ONE TASK, MANY INTERFACES: SIMPLE CROSS-INTERFACE CONSISTENCY TRAINING IMPROVES LLM AGENT ROBUSTNESS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents learn to use tools through a particular interface, and their competence stays tied to it. Semantics-preserving perturbations of that interface, such as renaming an action or adding an equivalent one, leave the underlying task unchanged yet make the agent worse at it. Multi-interface training adds perturbed interfaces to the training data but still scores each training example independently. It therefore broadens this dependence from one interface to the training distribution of interfaces, and perturbations outside that distribution still degrade the agent. We propose **Cross-Interface Policy Regularization (CIPR)**, a self-supervised objective that trains the agent to act consistently across interfaces offering the same tools. We replace action names with neutral identifiers, present each training example under two interfaces whose equivalent actions we know by construction, and penalize the KL divergence between the agent's action distributions under the two. We evaluate CIPR in reinforcement-learning and supervised post-training. On function calling, held-out interface perturbations lower the accuracy of a post-trained agent by over 30 points, and CIPR recovers 15.5 points, more than training on the same perturbed interfaces without the KL penalty, while also raising accuracy on the original interface. In supervised post-training, when the actions of a WebShop agent are renamed, CIPR raises its reward from 5.5 to 76.0[1].

1 INTRODUCTION

Language models are now trained to act as tool-using agents, either natively, as in Qwen3.5 and Kimi K2 (Qwen Team, 2026; Kimi Team, 2025), or by fine-tuning on agent trajectories and reinforcement learning from task rewards (Zeng et al., 2024; Qian et al., 2026). What these agents learn, however, is tied to the interface through which their tools are presented, meaning the names, descriptions, and argument schemas of the available tools. Figure 1 illustrates this with a weather agent trained on a single interface, which fails when its forecast tool is renamed and when a cheaper equivalent is added, although neither change alters the task. Agents fine-tuned on trajectories can lose most of their task success when action names are minimally rewritten (Gu et al., 2026), and function-calling models degrade when tools are renamed, similar tools are added, or requests are rephrased (Ye et al., 2024; Rabinovich & Tavor, 2025; Zhou et al., 2026b). Such changes are routine in deployment, where registries combine tools from different servers, providers revise their schemas, and users misspell requests (Zhou et al., 2026b). Interfaces can also be manipulated, since edits to a tool's description alone can make agents prefer it over equivalent tools (Faghih et al., 2025).

This failure is an instance of shortcut learning (Geirhos et al., 2020). On a fixed training interface, a policy that reads the tool descriptions and a policy that relies on a familiar name or menu position make the same calls and earn the same reward, so the task objective has no reason to prefer the first. The second policy fails as soon as the interface changes, even if the task is unchanged. Such a policy can still succeed on new tasks under its training interface, so the failure is one of generalizing across interfaces rather than of fitting noise in the training examples.

Prior training methods reduce this sensitivity by changing what the agent trains on. Function masking replaces tool and argument names with random strings (Lin et al., 2025), which removes one cue

[1]Anonymous code repository: https://anonymous.4open.science/r/QN1GAYENVL/.

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.

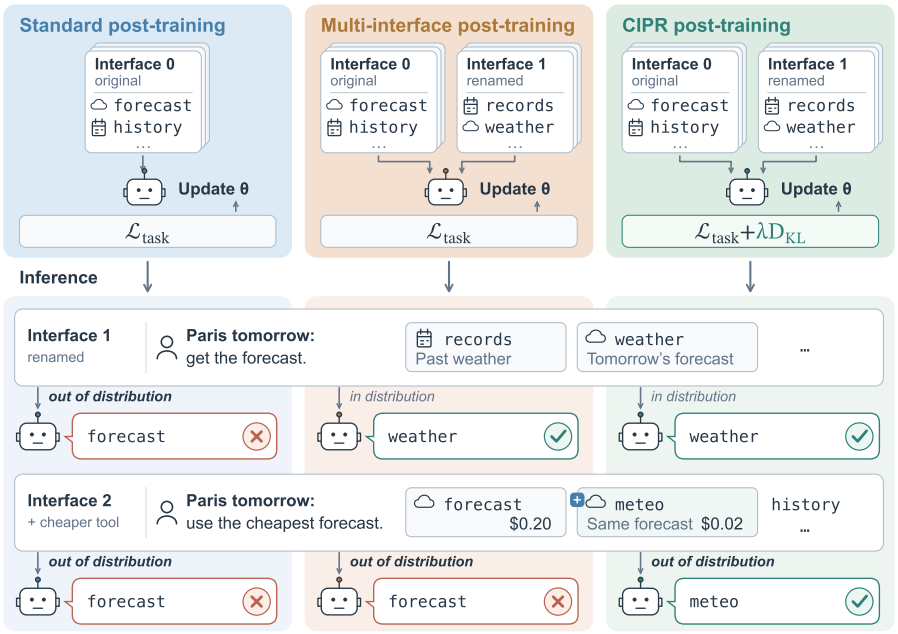

Figure 1: **Interface robustness of a tool-using agent** (schematic). In each column, we post-train an agent to fetch weather forecasts (top) and then test it at inference (bottom) on a renamed interface (*Interface 1*) and on one with an added, cheaper forecast tool (*Interface 2*). The label on each arrow into an agent says whether the test interface is in that agent's training distribution. Standard post-training on one interface fails both. Multi-interface training handles the renaming, which it saw during training, but misses the new tool. CIPR, which also compares the policy across interfaces, chooses by description in both cases.

but leaves others, such as a tool's position or the set of tools offered. Multi-interface training, also known as domain randomization, varies tool schemas, metadata, and question formats during training (Zhou et al., 2026b; Liu et al., 2026), and synthetic corpora such as ToolACE broaden the tools seen in training (Liu et al., 2025). Both approaches still score each training interface on its own. Their objective asks the policy to succeed under every interface it sees, but never to treat equivalent interfaces alike, so a policy can fit every training variant while keeping a shortcut that fails on a new kind of change, as in the middle column of Figure 1.

The interface changes an agent meets in deployment cannot all be anticipated, so robustness should not depend on training on each of them. Instead, the agent should treat interfaces that offer equivalent tools alike, including interfaces it has never seen. We propose *Cross-Interface Policy Regularization* (CIPR), which trains for this property directly (Figure 2). CIPR uses a correspondence that multi-interface training discards, since whenever we construct a perturbed interface, we know which of its tools is equivalent to which tool in the original. We canonicalize each training interface, replacing every tool name with a neutral identifier such as `function_01`, so that names carry no task information. Canonicalization leaves other cues in place, such as a tool's position, so we pair the canonical interface with a perturbed copy in which the tools are renamed, reordered, and joined by degraded twins. We then penalize the KL divergence between the agent's action distributions under the two interfaces, with the canonical interface as the reference. Unlike prior consistency regularization, which asks a model for the same output under both inputs (Xie et al., 2020; Zhou et al., 2026a), this term lets the response change with the tool names and compares only the distributions over actions, with each tool matched to its counterpart through the known correspondence. The consistency KL has two terms. The *rescoring consistency KL* holds a response fixed and compares the action distributions along it, which needs no additional generation, and the *rollout consistency KL* lets each interface produce its own responses, since a different interface can call for different reasoning. We add both terms to the task loss, and they need no labels beyond those of the task.

We evaluate CIPR in reinforcement-learning post-training on BFCL (Patil et al., 2025), under thirteen held-out perturbation types adapted from RobustBench-TC (Zhou et al., 2026b), with harder implicit versions of the cost and latency types, and in supervised post-training on WebShop (Yao et al., 2022). On two backbones, CIPR is more robust than the base model and standard post-training, and it improves on multi-interface training in every held-out condition while keeping original accuracy.

We make three contributions.

- CIPR, a self-supervised consistency KL between an agent's action distributions under two interfaces whose equivalent actions are known, with two terms that trade compute for coverage of the agent's reasoning (Section 3).
- A tool-calling instantiation that pairs canonicalization with a perturbation in which only descriptions and schemas identify the right tool (Section 3.2).
- Experiments in both post-training regimes (Section 4), in which CIPR improves on standard post-training and multi-interface training under held-out perturbations on BFCL and makes AgentLM robust to renamed actions on WebShop.

2 Related Work

Our work relates to studies of how tool-using agents respond to interface changes, to methods that train agents on varied interfaces, and to consistency regularization across equivalent inputs.

