiazheng Zhang, Yutao Fan, Zhihao Zhang, Shichun Liu, Mingxu Chai, Xiaowei Shi, Yitao Zhai, et al. Can rl improve generalization of llm agents? an empirical study. *arXiv preprint arXiv:2603.12011*, 2026.

Yutao Xie, Nathaniel Thomas, Nick Hansen, Yang Fu, Li Li, and Xiaolong Wang. TIPS: Turn-level information-potential reward shaping for search-augmented LLMs. In *International Conference on Learning Representations*, volume 2026, pp. 156549–156584, 2026.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. Simple-tir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 8424–8449, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Junjie Ye, Changhao Jiang, Zhengyin Du, Yufei Xu, Xuesong Yao, Zhiheng Xi, Xiaoran Fan, Qi Zhang, Tao Gui, Xuan-Jing Huang, et al. Feedback-driven tool-use improvements in large language models via automated build environments. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 2293–2323, 2026.

Yuanqing Yu, Zhefan Wang, Weizhi Ma, Shuai Wang, Chuhan Wu, Zhiqiang Guo, and Min Zhang. StepTool: Enhancing multi-step tool usage in LLMs via step-grained reinforcement learning. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pp. 3952–3962, 2025.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Siliang Zeng, Quan Wei, William Brown, Oana Frunza, Yuriy Nevmyvaka, Yang Katie Zhao, and Mingyi Hong. Reinforcing multi-turn reasoning in LLM agents via turn-level credit assignment. In *ICML 2025 Workshop on Computer Use Agents*, 2025a.

Yirong Zeng, Xiao Ding, Yutai Hou, Yuxian Wang, Li Du, Juyi Dai, Qiuyang Ding, Duyu Tang, Dandan Tu, Weiwen Liu, et al. Tool zero: Training tool-augmented llms via pure rl from scratch. In *EMNLP (Findings)*, pp. 9135–9147, 2025b.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Luyao Zhuang, Qinggang Zhang, Huachi Zhou, Yujing Zhang, and Xiao Huang. Losemb: Logic-guided semantic bridging for inductive tool retrieval. In *Proceedings of the ACM Web Conference 2026*, pp. 3835–3846, 2026.

A Additional Experimental Details

A.1 Dataset Details

We evaluate ToolCompass on two multi-turn tool-use benchmarks, AppWorld and FTRL, both of which provide executable tools and verifiable task outcomes. For each benchmark, we describe the dataset contents, the ID/OOD split construction, and the evaluation protocol below.

AppWorld. We use the publicly released AppWorld benchmark (Trivedi et al., 2024),[1] with 90 training tasks across 30 scenarios and 473 APIs. The benchmark provides two official test splits: Test-C contains all tasks that require at least one API from the designated unseen applications (i.e., Amazon and Gmail), and Test-N contains the test tasks that do not require them. Therefore, we treat Test-N as the ID test set and Test-C as the OOD test set, containing 168 and 417 tasks, respectively. No training task requires the unseen applications. Each task is scored by the official state-based evaluator, which runs task-specific tests against the final environment state. Post-training uses the outcome reward returned by this evaluator. We report the task success rate, i.e., the percentage of tasks that pass all evaluation tests. Results under the stricter Scenario Goal Completion (SGC) metric are provided in Appendix B.6.

FTRL. FTRL (Ye et al., 2026) covers single-hop and multi-hop tool-use queries, each associated with a subject domain. To construct a covariate-shift OOD split, we hold out five domains: Health, Medicine and Public Health; Music and Performing Arts; Politics, Governance and Law; Sports and Competitions; and Zoology and Animal Science. Their tools are excluded during post-training, while their function classes remain covered by the seen domains. The test set contains 168 ID queries and 32 OOD queries; an OOD query requires at least one held-out-domain tool. Post-training uses 1,615 training instances that do not require these tools. Following the official protocol, we compute the metrics per query: for a trajectory with p tool calls that solves q of the n required sub-questions, Solve-P $= q/p$ measures tool-invocation precision, Solve-R $= q/n$ measures task-completion recall, and Solve-F1 is their harmonic mean.

A.2 Input Prompts

We use the official task prompts of AppWorld (Trivedi et al., 2024) and FTRL (Ye et al., 2026), shown in Figures 8 and 9.

A.3 Implementation Details

General setup. We use Qwen3.5-4B and Qwen3.5-9B (Team, 2026) as policy backbones and adopt the same ReAct interaction scaffold (Yao et al., 2023) for all methods. Unless otherwise specified, ToolCompass uses GRPO as the host objective. All trainable methods are optimized with Adam. Following AppWorld (Trivedi et al., 2024), we select hyperparameters on its official development split. On FTRL (Ye et al., 2026), we follow MatchTIR's released protocol (Qu et al., 2026) for hyperparameter selection. We apply the same selection protocol to ToolCompass and all baselines. The default online training configurations for both benchmarks are summarized in Table 5.

Interaction and inference. We follow the released interaction protocol of each benchmark. In AppWorld, the agent retrieves app descriptions and API specifications on demand through the `api_docs` interface, and the specifications of Amazon and Gmail become available only at evaluation time. In FTRL, each query is paired with a fixed candidate set of tool schemas, which is passed to the policy through the native tool-calling interface. Training rollouts and evaluation both use sampled decoding with temperature 1.0 and top-$p = 1.0$. Each reported three-seed result trains and evaluates the complete pipeline with three random seeds, which control the initialization of the projection head and prototypes, the data order, and rollout sampling.