Interface sensitivity of tool-using agents. Tool-use benchmarks mostly score agents on the interface they were released with (Patil et al., 2025; Barres et al., 2026). A growing line of work perturbs the interface. RoTBench corrupts tool and argument names at graded noise levels (Ye et al., 2024), Rabinovich & Tavor (2025) rephrase queries and add related tools on BFCL, and small edits to tool descriptions change which tool an agent selects (Faghih et al., 2025; Blankenstein et al., 2026). PIPE rewrites environment interfaces while preserving task semantics and finds that fine-tuning on trajectories increases reliance on the training interface (Gu et al., 2026). RobustBench-TC organizes perturbations into categories grounded in production failures (Zhou et al., 2026b), and we use its taxonomy for evaluation. These studies measure the problem, whereas we address it during training.

Training on varied interfaces. Most remedies applied during training change what the agent sees. Hammer masks function and argument names during supervised fine-tuning (Lin et al., 2025), and ToolRL includes masked examples in its reinforcement-learning data (Qian et al., 2026). ToolACE synthesizes function-calling data over a pool of 26,507 diverse APIs (Liu et al., 2025), ToolRL-DR applies domain randomization to tool-use reinforcement learning (Zhou et al., 2026b), perturbation-augmented fine-tuning trains on perturbed trajectories and their corrections (Lv et al., 2026), and NoisyAgent trains on clean and noisy rollouts of the same tasks (Chen et al., 2026). Outside tool use, multi-format supervision makes models less sensitive to how a question is posed (Liu et al., 2026). These methods broaden the training distribution, but their objective still scores each training example on its own. We keep the perturbed interfaces and add a term that compares the policy across them.

Consistency across equivalent inputs. Consistency regularization penalizes the divergence between a model's predictions on two versions of an input that should receive the same prediction. In semi-supervised learning the two versions are an unlabeled example and its augmentation (Xie et al., 2020), and the same idea has been applied to reinforcement-learning policies on augmented observations (Raileanu et al., 2021) and to vision-language-action policies under perturbed observations (Luo et al., 2026). In concurrent work, PIRL applies it to reinforcement learning of multimodal LLMs with a per-token KL between the policy's next-token distributions for the same response under a clean and an adversarially perturbed prompt (Zhou et al., 2026a). Each of these methods compares the two predictions directly, which assumes the correct output is the same under both inputs. Renaming tools breaks this assumption, because the correct call is then written with different tokens, so we compare distributions over actions matched through the known correspondence between the two interfaces.

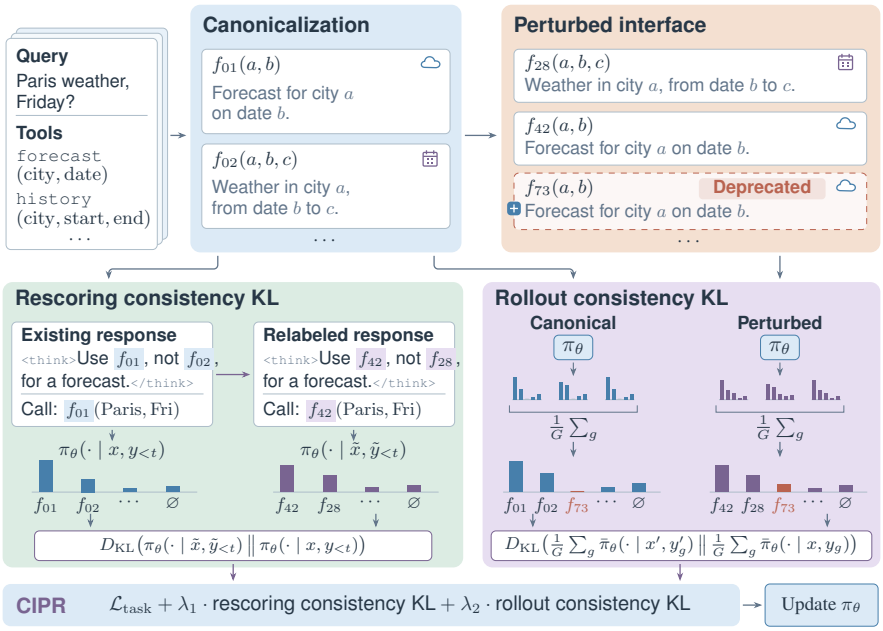

Figure 2: **Overview of CIPR.** We first canonicalize the interface, renaming the two tools of a weather query f_{01} and f_{02}, short for `function_01` and `function_02`. We then build a perturbed interface that adds a degraded twin of every tool, such as the deprecated forecast f_{73}, and assigns all tools new identifiers in a new order. Each bar chart is an action distribution, the probability of calling each tool or none (∅). The rescoring consistency KL holds one response fixed and compares these distributions at each of its decisions, before and after its tool names are relabeled in the reasoning and the call. The rollout consistency KL lets each interface produce its own responses, so the reasoning can differ, and compares their average distributions, in which the twin f_{73} has probability zero under the canonical interface. Probabilities are illustrative.

3 Cross-Interface Policy Regularization

We consider an LLM agent that receives a user query together with an interface, which lists the available tools with their names, descriptions, and argument schemas. At each step the agent either calls one of the listed tools or responds without a call. A perturbation of the interface is semantics-preserving when the tool that solves the task is still available and still solves it, even though its name, its position, or the surrounding options have changed. We want the agent to assign the same probability to equivalent tools under such perturbations, and CIPR trains for this in three steps. We canonicalize the interface so that tool names carry no task information (Section 3.1), pair every training example with a perturbed copy of its interface (Section 3.2), and penalize the divergence between the agent's action distributions under the two interfaces (Section 3.3). Figure 2 summarizes the procedure.

3.1 Canonicalization

The most direct shortcut available to a trained agent is a familiar tool name. Because a name that led to correct calls during training usually still does so at test time, a policy can select tools by name without reading their descriptions, and function-calling models are known to be misled by naming conventions (Lin et al., 2025). We remove this shortcut by canonicalizing the interface, replacing every tool name with a neutral identifier and every argument name with an identifier numbered within its tool, so that `forecast(city, date)` becomes `function_01(arg_01, arg_02)`. We keep descriptions, types, required arguments, and argument values unchanged, so the agent has to read the descriptions to tell the tools apart, and we apply the same mapping to every reference to these names in the query, in earlier turns, and in the training targets. At inference, a canonicalization adapter applies this mapping to the incoming interface and translates the generated call back to the

executable API, so the deployed tools themselves are never modified. Canonicalization is related to function masking (Lin et al., 2025), a training strategy that replaces names with random strings so that the model attends to descriptions; unlike masking, we also apply the mapping at inference, so the policy never meets the original names.

3.2 Perturbed interfaces

The consistency KL of Section 3.3 needs only a perturbed interface in which we know which tool is equivalent to which canonical tool. We design ours to also remove the cues that canonicalization leaves in place, such as a tool's position in the menu or the identifiers seen during training, and construct it for every training example from the canonical interface in three steps. We first give every tool a degraded twin (**twin expansion**), a copy whose description states that it is worse, for example deprecated or slower, or whose schema replaces an argument the task needs with an unrelated one. We then give the expanded menu new identifiers (**relabeling**) and shuffle the tools and their arguments (**reordering**). The query and the descriptions of the original tools are unchanged, and each twin differs from its original only in its description or schema. Appendix B lists the exact operations.

3.3 Consistency KL on equivalent actions

The task objective never compares which tools the policy chooses under the canonical and the perturbed interface. We add this comparison by building on consistency regularization (Xie et al., 2020; Raileanu et al., 2021), which penalizes the divergence between a model's predictions on two versions of an input that call for the same output. An agent's correct response, however, changes with the interface, because the tools it calls have different names, and what stays fixed is the correspondence between equivalent tools, which is known from the construction of the perturbed interface. We therefore compare the policy's distributions over actions rather than over tokens.

Action distributions. The policy π_θ generates a response y to a prompt x, which contains the interface, the query, and any earlier turns. Within y, the policy makes a decision each time it begins a tool call or begins the text that ends its turn, and each decision is an action a, either a tool declared in x or $\varnothing$, answering without a call. Writing $y_{<t}$ for the part of y before its t-th decision, we denote by $\pi_\theta(a \mid x, y_{<t})$ the probability that the policy continues $y_{<t}$ with action a. We read this probability in one forward pass from the probability of the tokens that open a call and of each declared tool's name after them, normalized over the declared tools (Appendix A), as in work that reads the action probabilities of LLM agents from their token probabilities (Carta et al., 2023; Tan et al., 2024). The two interfaces name the same tool differently, so we index actions by equivalence rather than by name or position, and a tool and its counterpart under the other interface share an index. A tool that only one interface declares, such as a twin, has an index of its own, with probability zero under the interface that does not declare it, so probability moved onto such a tool counts as disagreement instead of being merged with another. Arguments do not enter this distribution and are trained by the task objective alone.