ToolCompass configuration. We compute the auxiliary loss only on turns containing tool calls. For each tool call, we extract the hidden state of the call's final token from layer $l = 8$ of the policy backbone; Appendix A.4 illustrates the extraction. The projection head g_ψ is implemented as a two-layer MLP with output dimension $d = 128$. Function prototypes are randomly initialized as unit vectors and updated with an exponential moving average (EMA). We set the temperature $\tau = 0.4$,

[1] https://github.com/StonyBrookNLP/appworld

AppWorld ReAct Prompt

```
I am your supervisor and you are a super intelligent AI Assistant whose job is
to achieve my day-to-day tasks completely autonomously.

You will interact with apps using their associated APIs through a multi-step
conversation in a Python REPL. Write Python code; the environment will execute
it and return the result, which you can use in the next step.

# List the available apps.
print(apis.api_docs.show_app_descriptions())

# List the APIs of an app.
print(apis.api_docs.show_api_descriptions(app_name='<app_name>'))

# Show the specification of an API.
print(apis.api_docs.show_api_doc(
    app_name='<app_name>', api_name='<api_name>'))

Use only the provided APIs. Write one small code block at each step and use
results from previous steps when needed. When the task is complete, call
apis.supervisor.complete_task(); if an answer is required, pass it through the
answer argument.

My name is: {supervisor_first_name} {supervisor_last_name}.
My personal email is {supervisor_email} and phone number is
{supervisor_phone_number}.

Your task is: {task_description}

Reason about the next API call within <think> </think> tags and place the code
body within <code> </code> tags.
```

Figure 8: Task-facing AppWorld ReAct prompt. The demonstration and standard environment disclaimers in the released prompt are omitted for space.

FTRL ReAct Prompt

```
Please call given tools to answer the question. Please note that all your
information must be obtained by calling tools and not by answering the question
directly. If the call fails, you need to try to correct it and continue until
you arrive at an answer.

Question: {question}
```

Figure 9: Task prompt used for FTRL. The query-specific tool schemas are supplied through the native tool-calling interface.

the loss weights $\lambda_{\text{vMF}} = 0.02$ and $\lambda_{\text{sep}} = 2$, and the EMA factor $\alpha = 0.9$. Both the projection head and the prototypes are not used at deployment, so TOOLCOMPASS adds no inference-time overhead. For the distance-metric ablation in Figure 6(b), we replace the cosine score $\mathbf{a}^\top \mathbf{b}/\tau$ with the negative unsquared Euclidean score $-\|\mathbf{a}-\mathbf{b}\|_2/\tau$ in both $\mathcal{L}_{\text{var}}$ and $\mathcal{L}_{\text{sep}}$. Here, $\mathbf{a}$ is the unit-normalized tool-call representation or batch direction, and $\mathbf{b}$ is the unit prototype. Both variants use $\tau = 0.4$, with the projection head, EMA update, and loss weights unchanged.

A.4 TOOL-CALL REPRESENTATION EXTRACTION

TOOLCOMPASS represents each tool call by the hidden state of the call's final token. When a code block contains multiple API calls, we extract a separate representation for each call at its own final token. Figure 10 illustrates a code block with two API calls. For call i, we take the layer-$l = 8$ hidden state at its final token as $\mathbf{h}_i$ and compute $\mathbf{z}_i = \text{Normalize}(g_\psi(\mathbf{h}_i))$. The two calls therefore contribute two representations to the representation-shaping objective. The same extraction rule applies to the FTRL tool-calling format.

Table 5: Default online training configuration.

Hyperparameter	AppWorld	FTRL
Prompts per update	40	256
Rollouts per prompt	6	16
Global batch size	240	4,096
Maximum training turns	40	10
Maximum evaluation turns	40	20
Maximum generated tokens per turn	2,000	4,096
Maximum response length	2,000 (per turn)	23,000
Learning rate	1×10^{-6}	
Weight decay	0.01	
KL coefficient	0.001	
Entropy coefficient	0.001	
Clipping range	0.2	
Gradient clipping	1.0	
Training duration	10 epochs	3 epochs

Tool-Call Representation Extraction

```
<think> I first find the artist, then search for their songs. </think>
<code>
artists = apis.spotify.search_artists(query="Lily Moon")  # call 1 ends here
artist_id = artists[0]["artist_id"]
songs = apis.spotify.search_songs(
    query="Lily Moon", artist_id=artist_id
)  # call 2 ends here
</code>
```

Figure 10: Tool-call representation extraction from a code block containing two AppWorld API calls. The final token of each call provides its own layer-$l = 8$ hidden state, $\mathbf{h}_1$ and $\mathbf{h}_2$, which are independently projected and normalized.

A.5 Domain and Function-Class Annotations

Following Zheng et al. (2023), we use GPT-5.6-sol to annotate the tools in both benchmarks with domain labels and function-class labels.

Domain annotation. Both benchmarks share the same domain-annotation prompt, shown in Figure 11; the closed taxonomy is instantiated per benchmark. This yields nine application domains for AppWorld and 27 subject domains for FTRL. The ten AppWorld meta APIs (`api_docs` and `supervisor`) are infrastructure and receive no domain label.

Function-class annotation. GPT-5.6-sol assigns a function-class label to each of the 473 AppWorld APIs and 4,545 FTRL tools. The 463 ordinary AppWorld APIs are grouped into 14 function classes, while the ten meta APIs are labeled META and excluded from the representation-shaping objective; the FTRL tools are grouped into nine function classes. Only the labels of tools observed during post-training enter the objective, and the annotations of OOD tools are never accessed. The prompts are shown in Figures 12 and 13, and Table 6 summarizes the statistics of the resulting annotations.

Annotation verification. To assess annotation quality, three annotators independently verify all 463 AppWorld APIs and a random sample of 200 FTRL tools, and we compute Fleiss's kappa over the three sets of labels. The kappa scores are 0.86 on AppWorld and 0.83 on FTRL, with raw agreement rates of 88.77% and 85.17%, respectively; the remaining disagreements are resolved by discussion. The strong agreement indicates that the function classes can be consistently identified from tool specifications.

Domain Annotation Prompt

```
You label software tools by their DOMAIN.

The domain is the real-world topic or application area that a tool
operates on, not the operation it performs. Use the tool description as the primary
evidence. The tool name is supporting evidence only. Choose exactly one
domain from the closed taxonomy below:

{domain_taxonomy}

Important boundaries:
- Judge the topic of the data the tool reads or writes, not the action
  type; functional intent is labeled separately.
- A tool that searches, computes, or verifies within one topic area
  belongs to that area.
- If several domains are involved, label the dominant one. Lower
  confidence when the description is genuinely underspecified.

Input:
tool_name: {tool_name}
description: {description}

Return strict JSON only:
{"domain": "<ONE_DOMAIN>", "reason": "<one sentence>",
 "confidence": <0.0--1.0>}
```

Figure 11: Prompt used to annotate tool domains with GPT-5.6-sol. The closed taxonomy is instantiated per benchmark.

Table 6: Statistics of the function-class annotations. The ten AppWorld meta APIs are excluded from the representation-shaping objective.

Dataset	Tools/APIs	Function classes	Average class size
AppWorld	463	14	33.1
FTRL	4,545	9	505.0

A.6 Baseline Details

We follow the baseline formulations in the cited papers and use the interaction and evaluation protocols described in Appendix A.3. Baseline-specific objectives are summarized below.

Prompting-based agents. GPT-5.5 (OpenAI, 2026), Claude Opus 4.8 (Anthropic, 2026), GLM-5.2 (Zeng et al., 2026), and DeepSeek V4 Pro (Xu et al., 2026) are evaluated without any post-training. Each model receives the same ReAct prompt, tool specifications, observations, and interaction budget as in the main setup.

General post-training objectives. GRPO (Shao et al., 2024) optimizes the outcome reward and applies the same trajectory-level advantage to all tool calls in a rollout. RFT (Yuan et al., 2023) fine-tunes on successful rollouts selected by the task evaluator, while DMPO (Shi et al., 2024) constructs trajectory-level preference pairs from the same evaluator scores.

Turn-level tool-use post-training methods. StepTool (Yu et al., 2025) and FTRL-M (Ye et al., 2026) assign turn-level rewards, where FTRL-M denotes the multi-turn variant of FTRL. MatchTIR (Qu et al., 2026) uses the KM variant with AppWorld's reference API calls and FTRL's released tool-call annotations, combining turn-level and trajectory-level advantages. SOAR (Li et al., 2026b) derives supervision from observation tokens, while TRACE (Tao et al., 2026) follows the authors' turn-level credit-assignment setup.

Tool-use RL methods. SimpleTIR (Xue et al., 2026) applies void-turn filtering during training, and ToolMaster (Gao et al., 2026) combines trajectory-based supervised fine-tuning with subsequent reinforcement learning. LOOP (Chen et al., 2025) retains its leave-one-out advantage estimation and rollout reuse.

AppWorld Annotation Prompt

```
You label AppWorld APIs by their dominant externally visible function. Classify
what the API does, rather than its application, domain, or surface name. Choose
exactly one family from the closed taxonomy below:

- AUTH_ACTION: Manage an account, authentication, authorization, or session
  lifecycle.
- CHECK_STATE: Check existence or make an explicit state judgment.
- CONTROL_ACTION: Control a transient process, such as playback or a device.
- CREATE_OBJECT: Create a new record, object, or piece of content.
- DELETE_ACTION: Delete, cancel, or remove an object.
- ORGANIZE_ACTION: Change membership, order, location, or association.
- READ_ITEM: Retrieve a known item, detail, scalar value, or state.
- READ_LIST: Search, discover, filter, or enumerate candidate items.
- SEND_ACTION: Send, reply, forward, invite, notify, or remind.
- TOGGLE_ACTION: Change a discrete flag or lifecycle status.
- TRANSFER_ACTION: Move money or assets between parties.
- TRANSFORM_ACTION: Copy, compress, decompress, convert, or transform supplied
  content.
- UPDATE_OBJECT: Modify fields or content of an existing object.
- WORKFLOW_ACTION: Commit an order, return, subscription, approval, or related
  application workflow.

Important boundaries:
- READ_LIST returns a collection; READ_ITEM returns one known item or property.
- UPDATE_OBJECT changes free-form content; TOGGLE_ACTION changes a discrete
  status; CONTROL_ACTION controls a transient process.
- ORGANIZE_ACTION changes a relationship or collection without creating the
  primary object.
- Use TRANSFER_ACTION whenever money or assets move, even if the API name
  contains "create" or "record."
- Reserve WORKFLOW_ACTION for application-level commitments.
- If the API belongs to api_docs or supervisor, return META.META; otherwise
  choose one of the 14 families above.

Input:
tool_name: {tool_name}
description: {description}
arguments: {arguments}
response: {response_specification}

Return strict JSON only:
{"family": "<ONE_FAMILY>", "reason": "<one sentence>",
 "confidence": <0.0--1.0>}
```

Figure 12: Prompt used to annotate AppWorld APIs with GPT-5.6-sol.

Representation-learning method. SEAL (Li et al., 2026c) is run with its released implementation without further modification.

OOD-generalization methods. CORAL (Sun & Saenko, 2016) aligns representation covariances, while Group DRO (Sagawa et al., 2020) reweights training-group losses. On AppWorld, CORAL aligns tool-call representations across training applications, and Group DRO reweights application-group policy losses. On FTRL, we assign each tool call to the subject-domain label of the invoked tool, so calls within a mixed-domain query may belong to different groups. CORAL aligns representation covariances across source-training groups, whereas Group DRO computes and reweights group-wise policy losses; held-out domains are excluded from both. ToolRL (Qian et al., 2025) and PAFT (Lv et al., 2026) use their tool-call reward and training-time trajectory perturbations, respectively.

A.7 Compute Resources and Time

Software and hardware. We conduct all experiments using Python 3.12.13 and PyTorch 2.11.0 with CUDA 12.8 on four NVIDIA H200 GPUs with 141 GB memory.