Two consistency terms. We compare these distributions under the two interfaces in two ways, which differ in whether the agent's reasoning is held fixed (Figure 2).

- **Rescoring consistency KL.** When only the tool names change, the reasoning behind a response remains valid under both interfaces, so this term holds a response fixed and compares the decisions along it. We take a response y to the canonical prompt x, a rollout of the task objective or a demonstration in supervised post-training, and relabel it, giving every tool a new neutral name in the prompt and throughout the response. Each decision of the relabeled response $\tilde{y}$ to the relabeled prompt $\tilde{x}$ sits at the same position as in y, and we average the divergence over the n_y decisions of y,

$$\mathrm{KL}_{\mathrm{rescore}}(x, y) = \frac{1}{n_y} \sum_{t=1}^{n_y} D_{\mathrm{KL}}\big(\pi_\theta(\cdot \mid \tilde{x}, \tilde{y}_{<t}) \,\big\|\, \mathrm{sg}[\pi_\theta(\cdot \mid x, y_{<t})]\big). \tag{1}$$

  Scoring $\tilde{y}$ needs only forward passes and no generation.
- **Rollout consistency KL.** When the interface changes which tools are available, the agent should reason differently, and comparing its decisions along a fixed response would penalize

exactly that. This term therefore lets each interface produce its own responses. We sample G responses $y_1, \ldots, y_G$ to x and G responses $y'_1, \ldots, y'_G$ to x', the prompt with the full perturbed interface of Section 3.2, and compare the average action distributions of the two groups,

$$\mathrm{KL}_{\mathrm{rollout}}(x) = D_{\mathrm{KL}}\Big(\frac{1}{G}\sum_{g=1}^{G} \bar{\pi}_\theta(\cdot \mid x', y'_g) \,\Big\|\, \mathrm{sg}\Big[\frac{1}{G}\sum_{g=1}^{G} \bar{\pi}_\theta(\cdot \mid x, y_g)\Big]\Big), \tag{2}$$

where $\bar{\pi}_\theta(a \mid x, y) = \frac{1}{n_y}\sum_{t=1}^{n_y} \pi_\theta(a \mid x, y_{<t})$ averages the action distribution over the decisions of one response. Only the responses are sampled, so each rollout contributes its whole action distribution at every decision rather than only the action it took. Gradients flow through these probabilities but not through the sampling. Because the two groups are sampled independently, their responses do not correspond, so we compare only their averages, at the cost of G additional rollouts per prompt.

Objective. We add both terms to the task loss of Section 3.4,

$$\mathcal{L}(\theta) = \mathcal{L}_{\mathrm{task}}(\theta) + \lambda_1 \,\mathbb{E}_{x,y}\big[\mathrm{KL}_{\mathrm{rescore}}(x, y)\big] + \lambda_2 \,\mathbb{E}_x\big[\mathrm{KL}_{\mathrm{rollout}}(x)\big], \tag{3}$$

where x ranges over training prompts and y over the responses that the task loss uses, $D_{\mathrm{KL}}(p \,\|\, q) = \sum_a p(a) \log \frac{p(a)}{q(a)}$ sums over the tools and $\varnothing$, sg stops the gradient, and $\lambda_1, \lambda_2 \geq 0$. In both terms the reference is the policy itself under the canonical interface, so no teacher model or additional labels are needed. We stop its gradient so that the perturbed distribution moves toward it rather than both moving toward each other. A tool that only the perturbed interface declares has reference probability zero, so we floor reference probabilities at a small constant ϵ inside the logarithm, which keeps the divergence finite and makes the penalty for such a tool grow with its probability under the perturbed interface.

3.4 Task loss and inference

The consistency terms do not depend on how $\mathcal{L}_{\mathrm{task}}$ is computed, and we use two task losses. In reinforcement learning, we use the GRPO objective of ToolRL (Qian et al., 2026), computed on rollouts under both interfaces. In supervised post-training, we use a distillation loss and only the rescoring consistency KL (Section 4.3). We add the consistency terms to the loss rather than to the reward, so agreement cannot substitute for correct calls and the task loss remains responsible for solving the task. At inference, we run the model with the canonicalization adapter alone; perturbed interfaces, relabeled copies, and paired rollouts exist only during training. Algorithm 1 in Appendix A gives the full procedure.

Nothing in the consistency KL is specific to tools; it requires only actions with model probabilities and a perturbation under which equivalent actions are known.

4 Experiments

We evaluate CIPR in both regimes of agent post-training. If the consistency KL removes interface shortcuts rather than fitting the training perturbations, CIPR should raise accuracy under held-out perturbations, keep or raise accuracy on the original interface, and improve on multi-interface training, which sees the same perturbed interfaces without the consistency KL. We test these three predictions by adding CIPR to reinforcement learning for function calling (Section 4.2); we then apply it to supervised post-training on WebShop (Section 4.3) and ablate its components (Section 4.4).

4.1 Setup

We build all reinforcement-learning experiments on the public ToolRL recipe (Qian et al., 2026), so the compared models differ only in their training interfaces and objective.

Training. We train with GRPO (Shao et al., 2024) on the 3,920 ToolRL training examples, rewarding correct calls and format. We use two backbones, Qwen2.5-3B-Instruct (Qwen Team, 2024) with tools in the prompt and Qwen3.5-4B (Qwen Team, 2026) with thinking and native function calling. Appendix A lists hyperparameters.

Table 1: **Full-call accuracy (%) on BFCL under interface perturbations.** *Tool names* gives the names each model sees at evaluation. Each model is evaluated with the names it was trained on, as it would be deployed, so Base and ToolRL (GRPO) see the original names, and the multi-interface and CIPR models, which are trained with canonical names, see canonical names through the canonicalization adapter. Gray rows evaluate Base and ToolRL (GRPO) through the canonicalization adapter as well. *In distribution* applies the perturbation used in training (Section 3.2), which is defined on canonical names, so rows with the original names show n/a. *Mean* averages the three held-out conditions, and *Gap* is original accuracy minus the mean. Bold marks the best value per backbone, excluding gray rows. All 95% confidence intervals have half-widths of at most 2.0 points, and Appendix D reports each perturbation type.

			In distribution ↑	Held-out perturbations ↑				
Model	Tool names	Original ↑	Twins & reordering	Typos & distractors	Cost & lat. (explicit)	Cost & lat. (implicit)	Mean	Gap ↓
Qwen2.5-3B, tools in prompt								
Base	original	63.7	n/a	59.9	24.2	17.7	33.9	29.8
	canonical	61.5	43.5	59.3	41.9	28.5	43.2	18.3
ToolRL (GRPO)	original	73.3	n/a	71.6	29.3	20.9	40.6	32.7
	canonical	70.7	49.1	68.9	46.0	33.5	49.5	21.2
+ multi-interface	canonical	74.0	68.2	71.5	51.5	39.6	54.2	19.8
+ CIPR (ours)	canonical	**74.9**	**68.8**	**72.2**	**53.6**	**42.3**	**56.1**	**18.8**
Qwen3.5-4B, native function calling								
Base	original	84.2	n/a	83.4	83.1	55.3	73.9	10.2
	canonical	82.0	76.3	81.3	81.0	58.8	73.7	8.3
ToolRL (GRPO)	original	84.8	n/a	84.2	**84.8**	60.2	76.4	8.4
	canonical	84.6	77.1	84.0	84.0	52.3	73.4	11.1
+ multi-interface	canonical	84.8	**84.8**	84.1	84.2	70.7	79.7	5.1
+ CIPR (ours)	canonical	**85.0**	83.8	**84.5**	84.6	**73.0**	**80.7**	**4.2**

Baselines. We compare four models on each backbone. *Base* is the backbone without post-training, and *ToolRL (GRPO)*, our standard post-training baseline, is the backbone post-trained with the ToolRL recipe; both are evaluated with the original tool names, as they would be deployed. Multi-interface training adds the perturbed interfaces of Section 3.2 to ToolRL training, as in ToolRL-DR (Zhou et al., 2026b), and CIPR adds both consistency terms to it; the tables label these models + multi-interface and + CIPR. We train both with canonical names on the same interfaces and evaluate both through the canonicalization adapter, so that they differ only in whether the policy is compared across them.