FTRL Annotation Prompt

```
You label software tools by their dominant FUNCTIONAL INTENT.

Use the tool description as the primary evidence. The tool name is supporting
evidence only. Choose exactly one family from the closed taxonomy below:

- RETRIEVE_PROVIDE: Retrieve or provide factual information about a known
  subject, item, record, or property; no discovery, comparison, verification,
  or transformation is dominant.
- SEARCH_LOCATE: Find, search, query, filter, or locate candidate entities,
  people, places, records, works, or resources.
- IDENTIFY_CLASSIFY: Determine identity, type, category, author,
  creator, owner, origin, membership, or another categorical attribute.
- ANALYZE_SUMMARIZE: Analyze, explain, synthesize, interpret, correlate, or
  summarize information rather than merely retrieving one fact.
- CALCULATE_MEASURE: Calculate, count, estimate, aggregate, convert, measure,
  or derive a numeric value.
- COMPARE_SELECT: Compare, rank, score, optimize, recommend, or select among
  alternatives, including superlatives.
- TRACK_FORECAST: Track or monitor change over time, reconstruct progression or
  history, or forecast a future value or state.
- VERIFY_VALIDATE: Check, verify, validate, test, assess compliance,
  availability, or consistency, or return a yes/no validity judgment.
- TRANSFORM_EXTRACT: Extract structured content, parse, translate, normalize,
  convert representation, map, or transform supplied content.

Important boundaries:
- SEARCH_LOCATE discovers candidate entities; RETRIEVE_PROVIDE fetches facts
  about a subject already specified by the caller.
- IDENTIFY_CLASSIFY resolves identity, type, category, origin, or ownership.
- COMPARE_SELECT includes ranking, recommendation, best, nearest, largest, and
  choosing among candidates.
- VERIFY_VALIDATE is an explicit check or validity judgment, not an ordinary
  lookup.
- CALCULATE_MEASURE must have a derived numeric result as the dominant
  function.
- TRACK_FORECAST concerns temporal progression or prediction.
- TRANSFORM_EXTRACT operates on supplied content or changes its representation.
- If several operations are mentioned, label the dominant externally visible
  result. Lower confidence when the description is genuinely underspecified.

Input:
tool_name: {tool_name}
description: {description}

Return strict JSON only:
{"family": "<ONE_FAMILY>", "primitive": "<concise verb phrase>",
 "reason": "<one sentence>", "confidence": <0.0--1.0>}
```

Figure 13: Prompt used to annotate FTRL tools with GPT-5.6-sol.

Training time. With Qwen3.5-4B, a complete TOOLCOMPASS run takes approximately 6.6 hours on AppWorld and 21.9 hours on FTRL. Compared with vanilla GRPO under the same configuration, the projection head, the prototype updates, and the representation-shaping losses add less than 1% training time.

A.8 TRAINING ALGORITHM

We provide the complete post-training procedure of TOOLCOMPASS in Algorithm 1, which summarizes the joint optimization of the host objective and the representation-shaping objective, together with the EMA prototype updates.

Algorithm 1 Post-training with TOOLCOMPASS

Require: Policy π_θ; projection head g_ψ; function labels $c(u)$; host objective $\mathcal{L}_{\text{post}}$; layer l; temperature τ; loss weights λ_{vMF} and λ_{sep}; EMA factor α

Randomly initialize the projection head g_ψ and unit prototypes $\{\boldsymbol{\mu}_k\}_{k\in\mathcal{C}}$
for each post-training update **do**
  Sample a batch of trajectories with π_θ and compute $\mathcal{L}_{\text{post}}$
  Initialize the minibatch of tool-call representations $\mathcal{B} \leftarrow \varnothing$
  for each observed tool call a_i invoking tool u_i **do**
    Extract $\mathbf{h}_i$ from layer l (Eq. 1) and compute $\mathbf{z}_i = \text{Normalize}(g_\psi(\mathbf{h}_i))$
    Assign $c_i = c(u_i)$ and add $(\mathbf{z}_i, c_i)$ to $\mathcal{B}$
  end for
  Compute normalized batch directions $\bar{\mathbf{z}}_k$ for classes observed in $\mathcal{B}$
  Compute $\mathcal{L}_{\text{var}}$ (Eq. 3) and $\mathcal{L}_{\text{sep}}$ (Eq. 4) from $\mathcal{B}$, treating the prototypes as stop-gradient targets
  Update θ and ψ using $\mathcal{L}_{\text{post}} + \lambda_{\text{vMF}}(\mathcal{L}_{\text{var}} + \lambda_{\text{sep}}\mathcal{L}_{\text{sep}})$ (Eqs. 6 and 7)
  for each function class k observed in $\mathcal{B}$ **do**
    Update $\boldsymbol{\mu}_k \leftarrow \text{Normalize}(\alpha\boldsymbol{\mu}_k + (1-\alpha)\bar{\mathbf{z}}_k)$ (Eq. 5)
  end for
end for
return the post-trained policy π_θ

B ADDITIONAL ANALYSIS

Unless otherwise specified, we use Qwen3.5-4B with GRPO as the host objective.

B.1 SEMANTIC OOD ANALYSIS

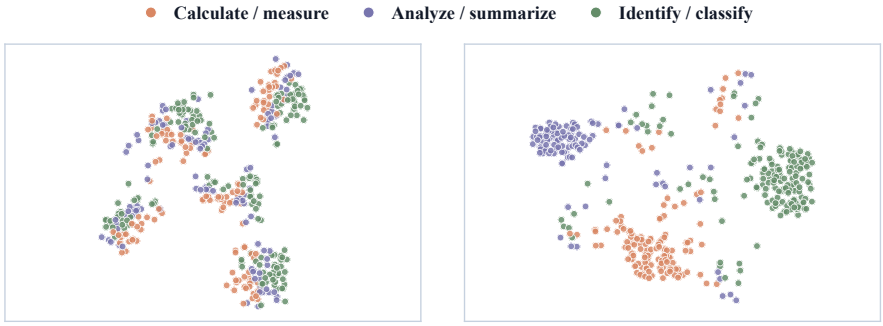
We evaluate semantic OOD transfer on FTRL by jointly holding out `CALCULATE_MEASURE`, `ANALYZE_SUMMARIZE`, and `IDENTIFY_CLASSIFY` during post-training. GRPO and TOOLCOMPASS use the same Qwen3.5-4B initialization and filtered training data. We hold out 300 examples from the training data as a validation set. Evaluation includes 92 test queries requiring at least one held-out function. Table 7 reports per-query Solve-P, Solve-R, and Solve-F1 averaged over these queries; other configurations follow Appendix A.3.

Table 7: Semantic OOD results (%) on FTRL using Qwen3.5-4B.

Method	Solve-P	Solve-R	Solve-F1
GRPO	50.84	64.09	52.04
TOOLCOMPASS	**53.02**	**70.21**	**59.11**

Table 7 shows that TOOLCOMPASS improves Solve-F1 over GRPO by 7.07 points on semantic OOD tasks, with gains of 2.18 and 6.12 points in Solve-P and Solve-R, respectively. We attribute this gain to function-level supervision, which encourages the shared backbone to capture operational intent across tools. Together with pretrained semantic knowledge and tool descriptions, this abstraction helps the policy select appropriate calls for functions unseen during post-training.