Held-out perturbations. We evaluate on the 2,501 single-turn items of BFCL v4 (Patil et al., 2025) that have a reference call, under thirteen perturbation types adapted from RobustBench-TC (Zhou et al., 2026b) (Appendix C). None of them is used during training, and we report them in three conditions. *Typos & distractor tools* introduces typos into the query or adds a distractor for every tool the task calls, either an unrelated tool renamed to resemble it or a duplicate with the same name whose description or parameters are corrupted. The six cost and latency types add a cheaper or faster equivalent of the correct tool under a new name, so that the familiar name now points to the worse choice. As in RobustBench-TC, the query in *Cost & latency (explicit)* asks for the better tool, for example "Please use a cost-effective option". This request tells the agent what to prefer, so we add a harder version, *Cost & latency (implicit)*, in which the query only asks the agent to describe the available tools and states no preference. These two conditions are the hardest test, because the right choice depends only on a stated cost or latency, and preferring a cheaper or faster tool is never rewarded during training. Beyond single-turn calls, we evaluate the three conditions on the airline and retail domains of τ^2-bench (Barres et al., 2026), where the agent completes multi-turn tasks with a simulated user, and Appendix F also reports RoTBench (Ye et al., 2024), which corrupts tool and parameter names at graded levels.

4.2 Robustness to held-out perturbations

Table 1 tests the three predictions on BFCL.

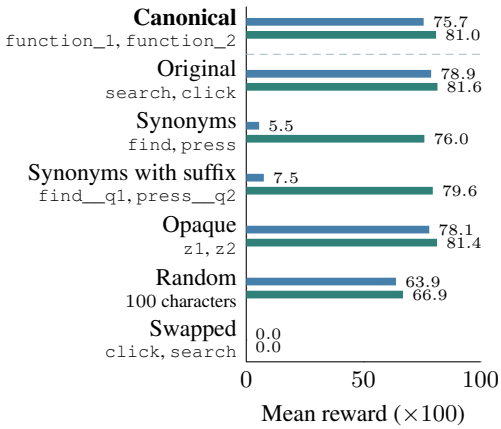

Figure 3: **WebShop** evaluation on AgentLM-7B before (blue) and after (green) CIPR self-distillation, under different names.

Table 2: **Ablations on Qwen2.5-3B** (accuracy, %, canonical names). Each row removes one component from CIPR, and removing the consistency KL gives multi-interface training; †one added near-duplicate tool takes the place of the twins. The gray row is ToolRL (GRPO) with canonical names.

	Original ↑	Cost & latency (explicit) ↑
+ CIPR	74.9	53.6
− consistency KL	73.5	51.7
− twins	73.5	48.1
− twins and reordering†	74.7	43.7
ToolRL (GRPO)	70.7	46.0

Comparison with ToolRL (GRPO). Standard post-training leaves most of the gap in place. On Qwen2.5-3B, ToolRL (GRPO) raises original accuracy by 9.6 points, yet its gap grows from 29.8 to 32.7 points, because the familiar name it learned to follow now points to the worse tool. On Qwen3.5-4B, whose base model is already trained for tool use, the gap narrows only from 10.2 to 8.4 points.

Canonical names alone. Canonical names are not a free improvement for a model trained without them (gray rows). They lower the original accuracy of both base models and of ToolRL (GRPO), and although they raise the cost and latency accuracy of ToolRL (GRPO) on Qwen2.5-3B, they lower its accuracy in every held-out condition on Qwen3.5-4B.

Comparison with multi-interface training. Training on perturbed interfaces makes canonical names pay off, and the consistency KL adds to that gain. On both backbones, CIPR is higher than multi-interface training on the original interface and in every held-out condition. The largest gains are on the cost and latency perturbations, the ones least like anything seen in training, with 2.1 and 2.7 points on Qwen2.5-3B and 2.3 points on the implicit version for Qwen3.5-4B, while the other differences are within the intervals. Under the perturbation used in training, the two models differ by about a point or less on both backbones, so the advantage of CIPR lies in the perturbations that training did not cover. On both backbones, CIPR thus matches or exceeds the original accuracy of ToolRL (GRPO) while raising its mean accuracy under perturbation by 15.5 and 4.3 points.

Interactive tasks. On τ^2-bench, the Qwen3.5-4B CIPR model has the smallest gap in both domains between its original reward and its mean reward under the three conditions, although ToolRL (GRPO) with the original names has a higher reward on the original interface and under every condition (Appendix D).

4.3 Supervised post-training

We next apply CIPR to an agent trained by supervised learning, AgentLM-7B (Zeng et al., 2024), whose WebShop policy collapses when its actions are renamed (Gu et al., 2026).

Setting. AgentLM is fine-tuned on agent trajectories mixed with general instruction data, and acts on WebShop (Yao et al., 2022) through two actions named `search` and `click`. The released model already solves the task under its original action names, so we apply CIPR as self-distillation. The released model is the teacher and initializes the student, which we train to match it under canonical names and, through the rescoring consistency KL, under relabeled names, without additional rollouts.

Results. Consistent with Gu et al. (2026), renaming `search` and `click` to the synonyms `find` and `press` reduces the released model's reward from 78.9 to 5.5 (Figure 3). After self-distillation, the agent reaches 81.0 under canonical names, the interface it is deployed with, against 75.7 for the released model behind the same adapter, so the adapter alone does not explain the gain. Seeing the names directly, the trained agent is also robust to neutral renaming, with 76.0 under the synonyms. Swapped names, each of which suggests the other action, drive both models to zero, which is why CIPR keeps the canonicalization adapter, behind which a misleading name never reaches the policy.

4.4 Ablations

Table 2 ablates the consistency KL, the twins, and the reordering of the perturbed interface on Qwen2.5-3B, with every model evaluated under canonical names (Appendix D lists every run). It reports accuracy on the original interface and on the explicit cost and latency perturbations, where only the stated cost or latency identifies the right tool.

Effect of the consistency KL. Without the consistency KL, the method reduces to multi-interface training on the same interfaces, and accuracy falls from 53.6% to 51.7% on the explicit cost and latency perturbations and from 74.9% to 73.5% on the original interface, consistent with the difference between the two methods in Table 1.

Effect of the perturbed interface. The twins contribute the most. Without them, accuracy on the explicit cost and latency perturbations falls from 53.6% to 48.1%, and with a single added near-duplicate tool in place of twins and reordering it falls to 43.7%, below ToolRL (GRPO) with canonical names. Trained with a single added tool, the model relies on position rather than on the stated costs, with 70.8% when the correct tool comes first and 16.5% when the costlier equivalent does (Appendix D). The consistency KL also gains more from twins, adding 1.9 points with them against 0.6 with reordering alone.

Effect of canonicalization at inference. Canonical names are needed at inference as well. Evaluated with the original names, the CIPR model reaches 28.7% on the explicit cost and latency perturbations in the original menu order, against 60.5% with canonical names, because the familiar name belongs to the costlier tool.

5 Conclusion

LLM agents learn to use tools through a particular interface, and the task objective cannot tell a decision that depends on the task from one that depends on how the tools are presented. We supply that distinction with CIPR, a self-supervised consistency KL between the agent's action distributions under two interfaces whose equivalent actions are known by construction. Added to multi-interface training, it improves accuracy under every held-out condition of BFCL on both backbones without lowering accuracy on the original interface. Applied as self-distillation, it makes AgentLM robust to renaming its WebShop actions. Because the targets come from the policy itself, the objective needs no annotation beyond the task, which makes it a natural candidate for agentic training at scale.

Limitations. CIPR needs the canonicalization adapter at inference, because a name that misrepresents its action, as in the swapped WebShop condition, still misleads a trained agent that sees it directly. The consistency KL constrains which tool the agent calls and whether it calls one, while arguments are learned from the task loss alone, and argument errors barely change in our error analysis (Appendix E). It also requires a perturbation whose equivalences are known, which is direct for tool menus but harder to construct for richer actions such as edits to a program. The rollout consistency KL samples an additional group of rollouts per prompt, which raises the cost of training. Our experiments use models of up to 9B parameters, one training run per configuration, and mostly single-turn function calling, so the effect at larger scale and in long multi-turn tasks remains to be established.

AI USE STATEMENT

In this work, we used generative AI tools for implementing the method and the pipelines that build its data. We have not used generative AI tools for generating synthetic data or designing experiments, and mathematical proofs and translation are not applicable to this work. Additionally, we used generative AI tools for drafting the paper. We have reviewed all AI-assisted work, including the code, and the draft served as an outline that we substantially rewrote and edited. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. In *International Conference on Machine Learning*, 2026.

Thierry Blankenstein, Jialin Yu, Zixuan Li, Vassilis Plachouras, Sunando Sengupta, Philip Torr, Yarin Gal, Alasdair Paren, and Adel Bibi. Biasbusters: Uncovering and mitigating tool selection bias in large language models. In *International Conference on Learning Representations*, volume 2026, pp. 102460–102489, 2026.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 3676–3713, 2023.

Yuxin Chen, Xiaodong Cai, Junfeng Fang, Zhuowen Han, Yu Wang, Yaorui Shi, Yi Zhang, Qi Gu, Xunliang Cai, Xiang Wang, An Zhang, and Tat-Seng Chua. Learning to act under noise: Enhancing agent robustness via noisy environments, 2026.

Kazem Faghih, Wenxiao Wang, Yize Cheng, Siddhant Bharti, Gaurang Sriramanan, Sriram Balasubramanian, Parsa Hosseini, and Soheil Feizi. Tool preferences in agentic llms are unreliable. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20965–20980, 2025.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020. doi: 10.1038/s42256-020-00257-z.

Weizheng Gu, Chengze Li, Zhuohao Yu, Mengyuan Sun, Zhibang Yang, Wei Wang, Hongrui Jia, Shikun Zhang, and Wei Ye. What do agents learn from trajectory-SFT: Semantics or interfaces? In *International Conference on Machine Learning*, 2026.

Kimi Team. Kimi K2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, et al. Robust function-calling for on-device language model via function masking. In *The Thirteenth International Conference on Learning Representations*, 2025.

June M. Liu, Shaomian Zheng, He Cao, Dingnan Jin, Qing Cui, and Jun Zhou. Improving cross-format robustness in language models with multi-format training, 2026.

Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, et al. Toolace: Winning the points of llm function calling. In *International conference on learning representations*, volume 2025, pp. 41359–41381, 2025.

Jingzhou Luo, Yifan Wen, Yongjie Bai, Xinshuai Song, Yang Liu, and Liang Lin. RoVLA: Multi-consistency constraints for robust vision-language-action models, 2026.

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. In *International Conference on Machine Learning*, 2026.

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

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, pp. 105523–105553, 2026.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

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

Ella Rabinovich and Ateret Anaby Tavor. On the robustness of agentic function calling. In *Proceedings of the 5th Workshop on Trustworthy NLP (TrustNLP 2025)*, pp. 298–304, 2025.

Roberta Raileanu, Maxwell Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. Automatic data augmentation for generalization in reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 5402–5415, 2021.

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

Weihao Tan, Wentao Zhang, Shanqi Liu, Longtao Zheng, Xinrun Wang, and Bo An. True knowledge comes from practice: Aligning large language models with embodied environments via reinforcement learning. In *International Conference on Learning Representations*, 2024.

Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. Unsupervised data augmentation for consistency training. In *Advances in neural information processing systems*, volume 33, pp. 6256–6268, 2020.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20744–20757, 2022.

Junjie Ye, Yilong Wu, Songyang Gao, Caishuang Huang, Sixian Li, Guanyu Li, Xiaoran Fan, Qi Zhang, Tao Gui, and Xuan-Jing Huang. Rotbench: A multi-level benchmark for evaluating the robustness of large language models in tool learning. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 313–333, 2024.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, 2024.

Pengfei Zhou, Zhiwei Tang, Xiaopeng Peng, Chenrui Zhou, Lama Moukheiber, Yixing Ma, Bin Xu, Jiajun Song, Zhenglin Wan, Wangbo Zhao, Jiasheng Tang, Bohan Zhuang, Fan Wang, and Yang You. Improving generalization robustness of multimodal RLVR, 2026a.

Xiaolin Zhou, Aojie Yuan, Zheng Luo, Zipeng Ling, Xixiao Pan, Yicheng Gao, Haiyue Zhang, Jiate Li, Shuli Jiang, Prince Zizhuang Wang, Zixuan Zhu, Jinbo Liu, Ryan A. Rossi, Hua Wei, and Xiyang Hu. When simulation lies: A sim-to-real benchmark and domain-randomized RL recipe for tool-use agents, 2026b.

A Training details

This appendix gives the hyperparameters of the reinforcement-learning runs (Table 3) and the full CIPR training procedure (Algorithm 1).

Hyperparameters. All reinforcement-learning runs build on ToolRL (Qian et al., 2026) and keep its optimizer settings. The task objective is GRPO (Shao et al., 2024) with rewards for call correctness and response format and no reference-policy KL; calling a degraded twin earns the same reward as any other wrong call.

Table 3: Hyperparameters of the reinforcement-learning runs.

	Qwen2.5-3B	Qwen3.5-4B
Training examples	3,920	3,920
Learning rate	10^{-6}	10^{-6}
Prompts per update	512	256
Optimization minibatch	128	128
Training budget	15 epochs (about 105 updates)	15 epochs (about 105 updates)
Rollouts per interface (G)	4	4
Rescoring consistency KL coefficient (λ_1)	0.02	0.02
Rollout consistency KL coefficient (λ_2)	0.005	0.005
Floor on reference probabilities (ϵ)	10^{-12}	10^{-12}

Training procedure. Algorithm 1 summarizes CIPR training for tool calling. In self-distillation (Section 4.3), the task loss is the KL from the released model under original names to the student under canonical names, and we use only the rescoring consistency KL and sample no rollouts.

Computing the action distribution. We read the probabilities $\pi_\theta(a \mid x, y_{<t})$ of Section 3.3 from the token probabilities at each decision in one forward pass. Let o be the tokens that open a call and precede the tool name, for example `<tool_call>` and the formatting before the name, and let n_a be the name of tool a. Then $\pi_\theta(a \mid x, y_{<t}) = \pi_\theta(o \mid x, y_{<t})\, \pi_\theta(n_a \mid x, y_{<t}, o) / \sum_b \pi_\theta(n_b \mid x, y_{<t}, o)$ for a declared tool a, where b ranges over the declared tools, and $\pi_\theta(\varnothing \mid x, y_{<t}) = 1 - \pi_\theta(o \mid x, y_{<t})$. A name that spans several tokens is scored token by token, each conditioned on the tokens before it.

Algorithm 1: CIPR training for tool calling.

Input: policy π_θ, training prompts $\mathcal{D}$, group size G, coefficients λ_1 and λ_2
foreach *minibatch* $\mathcal{B} \subset \mathcal{D}$ **do**
  foreach *prompt in* $\mathcal{B}$ **do**
    $x \leftarrow$ canonicalization of the prompt ; `// Section 3.1`
    $x' \leftarrow$ perturbed copy of x ; `// Table 4`
    Sample $y_1, \ldots, y_G \sim \pi_\theta(\cdot \mid x)$ and $y'_1, \ldots, y'_G \sim \pi_\theta(\cdot \mid x')$ and score them;
    Relabel a canonical response and rescore its decisions ; `// rescoring consistency KL`
    Average the action distributions of each group ; `// Eq. (2)`
  Take a gradient step on $\mathcal{L}(\theta)$ of Eq. (3), with the reference distributions detached;

B Perturbed training interfaces

Table 4 lists the operations that build the perturbed training interface from the canonical one. The rescoring consistency KL uses the relabeling alone, without twins (Section 3.3); the rollout consistency KL and the task loss use the full construction. The in-distribution evaluation of Table 1 applies the full construction once to every BFCL item, with a fixed seed.

Table 4: Construction of the perturbed training interface. The query is unchanged, and f_{01} abbreviates `function_01`. These operations build the training pairs and the in-distribution evaluation of Table 1; Table 5 lists the held-out perturbations.

Operation	Change	Example
Twin expansion	Copy every tool and degrade the copy's description or required arguments. The original stays available.	A forecast tool gains a copy marked deprecated; a date-range query loses its end date in favor of an unrelated width argument.
Relabeling	Assign new neutral identifiers to the expanded menu and update every reference to them.	f_{01} becomes f_{42}, and its twin becomes f_{73}.
Tool reordering	Shuffle the tools after expansion, regardless of whether a tool is an original or a twin.	Twins and originals do not occupy fixed positions.
Argument reordering	Shuffle argument declarations while keeping their names, types, descriptions, and requiredness.	(a, b, c) becomes (c, a, b) with the same values bound.