To further examine this transfer, we visualize tool-call representations for the three held-out functions with t-SNE (Figure 14). The visualization shows more coherent within-function groups and clearer separation under TOOLCOMPASS than GRPO, illustrating how function-level organization can extend to unseen functions.



(a) GRPO (b) ToolCompass

Figure 14: **t-SNE visualization of jointly held-out functions.** Colors denote the three function classes excluded from post-training. The two panels compare tool-call representations under GRPO and TOOLCOMPASS.

B.2 Comparison of Auxiliary Objectives

We compare ToolCompass with Function CE (linear classification) and SupCon (Khosla et al., 2020) under the same function labels, representation layer, projection head, normalization, and training budget. Each objective uses the same development-set tuning budget. SupCon uses temperature $\tau = 0.07$, with other calls sharing the same function label as positives and self-pairs excluded. Anchors without a positive are assigned zero loss and included in the batch average, while remaining available as negatives for other anchors.

Table 8: Comparison of auxiliary objectives on AppWorld with Qwen3.5-4B and GRPO. Values are task success rates (%).

Method	ID	OOD	Total
GRPO	58.33	56.83	57.26
GRPO + Function CE	69.05	57.31	60.68
GRPO + SupCon	60.71	61.63	61.37
ToolCompass	**70.63**	**64.75**	**66.44**

ToolCompass outperforms Function CE and SupCon by 7.44 and 3.12 percentage points in OOD task success, respectively, supporting the benefit of the proposed objective beyond generic function supervision.

B.3 Effect of Loss Weights

ToolCompass uses two weighting hyperparameters: λ_{vMF} balances the representation-shaping objective against the host objective, and λ_{sep} controls the strength of inter-function separation within it. We independently vary each weight while fixing the other to its default value, and report the task success rate in Figure 15.

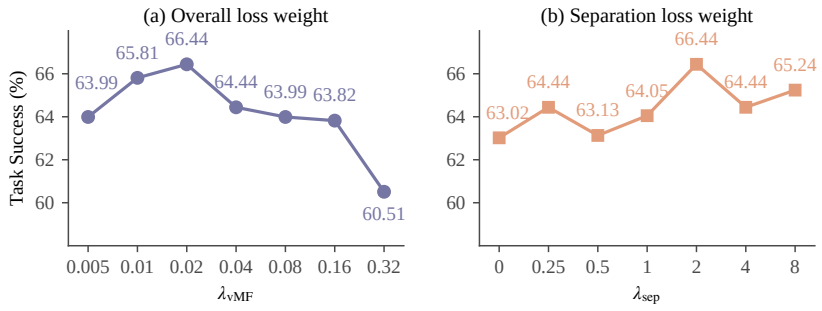


Figure 15: **Effect of loss weights on AppWorld task success.** We vary one weight while fixing the other to its default value.

As shown in Figure 15(a), performance peaks at $\lambda_{\text{vMF}} = 0.02$, reaching 66.44%, and remains competitive for nearby values, indicating that TOOLCOMPASS is not overly sensitive to the precise choice of this weight. A very small weight provides insufficient function-level supervision, whereas an overly large weight overemphasizes representation shaping relative to the host objective. Figure 15(b) shows a similar trend for λ_{sep}, which performs best at 2.0 with 66.44%: weaker separation does not sufficiently distinguish different function classes, while stronger separation reduces the flexibility within each class.

B.4 EFFECT OF THE EMA FACTOR

We vary the EMA factor α of the prototype update in Eq. 5 while keeping all other hyperparameters at their default values. The endpoint $\alpha = 0$ reduces to the batch-estimated prototypes in Figure 6(a), and $\alpha = 1.0$ freezes the prototypes at their initial estimates. As shown in Figure 16, performance peaks at $\alpha = 0.9$, reaching 66.44%: a small α lets the prototypes fluctuate with individual minibatches, while a large α makes them adapt too slowly to the evolving representations.

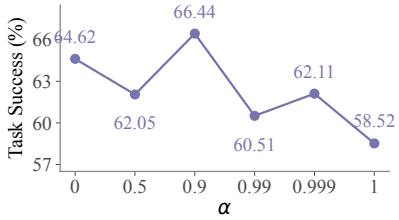


Figure 16: **Effect of the EMA factor α on AppWorld task success.**

B.5 QUALITATIVE ANALYSIS

We present two AppWorld Test-C case studies to show how TOOLCOMPASS guides tool trialing on tasks that require the unseen applications. For each task, we show the complete trajectories of GRPO, MatchTIR, and TOOLCOMPASS, and every call is labeled as direct tool use, useful trialing, or unproductive trialing following the protocol in Appendix B.8. The task instructions, resource identifiers, and environment observations come from the executable benchmark instances; authentication and API-documentation calls are omitted. Sample 1 (Figure 17) requires comparing two gift-wrapping options and completing a checkout on Amazon. Sample 2 (Figure 18) requires replacing an attachment in a Gmail draft with a file from the file system.

The two cases show the same pattern from different sides. TOOLCOMPASS explores when exploration is needed: its trials stay on functionally related APIs, and each error is followed by a corrective call. GRPO explores without guidance, which wastes the budget in Sample 2 and causes a failure in Sample 1. MatchTIR suppresses exploration and terminates before the required final actions in both cases. This matches the aggregate behavior in Figure 7: TOOLCOMPASS keeps the useful trials while cutting the unproductive ones.

B.6 APPWORLD SGC RESULTS

In addition to the task success rate, we report Scenario Goal Completion (SGC), a stricter metric that credits a scenario only when all of its task variants are completed successfully. The results are shown in Table 9.

Table 9: AppWorld SGC results (%) using Qwen3.5-4B, averaged over three seeds.

Method	ID	OOD	Total
LOOP	39.88	24.22	28.72
SEAL	41.07	26.38	30.60
TOOLCOMPASS	**48.21**	**29.50**	**34.87**

As shown in Table 9, TOOLCOMPASS achieves the best SGC on all splits. In particular, it improves OOD SGC by 5.28 points over LOOP and by 3.12 points over SEAL. These results indicate that the gains of TOOLCOMPASS extend beyond individual tasks and remain consistent across the task variants of the same scenario.