C Evaluation details

This appendix describes the evaluation items, the held-out perturbation types, the action surfaces and decoding, and the scoring procedure used in Section 4.

Item set. We evaluate on the 2,501 single-turn items of BFCL v4 (Patil et al., 2025) that have a reference call, across its ten partitions, namely `live_multiple` (1,053), `simple_python` (400), `live_simple` (258), `multiple`, `parallel`, and `parallel_multiple` (200 each), `simple_java` (100), `simple_javascript` (50), `live_parallel_multiple` (24), and `live_parallel` (16). We apply every perturbation to the same items, so comparisons between conditions are paired by item.

Perturbation types. Table 5 lists the thirteen held-out perturbation types, adapted from RobustBench-TC (Zhou et al., 2026b), and the condition in which we report each. The six cost and latency types share one construction. The menu contains the correct tool and an equivalent with the same description but a higher stated cost or latency; the equivalent keeps the original tool's name and the correct tool receives a new one, so that the familiar name points to the costlier or slower tool. The cost or latency is stated as a tag, in natural language, or as a tag together with an abbreviated name for the correct tool. Each type is evaluated in two versions. In the explicit version, as in RobustBench-TC, the query asks for the cheaper or faster tool; in the implicit version, which we add, the query asks the agent to describe the available tools and their properties before acting and states no preference. With canonical names, we canonicalize after constructing each interface, and because canonical identifiers follow the menu order, we average two orders, the original one and one in which the costlier or slower equivalent comes first; with original names, we use the original order. Figure 4 shows one item under each construction.

Action surface and decoding. Qwen3.5 models are trained and evaluated with native function calling, in which the tool menu is passed through the model's tool interface and calls are parsed by its structured parser. Qwen2.5 models receive the tools in the prompt. We report the two surfaces separately. Every model decodes with temperature 0 and at most 4,096 new tokens, and Qwen3.5 models generate with thinking enabled, the default of their chat template.

Scoring. A response is correct if it calls the correct tool with the correct arguments. The released checker validates a single-tool item against the first declared tool and a duplicated name against its first declaration, so once a perturbation adds a tool it can reject the correct call and accept a call to the added one; we validate against the tool that the reference call names and accept a call if any declaration of the called name permits it. Intervals are 95% confidence intervals over the shared item set, paired over items for the gap, and do not reflect variation across training runs.

Canonicalization
`BFCL item multiple_73, original interface`

Query. Who was the founder of Buddhism and where was it originated?

Original names.
`religion.get_core_beliefs(religion)`: Retrieves the core beliefs and practices of a specified religion.
`religion.get_origin(religion)`: Retrieves the origin and founder information of a specified religion.
Reference call: `religion.get_origin(religion="Buddhism")`

Canonical names.
`function_01(arg_01)`: Retrieves the core beliefs and practices of a specified religion.
`function_02(arg_01)`: Retrieves the origin and founder information of a specified religion.
Reference call: `function_02(arg_01="Buddhism")`

Cost, tag
`the same item with an added costlier equivalent`

Original names.
`religion.get_core_beliefs`: … `[Cost: $0.107/call]`
`religion.get_origin_1`: Retrieves the origin and founder information of a specified religion. `[Cost: $0.010/call]`
`religion.get_origin`: Retrieves the origin and founder information of a specified religion. `[Cost: $0.210/call]`

Canonical names, original order. `function_01`: … $0.107; `function_02`: … $0.010; `function_03`: … $0.210.
Canonical names, costlier first. `function_01`: … $0.107; `function_02`: … $0.210; `function_03`: … $0.010.

The natural-language form states the cost as `(costs approximately $0.010 per call)`; the abbreviated form also shortens the name of the correct tool.

Explicit version. The query ends with one of *"Please use a cost-effective option."*, *"Choose the option with minimal cost."*, or *"I have a limited budget for this task."*; the latency types use *"Use the tool with the shortest response time."* or *"Please use a fast option."*

Implicit version. The query ends with *"Before making a decision, give a short record of every tool, what it does, and its important properties."*

Figure 4: One BFCL item before and after canonicalization (top), and under the cost perturbation with the query endings of the explicit and implicit versions (bottom). Descriptions are unchanged by canonicalization; tool and argument names are replaced.

D Additional results

This appendix reports the results of Table 1 by perturbation type and at two further sizes of Qwen3.5, the τ^2-bench results, and every ablation run.

Results by perturbation type. Tables 6 and 7 break each held-out condition of Table 1 into its perturbation types, and the second adds CIPR at two further sizes of Qwen3.5. Each mean row is the corresponding column of Table 1. Half-widths of 95% confidence intervals are at most 2.0 points for the original interface, the condition means, and single types, and at most 1.5 points for the gap.

Interactive tasks. Table 8 reports the τ^2-bench values summarized in Section 4.2. Rewards are averaged over scored simulations.

Table 5: Held-out perturbation types used on BFCL, adapted from RobustBench-TC and regenerated for all 2,501 items. In all six cost and latency types the correct tool receives a new name and the familiar name goes to the costlier or slower equivalent, which RobustBench-TC does only in its abbreviated variants, and the implicit version is ours. None is used during training.

Condition	Types	What changes
Typos & distractor tools	Typo	Character-level typos, with letters swapped, substituted, dropped, or doubled, in about one in five query words of four or more letters.
	RedunTool	For each tool the task calls, an unrelated tool whose name matches that tool's name except in the last word.
	Dup-NoDesc	For each tool the task calls, a duplicate with the same name, no description, and no parameters.
	Dup-Desc	The same duplicate with the original description and no parameters.
	Dup-WrongP	The same duplicate with no description and the parameters of another tool.
	Dup-DescWP	The same duplicate with the original description and the parameters of another tool.
	Dup-SwapDP	The same duplicate with the description and parameters of another tool.
Cost & latency, explicit and implicit	Cost description as a tag, in natural language, or with an abbreviated name	An equivalent with a higher stated cost keeps the original name, and the correct tool is renamed.
	Latency description, same three forms	An equivalent with a higher stated latency keeps the original name, and the correct tool is renamed.
	Explicit version	The query asks for the cheaper or faster tool.
	Implicit version (ours)	The query asks for a description of the tools and states no preference.

Ablation runs. Table 9 lists every ablation run of Section 4.4 with its accuracy under typos and distractor tools, and its accuracy on the explicit cost and latency perturbations in each menu order. All runs share the hyperparameters of Table 3, except that one run samples groups of $G = 16$.

The two menu orders show what each perturbed interface teaches the agent to rely on. After training with a single added tool, accuracy is 70.8% when the correct tool comes first and 16.5% when the costlier equivalent does; reordering alone reverses the difference, and twins with reordering reduce it to between 10 and 14 points. Canonical names help in the original order whatever the training, since ToolRL (GRPO) rises from 29.6% with the original names to 59.7% with canonical names and the CIPR model from 28.7% to 60.5%. The advantage of CIPR over ToolRL (GRPO) therefore lies in the order in which the costlier equivalent comes first, 46.9% against 32.4%.

Three comparisons do not appear in Table 2. Giving calls to a degraded twin a lower reward than other wrong calls leaves multi-interface training unchanged, 51.4% against 51.7%, so the final recipe uses no such penalty. With a single added tool, letting its rollouts enter only the consistency terms gives 45.4%, against 43.7% when they also enter the task loss, and the final recipe trains on the perturbed rollouts (Section 3.4). With reordering as the only perturbation, groups of $G = 16$ give 46.3%, against 48.1% with $G = 4$.

E Error analysis

Aggregate accuracy shows how often a model is correct, not what it does when it is wrong. We decompose the failures of Qwen3.5-4B ToolRL (GRPO) and CIPR under the implicit cost and latency perturbations, pooling the six types and both menu orders. Each model is scored on the same 30,012 pairs of condition and item.

Taxonomy. The benchmark's error code for a wrong tool name does not distinguish calling the costlier or slower equivalent of the correct tool from calling an unrelated tool, so we resolve each

Table 6: Accuracy (%) on BFCL by perturbation type for Qwen2.5-3B, with tools in the prompt. Types are those of Table 5. Base and ToolRL (GRPO) are evaluated with the original names and, in gray, through the canonicalization adapter; the multi-interface and CIPR models use canonical names. With canonical names, each cost and latency entry averages the two menu orders.