Sample 1

Task. Buy one Samsung Galaxy Buds Pro and use the cheaper option: Amazon direct gift wrapping or a separately purchased wrapper. **OOD app.** Amazon

AppWorld Test-C `ec437da_2`. Target: product 735 ($200); cheapest separate wrapper: product 2323 ($10); direct wrap fee: $5.

D: Direct tool use **U**: Useful trialing **P**: Unproductive trialing

GRPO — Failure

T1 P amazon.show_wish_list() → empty; task never uses it.
T2 P amazon.show_recommended_products() → unrelated products.
T3 U amazon.search_products("Samsung Galaxy Buds") → several models; refine the name.
T4 D amazon.search_products("Samsung Galaxy Buds Pro") → id 735, $200.
T5 D amazon.show_product(735) → inventory 8.
T6 P amazon.show_product_rating_distribution(735) → ratings do not price wrapping.
T7 U amazon.show_product_reviews(735) → no fee stated; inspect Q&A.
T8 U amazon.show_product_questions(735) → no fee stated; compare wrapper listings.
T9 U amazon.search_products("gift wrap", max_price=3) → empty; raise ceiling.
T10 U amazon.search_products("gift wrap", max_price=5) → empty; raise again.
T11 U amazon.search_products("gift wrap", max_price=8) → empty; remove ceiling.
T12 U amazon.search_products("gift bag", sort_by="price") → cheapest candidate $12.
T13 U amazon.search_products("wrapping paper", sort_by="price") → candidate id 2323, $10.
T14 D amazon.search_products("gift wrap", sort_by="price") → confirms id 2323 is cheapest, $10.
T15 U amazon.add_gift_wrapping_to_product(735,1) → product not in cart.
T16 P amazon.update_product_quantity_in_cart(735,1) → repeats the cart prerequisite.
T17 P amazon.delete_product_from_cart(735) → repeats the same prerequisite.
T18 D amazon.add_product_to_cart(735,1,True) → added.
T19 U amazon.add_gift_wrapping_to_product(735,2) → cart only has 1.
T20 P amazon.add_gift_wrapping_to_product(735,2) → repeats T19 unchanged.
T21 D amazon.add_gift_wrapping_to_product(735,1) → 1/1 marked.
T22 D amazon.show_cart() → gift-wrap fee $5, below $10.
T23 U amazon.place_order(1,1) → invalid owned-resource IDs; retrieve account IDs.
T24 D amazon.show_payment_cards() → active card 410.
T25 D amazon.show_addresses() → Home address 183.
T26 P amazon.show_browsing_history() → unrelated history.
T27 P amazon.show_orders() → no current order.
T28 P amazon.show_recommended_products() → repeats unrelated suggestions.
T29 P amazon.show_wish_list() → still irrelevant.
T30 P amazon.show_product_reviews(735) → repeats T7.
T31 P amazon.show_product_questions(735) → repeats T8.
T32 P amazon.show_cart() → unchanged; no progress toward checkout.
T33 P amazon.show_product_purchases(735) → unrelated purchase records.
T34 P amazon.show_product_feature_choices(735) → rechecks a resolved branch.
T35 P amazon.show_cart() → unchanged again; trajectory terminates.

MatchTIR — Failure

T1 D amazon.search_products("Samsung Galaxy Buds Pro") → id 735, $200.
T2 D amazon.show_product(735) → inventory 8.
T3 P amazon.show_product_rating_distribution(735) → irrelevant to wrap price.
T4 U amazon.search_products("gift wrap", max_price=5) → empty; remove ceiling.
T5 D amazon.search_products("gift wrap", sort_by="price") → id 2323, $10.
T6 D amazon.show_product(2323) → separate wrapper is available.
T7 P amazon.search_products("gift bag") → no cheaper option than T5.
T8 U amazon.add_gift_wrapping_to_product(735,1) → product not in cart.
T9 P amazon.update_product_quantity_in_cart(735,1) → repeats the same prerequisite.
T10 D amazon.add_product_to_cart(735,1,True) → added.
T11 U amazon.add_gift_wrapping_to_product(735,2) → cart only has 1.
T12 D amazon.add_gift_wrapping_to_product(735,1) → 1/1 marked.
T13 D amazon.show_cart() → direct fee $5, cheaper than $10.
T14 P amazon.show_browsing_history() → unrelated history.
T15 P amazon.show_orders() → no current order; trajectory stops before checkout.

TOOLCOMPASS — Success

T1 P amazon.show_wish_list() → empty; not part of the task.
T2 U amazon.search_products("Samsung Galaxy Buds") → multiple models; refine query.
T3 D amazon.search_products("Samsung Galaxy Buds Pro") → id 735, $200.
T4 D amazon.show_product(735) → inventory 8.
T5 P amazon.show_product_rating_distribution(735) → no price evidence.
T6 U amazon.show_product_reviews(735) → no wrap fee; inspect Q&A.
T7 U amazon.show_product_questions(735) → no fee; test product options.
T8 U amazon.show_product_feature_choices(735) → no wrap variant; use cart-level API.
T9 U amazon.search_products("gift wrap", max_price=3) → empty; raise ceiling.
T10 U amazon.search_products("gift wrap", max_price=5) → empty; raise again.
T11 U amazon.search_products("gift wrap", max_price=8) → empty; remove ceiling.
T12 U amazon.search_products("gift bag", sort_by="price") → cheapest candidate $12.
T13 U amazon.search_products("wrapping paper", sort_by="price") → id 2323, $10.
T14 U amazon.search_products("gift box", max_price=5) → empty; $5 threshold stands.
T15 U amazon.search_products("gift wrapper", sort_by="price") → same id 2323; synonym resolved.
T16 D amazon.search_products("gift wrap", sort_by="price") → confirms id 2323, $10.
T17 D amazon.show_product(2323) → wrapper available.
T18 U amazon.add_gift_wrapping_to_product(735,1) → product not in cart.
T19 D amazon.add_product_to_cart(735,1,True) → added.
T20 U amazon.add_gift_wrapping_to_product(735,2) → cart only has 1.
T21 D amazon.add_gift_wrapping_to_product(735,1) → 1/1 marked.
T22 D amazon.show_cart() → direct fee $5, below $10.
T23 P amazon.show_browsing_history() → unrelated history.
T24 U amazon.place_order(1,1) → invalid owned-resource IDs; retrieve them.
T25 D amazon.show_payment_cards() → active card 410.
T26 D amazon.show_addresses() → Home address 183.
T27 P amazon.show_orders() → no current order; no new signal.
T28 P amazon.show_recommended_products() → unrelated suggestions.
T29 D amazon.place_order(410,183) → order created; gift-wrap quantity 1.