	Base		ToolRL (GRPO)			
	original	canonical	original	canonical	+ multi-interface	+ CIPR
Original interface	63.7	61.5	73.3	70.7	74.0	74.9
Typos & distractor tools						
Typo	59.1	57.4	68.7	67.3	69.6	70.5
RedunTool	63.2	62.6	72.5	69.5	73.5	73.6
Dup-NoDesc	58.3	62.2	72.4	71.9	73.6	75.1
Dup-Desc	56.8	60.9	72.3	71.4	73.6	74.6
Dup-WrongP	60.2	58.1	71.0	68.1	70.1	70.7
Dup-DescWP	59.6	54.4	71.7	65.0	68.7	68.9
Dup-SwapDP	62.5	59.3	72.5	69.4	71.8	72.2
Mean	59.9	59.3	71.6	68.9	71.5	72.2
Cost & latency (explicit)						
Cost, tag	18.3	43.7	24.2	47.6	52.0	53.0
Cost, natural language	18.8	43.2	22.4	48.1	52.4	52.9
Cost, abbreviated name	26.5	30.9	35.8	32.8	36.9	37.4
Latency, tag	26.5	49.8	29.8	54.2	59.7	63.9
Latency, natural language	22.5	45.3	23.9	50.6	54.8	56.9
Latency, abbreviated name	32.4	38.4	39.5	42.9	53.3	57.7
Mean	24.2	41.9	29.3	46.0	51.5	53.6
Cost & latency (implicit)						
Cost, tag	13.4	30.0	15.3	34.3	37.8	38.7
Cost, natural language	11.5	27.9	13.8	32.7	36.8	38.1
Cost, abbreviated name	22.6	24.4	29.3	29.8	30.8	30.7
Latency, tag	17.4	30.9	19.5	35.6	45.3	50.9
Latency, natural language	13.1	29.3	13.9	34.6	41.1	45.5
Latency, abbreviated name	28.3	28.4	33.7	34.1	45.7	49.8
Mean	17.7	28.5	20.9	33.5	39.6	42.3
Mean of the three conditions	33.9	43.2	40.6	49.5	54.2	56.1
Gap	29.8	18.3	32.7	21.2	19.8	18.8

failure against the declared menu. A wrong call is attributed to the equivalent when the called tool has the same description as the correct one once the cost or latency annotation is removed, and to an unrelated tool otherwise. Of the wrong-name errors made by ToolRL (GRPO), 98.9% call the equivalent and 1.1% an unrelated tool.

Results. Figure 5 shows that the reduction concentrates in one class, calling the costlier or slower equivalent, which falls by 24.56 points, from the largest failure class to about a quarter of its former share. This is the decision the consistency KL constrains, since it compares distributions over tools and the no-call outcome and does not inspect arguments.

Argument failures barely change, and their share rises because more items reach the argument check. Given that the correct tool was called, the rate of argument failure is 12.07% for ToolRL (GRPO) and 11.61% for CIPR.

Two small classes move against CIPR. The CIPR model declines to call any tool on 388 more pairs and emits the wrong number of calls on 120 more. The query in this family asks for a description of the tools before acting, and the CIPR model is more likely to give the description and stop. Because the action distribution includes a no-call entry, an objective that equalizes distributions across interfaces can be partly satisfied by moving probability onto not calling, and we regard this as a cost of the current instantiation.

Table 7: Accuracy (%) on BFCL by perturbation type for Qwen3.5 with native function calling. All columns are Qwen3.5-4B except the last two, which train CIPR on Qwen3.5-2B and Qwen3.5-9B. Rows and conventions as in Table 6.

	Base		ToolRL (GRPO)			+ CIPR		
	original	canonical	original	canonical	+ multi-interface	4B	2B	9B
Original interface	84.2	82.0	84.8	84.6	84.8	85.0	74.9	85.8
Typos & distractor tools								
Typo	79.2	78.0	80.2	79.9	79.6	80.7	68.1	81.8
RedunTool	84.5	81.8	84.7	85.0	84.6	85.1	75.5	85.9
Dup-NoDesc	84.4	82.4	84.9	85.0	85.2	85.0	74.7	86.0
Dup-Desc	84.6	82.0	85.1	84.3	85.2	85.3	75.0	85.7
Dup-WrongP	84.0	82.6	84.8	84.4	84.6	85.7	72.1	85.6
Dup-DescWP	83.0	80.4	84.3	84.6	84.2	84.7	65.3	85.2
Dup-SwapDP	84.0	81.8	85.0	84.6	85.3	85.3	74.6	85.4
Mean	83.4	81.3	84.2	84.0	84.1	84.5	72.2	85.1
Cost & latency (explicit)								
Cost, tag	83.5	80.9	85.0	84.1	83.8	84.6	66.3	84.9
Cost, natural language	83.2	81.0	85.2	83.8	84.3	84.5	67.1	85.0
Cost, abbreviated name	79.4	76.7	82.2	81.0	83.1	82.3	56.4	84.3
Latency, tag	84.6	83.3	85.6	85.1	84.8	85.7	74.0	85.8
Latency, natural language	84.0	82.6	85.6	85.2	84.4	85.5	74.1	86.1
Latency, abbreviated name	83.7	81.5	85.2	84.6	84.5	85.2	59.1	85.5
Mean	83.1	81.0	84.8	84.0	84.2	84.6	66.1	85.3
Cost & latency (implicit)								
Cost, tag	58.1	62.8	64.9	55.3	71.6	75.1	44.6	70.6
Cost, natural language	60.2	63.4	65.5	55.6	71.4	74.8	47.1	70.4
Cost, abbreviated name	54.8	52.8	57.0	49.9	66.2	67.5	35.8	68.7
Latency, tag	54.7	63.1	60.2	54.2	73.2	75.2	53.2	72.5
Latency, natural language	51.1	60.3	58.1	51.8	71.9	74.5	50.5	72.9
Latency, abbreviated name	53.0	50.6	55.4	47.1	70.1	71.0	43.9	69.3
Mean	55.3	58.8	60.2	52.3	70.7	73.0	45.8	70.7
Mean of the three conditions	73.9	73.7	76.4	73.4	79.7	80.7	61.4	80.4
Gap	10.2	8.3	8.4	11.1	5.1	4.2	13.5	5.4

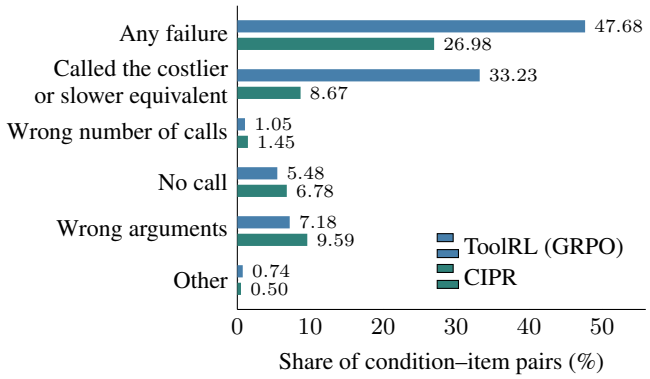

Figure 5: Failures of Qwen3.5-4B under the implicit cost and latency perturbations, as a percentage of the 30,012 condition–item pairs. The classes partition the failures, so the bars below *Any failure* sum to it up to rounding.

Examples. Figure 6 gives instances of the failure classes, with the canonical names the model saw. The upper part of each box shows what the model was given and the reference call; the lower part shows the calls that the models made and the scorer's message, as logged. The evaluation logs a model's text only when no call can be parsed, which is why only the no-call box quotes it.

Table 8: Mean task reward ($\times 100$) on τ^2-bench. Rows and conventions follow Table 1, with Base and ToolRL (GRPO) using the original names and the multi-interface and CIPR models canonical names. *Original* is 50 simulations on airline and 74 to 93 scored simulations of the 114 retail tasks, with 95% confidence half-widths of 12 to 14 points on airline and 4 to 11 on retail.