Method	Calls	Direct	Useful	Unproductive	Outcome
GRPO	35	8	11	16	Failure
MatchTIR	15	7	3	5	Failure
TOOLCOMPASS	29	10	14	5	Success

Figure 17: **Sample 1 (Amazon: compare gift-wrapping options and check out).** TOOLCOMPASS turns each error into a corrective next step and keeps its trials on checkout-relevant APIs, completing the order in 29 calls with only 5 unproductive ones. GRPO finds the same evidence but spends 16 calls on unrelated or repeated queries and exhausts the interaction budget before checkout. MatchTIR stops after 15 calls without attempting checkout.

Sample 2

Task. In the reply draft to my manager, replace the mistakenly attached resume with the headshot stored in my file system, then send the email. **OOD app.** Gmail

AppWorld Test-C `bcb9696_1`. Manager: Brandon Weber; target draft 518; resume attachment 8483; distractor thread 47823.

D: Direct tool use **U**: Useful trialing **P**: Unproductive trialing

GRPO — Success

```
T1 U gmail.show_inbox_threads("manager") → many threads; refine by headshot.
T2 D gmail.show_inbox_threads("headshot") → threads 47821 and 47823.
T3 U gmail.show_thread(47823) → Adrian, not the manager; rule out.
T4 D gmail.show_thread(47821) → Brandon request, email 20867.
T5 U gmail.show_drafts(query="headshot") → empty; filter by recipient.
T6 D gmail.show_drafts(recipient_email= "brandon-webe@gmail.com") → draft 518.
T7 P gmail.show_outbox_threads("headshot") → distractor only.
T8 P gmail.show_spam_threads("headshot") → empty.
T9 P gmail.show_snoozed_threads("headshot") → empty.
T10 U gmail.remove_attachment_from_ draft(518,20867) → no such attachment; inspect draft IDs.
T11 D gmail.show_draft(518) → resume attachment 8483.
T12 D gmail.remove_attachment_from_ draft(518,8483) → deleted.
T13 U file_system.file_exists("/home/ norman/headshot.jpeg") → false; search directories.
T14 P file_system.show_file("/home/norman/ headshot.jpeg") → repeats T13 as an error.
T15 U file_system.show_directory("/home/ norman","headshot","files",False) → empty; descend.
T16 U file_system.show_directory("/home/ norman/documents","headshot","files",False) → empty; recurse.
T17 U file_system.show_directory("/home/ norman/documents","headshot","files",True) → exact work path.
T18 D file_system.show_file(".../work/ professional_headshot.jpeg") → target file.
T19 P file_system.show_directory("/", "resume","files") → irrelevant files.
T20 P file_system.show_directory("/home/ norman/downloads") → irrelevant folder.
T21 U gmail.upload_attachments_to_draft(518,[]) → at least one path required.
T22 P gmail.upload_attachments_to_ draft(518,["/home/norman/headshot.jpeg"]) → repeats resolved path error.
T23 D gmail.upload_attachments_to_draft(518,["... /professional_headshot.jpeg"]) → id 8485.
T24 P gmail.show_thread(47823) → repeats distractor branch.
T25 P gmail.show_outbox_threads("Brandon") → premature check.
T26 P gmail.show_inbox_threads("headshot") → repeats T2.
T27 D gmail.show_draft(518) → only headshot 8485 remains.
T28 D gmail.send_email_from_draft(518) → sent email 20870 to Brandon.
```

MatchTIR — Failure

```
T1 D gmail.show_inbox_threads("headshot") → threads 47821 and 47823.
T2 U gmail.show_thread(47823) → Adrian, not the manager; rule out.
T3 D gmail.show_thread(47821) → Brandon request, email 20867.
T4 D gmail.show_email(20867) → confirms the headshot request.
T5 P gmail.show_email(20869) → revisits distractor evidence.
T6 D gmail.show_drafts(recipient_email= "brandon-webe@gmail.com") → draft 518.
T7 P gmail.show_outbox_threads("headshot") → no target draft.
T8 P gmail.show_archived_threads("headshot") → empty.
T9 P gmail.show_starred_threads("headshot") → empty.
T10 P gmail.show_spam_threads("headshot") → empty.
T11 U gmail.remove_attachment_from_ draft(518,20867) → no such attachment; inspect IDs.
T12 D gmail.show_draft(518) → resume attachment 8483.
T13 D gmail.remove_attachment_from_ draft(518,8483) → deleted.
T14 U file_system.file_exists("/home/ norman/headshot.jpeg") → false.
T15 P file_system.show_file("/home/norman/ headshot.jpeg") → repeats T14.
T16 U file_system.show_directory("/home/ norman","headshot","files",False) → empty; recurse.
T17 U file_system.show_directory("/home/ norman/documents","headshot","files",False) → empty; enable recursion.
T18 D file_system.show_directory("/home/ norman","headshot","files",True) → exact work path.
T19 D file_system.show_file(".../work/ professional_headshot.jpeg") → target file.
T20 D gmail.upload_attachments_to_draft(518,["... /professional_headshot.jpeg"]) → id 8485.
T21 D gmail.show_draft(518) → only headshot remains.
T22 P gmail.show_snoozed_threads("headshot") → empty.
T23 P gmail.show_inbox_threads("headshot") → repeats T1; still unsent.
T24 P file_system.show_directory("/home/ norman/downloads") → unrelated folder; stops before send.
```

TOOLCOMPASS — Success

```
T1 U gmail.show_inbox_threads("manager") → broad results; refine by headshot.
T2 D gmail.show_inbox_threads("headshot") → threads 47821 and 47823.
T3 U gmail.show_thread(47823) → Adrian, not the manager; rule out.
T4 D gmail.show_thread(47821) → Brandon request, email 20867.
T5 U gmail.show_drafts(query="headshot") → empty; change filter.
T6 D gmail.show_drafts(recipient_email= "brandon-webe@gmail.com") → draft 518.
T7 P gmail.show_archived_threads("headshot") → empty.
T8 U gmail.remove_attachment_from_ draft(518,20867) → no such attachment; inspect IDs.
T9 D gmail.show_draft(518) → resume attachment 8483.
T10 D gmail.remove_attachment_from_ draft(518,8483) → deleted.
T11 U file_system.file_exists("/home/ norman/headshot.jpeg") → false; search.
T12 U file_system.show_directory("/home/ norman","headshot","files",False) → empty; descend.
T13 U file_system.show_directory("/home/ norman/documents","headshot","files",False) → empty; recurse.
T14 U file_system.show_directory("/home/ norman/documents","headshot","files",True) → exact work path.
T15 D file_system.show_file(".../work/ professional_headshot.jpeg") → target file.
T16 P file_system.show_directory("/", "resume","files") → irrelevant files.
T17 U gmail.upload_attachments_to_draft(518,[]) → at least one path required.
T18 U gmail.upload_attachments_to_ draft(518,["/home/norman/documents/work"]) → file path, not directory, required.
T19 D gmail.upload_attachments_to_draft(518,["... /professional_headshot.jpeg"]) → id 8485.
T20 P gmail.show_thread(47823) → one redundant distractor recheck.
T21 D gmail.show_draft(518) → only headshot 8485 remains.
T22 D gmail.send_email_from_draft(518) → sent email 20870 to Brandon.
```

Method	Calls	Direct	Useful	Unproductive	Outcome
GRPO	28	9	9	10	Success
MatchTIR	24	10	5	9	Failure
TOOLCOMPASS	22	9	10	3	Success

Figure 18: **Sample 2 (Gmail: replace a draft attachment and send).** All three methods locate the correct thread and draft, but they spend the remaining budget differently. TOOLCOMPASS resolves the path and upload-format errors with targeted follow-ups and finishes in 22 calls with only 3 unproductive ones. GRPO also succeeds but spends 10 of its 28 calls on unrelated mailbox checks and repeats. MatchTIR ends after 24 calls without sending the email.

B.7 FTRL SOLVE-P, SOLVE-R, AND SOLVE-F1 RESULTS

We compare TOOLCOMPASS with LOOP and SEAL on FTRL using Solve-P and Solve-R to distinguish tool-invocation precision from task-completion recall. Table 10 reports Solve-P, Solve-R, and Solve-F1 on the ID, OOD, and entire test sets using Qwen3.5-4B.

Table 10: FTRL results (%) using Qwen3.5-4B. Each cell reports Solve-P / Solve-R / Solve-F1.

Method	ID	OOD	Total
LOOP	37.80 / 57.60 / 41.83	51.90 / 76.40 / 57.72	40.06 / 60.61 / 44.37
SEAL	33.40 / 56.70 / 38.02	53.70 / 73.80 / 57.94	36.65 / 59.44 / 41.21
TOOLCOMPASS	**43.20 / 63.70 / 46.97**	**59.50 / 80.20 / 64.01**	**45.81 / 66.34 / 49.70**

TOOLCOMPASS achieves the best results on all three metrics across the ID, OOD, and total test sets. The gains cover both tool-invocation precision and task-completion recall.

B.8 TOOL-CALL CATEGORIZATION

For the tool-use behavior analysis in Figure 7 and Appendix B.5, GPT-5.6-sol serves as the LLM-as-a-judge (Zheng et al., 2023) and assigns each tool call to one of three categories: *direct tool use*, which performs a required operation or obtains information needed to complete the task; *useful trialing*, a plausible exploratory call that provides new, task-relevant information about tool capabilities, argument requirements, or execution conditions; and *unproductive trialing*, which is unrelated, redundant, or provides no new task-relevant information. The judge receives the task instruction, tool specifications, prior interaction history, current call, and its immediate feedback. The full prompt is shown in Figure 19.

Information novelty is assessed against the supplied tool specifications and prior interaction history. Subsequent actions and final task outcomes are excluded from the judgment.

Tool-Call Categorization Prompt

```
You label the current call using only the task, tool specifications, prior
history, current call, and its immediate feedback. Choose one label:

- DIRECT_TOOL_USE: Directly performs a required operation or obtains
  information needed to complete the task.
- USEFUL_TRIALING: A plausible exploration that provides new, task-relevant
  information about tool capabilities, arguments, or execution conditions.
- UNPRODUCTIVE_TRIALING: An unrelated or redundant call, or an exploration
  that provides no new task-relevant information.

Apply DIRECT_TOOL_USE first. Assess novelty against the tool specifications
and prior history. Errors can be useful if they reveal new relevant information.
Do not use later actions or the final task outcome.

Input:
task: {task_instruction}
tools: {tool_specifications}
history: {prior_interaction_history}
call: {current_call}
feedback: {current_feedback}

Return strict JSON only:
{"turn": <turn_id>, "label": "<ONE_LABEL>", "reason": "<one sentence>"}
```

Figure 19: Prompt used to categorize tool calls with GPT-5.6-sol.

To assess annotation quality, three annotators independently verify a random sample of 200 tool calls with the same information provided to the judge, and we compute Fleiss's kappa over the three sets of labels. The kappa score is 0.73, with a raw agreement rate of 82.00%; the remaining disagreements are resolved by discussion. Moreover, the LLM annotations agree with the final reviewed labels on 86.00% of the calls, supporting their use for the full behavior analysis.

B.9 REPRESENTATION-SPACE VISUALIZATION DETAILS

Figures 2 and 4 use raw layer-$l = 8$ backbone states without projection. Figure 2 applies t-SNE to AppWorld test rollouts; each point represents one call's final-token state. The protocol for Figure 14 applies the same procedure to rollouts from the 92 FTRL semantic OOD queries in Table 7. Figure 4 normalizes states from four function classes onto the three-dimensional unit sphere. Its panels share a viewpoint and jointly normalized color scale; the surface shows the displayed class's spherical kernel density.