	Airline				Retail			
Training	Original	Typos & distractors	Cost & lat. explicit	Cost & lat. implicit	Original	Typos & distractors	Cost & lat. explicit	Cost & lat. implicit
Qwen2.5-3B, tools in prompt								
Base	42.0	38.0	37.7	37.0	6.7	6.2	6.4	5.6
ToolRL (GRPO)	38.0	39.4	42.0	37.8	5.5	6.3	6.6	5.5
+ multi-interface	40.0	37.4	40.3	37.3	6.6	6.4	6.3	6.1
+ CIPR	44.0	40.3	39.2	39.5	6.5	6.2	5.9	6.1
Qwen3.5-4B, native function calling								
Base	76.0	71.1	70.3	72.3	65.3	57.3	59.7	60.5
ToolRL (GRPO)	76.0	73.7	72.7	73.7	66.7	63.2	65.5	63.4
+ multi-interface	74.0	66.6	69.8	71.2	58.1	53.0	58.5	59.1
+ CIPR	68.0	72.9	71.2	71.5	57.3	54.7	56.0	58.3
+ CIPR at other sizes, native function calling								
Qwen3.5-2B	48.0	37.1	44.0	43.2	30.4	17.8	22.1	20.7
Qwen3.5-9B	68.0	68.3	71.5	72.3	61.0	59.8	63.5	63.5

Table 9: Every ablation run on Qwen2.5-3B, with canonical names (accuracy, %). The last three columns split the explicit cost and latency perturbations of Table 2 by menu order, the original order and the one in which the costlier or slower equivalent comes first. Configurations shared with Table 1 differ from it by at most 0.5 points.

				Cost & latency (explicit)		
Perturbed interface	Consistency KL	Original	Typos & distractor tools	Original order	Costlier first	Mean
Twins, reordered	actions	74.7	72.1	60.5	46.9	53.7
	–	73.5	71.3	56.7	46.7	51.7
	–, twin penalty	73.6	70.7	56.4	46.4	51.4
Reordered	actions	73.5	71.8	36.3	59.9	48.1
	actions, $G = 16$	75.5	72.9	26.3	66.4	46.3
	–	75.0	72.2	37.0	58.1	47.5
One added tool	actions	74.7	72.2	70.8	16.5	43.7
	–	74.3	71.9	72.0	16.1	44.1
One added tool, not in the task loss						
	actions	73.4	70.7	70.0	20.8	45.4
None, ToolRL (GRPO)	–	70.7	68.8	59.7	32.4	46.0

F Interface robustness on RoTBench

We also evaluate the same models on RoTBench (Ye et al., 2024), which scores tool calls under five levels of noise in the names of tools and parameters.

Benchmark. We use the First-Turn split and all five shipped levels, a clean menu and the *slight*, *medium*, *heavy*, and *union* corruptions derived from it. Following the definitions of Ye et al. (2024), the slight level inserts, omits, or substitutes characters in the names of half of the tools or half of the parameters, and the medium level reverses such names or replaces them with random strings. The heavy level either shuffles all tool names among the tools, which detaches each name from its description, or, in half of the tools, adds a mandatory parameter with a random name and shuffles the parameter names, each with probability one half. The union level combines one noise on tool names with one on parameter names, drawn from the other levels. The corruptions ship as data,

Called the costlier or slower equivalent
`live_multiple_78-35-2`

Menu. `function_04` [$0.013] and `function_05` [$0.135], both described as "Provides assistance on a specific topic within the DartFX application".
Reference. `function_04`

ToolRL (GRPO) `[{"function_05": {"arg_01": "charts", "arg_02": "Spanish"}}]` ✗
Scorer: `Function name 'function_04' not found in model output.` The same tool at ten times the stated cost.

Called the costlier or slower equivalent
`live_multiple_61-23-0`

Query. "What movies are playing in Noida today?"
Menu. `function_01` [$0.016] and `function_02` [$0.191], both described as "Retrieves a list of movies playing in a specified city and cinema hall".
Reference. `function_01`

CIPR `[{"function_02": {"arg_01": "Noida"}}]` ✗
Scorer: `Function name 'function_01' not found in model output.` CIPR reduces this class but does not eliminate it.

Wrong number of calls
`live_multiple_190-84-0`

Query. "Find out the rewards for playing Fortnite on Playstation platform with different missions and trophies".
Menu. `function_01` [$0.011] and its equivalent `function_02` [$0.118].
Reference. `function_01`, called once.

ToolRL (GRPO) `function_01` called four times, once for each of `bronze`, `silver`, `gold`, and `platinum`. ✗
Scorer: `Wrong number of functions.`

No call
`live_simple_20-4-0`

Query. "Order a burger … no onions, extra cheese", together with the request to describe the available tools before acting.
Reference. `function_01`

CIPR No call is parsed; the logged output is prose, ending:
"Both functions serve the same purpose but differ significantly in cost. Function 01 is substantially cheaper ($0.015 vs $0.104), making it the more economical choice for this request. […] Would you like me to proceed with modifying the burger using Function 01?" ✗
Scorer: `Wrong number of functions.` The inventory is produced as asked, and the decision is deferred back to the user.

Wrong argument value
`live_multiple_4-2-1`

Query. A ride request stating "Address: 123 Hanoi Street".
Reference. The correct tool with the address `'123 Hanoi Street'`.

Both models `[{"function_01": {"arg_01": "123 Hanoi Street, Hanoi, Vietnam", "arg_02": "plus", "arg_03": 10}}]` ✗
Scorer: `Invalid value for parameter 'arg_01': '123 Hanoi Street, Hanoi, Vietnam'. Expected one of ['123 Hanoi Street'].`

Wrong argument type
`live_simple_106-63-0`

Menu. A query-classification tool whose `arg_04` is declared an `array` of `string`.

Both models ✗ Scorer: `Nested type checking failed for parameter 'arg_04'. Expected outer type array with inner type 'str'.`

Figure 6: Instances of the failure classes of Qwen3.5-4B under the implicit cost and latency perturbations. Descriptions are abbreviated, bracketed amounts are the stated costs, and […] marks omitted text.

so every model sees an identical menu at a given level. Scoring is the benchmark's deterministic Content-Filling criterion, which requires the predicted call to match an accepted reference on tool

name, argument keys, and argument values. We apply the benchmark's matcher unmodified and recompute it per item so that conditions can be paired.

Canonical condition. In the canonical condition, we replace every tool name with a canonical identifier assigned by position and leave descriptions, argument values, and the query unchanged. Because RoTBench's corruptions act mainly on names, canonicalizing a shipped corruption file would give a corrupted name the same canonical identifier as its clean version and undo most of the corruption. We therefore canonicalize the clean record first and re-apply each level's corruption to the canonical identifier. This preserves the structure of each level, introduces no duplicated identifier in either condition, and keeps the queries and their order identical, so the pairing is exact.

Results. Table 10 reports both conditions. Under original names, every model loses the most accuracy on the heavy and union levels. Canonicalization costs accuracy on the clean and lightly corrupted menus and recovers it on the heavy and union levels. On the heavy level, half of the items shuffle the tool names, so under original names each name labels the description of a different tool, which can mislead a model that selects tools by name; under canonical names the shuffled labels carry no meaning, and the choice falls back on the descriptions. Under canonical names, the spread between the clean menu and the corrupted levels narrows for every model, in several cases by more than half.

Comparison across backbones. The two backbones are evaluated on the action surface each was trained for, so entries can be compared within a backbone but not across backbones. On Qwen2.5-3B, canonical names help on the corrupted levels, and with them CIPR and multi-interface training share the best pooled accuracy. The largest effect in the table is scale, since the largest CIPR model is both the most accurate and the flattest across corruption levels, with a drop of only 0.3 points from the clean menu once names are canonical.

Table 10: Content-Filling accuracy (%) on the RoTBench First-Turn split under original names and canonical names. The last column pools the four corrupted levels. Subscripts are half-widths of 95% confidence intervals, computed per item and paired across conditions; $n = 105$ for Clean and Union and $n = 210$ for the other levels. Bold marks the best entry per column within a backbone.

	Corruption level					
Model	Clean	Slight	Medium	Heavy	Union	Corrupted
Original names						
Qwen2.5-3B						
Base	$39.0_{\pm 9.5}$	$37.1_{\pm 6.7}$	$38.1_{\pm 6.7}$	$21.4_{\pm 5.5}$	**28.6**$_{\pm 9.0}$	$31.7_{\pm 3.3}$
ToolRL (GRPO)	$40.0_{\pm 9.5}$	**41.0**$_{\pm 6.9}$	**41.0**$_{\pm 6.7}$	$22.9_{\pm 5.5}$	$23.8_{\pm 8.1}$	$33.3_{\pm 3.4}$
+ multi-interface	$40.0_{\pm 9.5}$	$40.0_{\pm 6.7}$	**41.0**$_{\pm 6.7}$	$23.8_{\pm 5.7}$	$25.7_{\pm 8.6}$	$33.6_{\pm 3.4}$
+ CIPR	**44.8**$_{\pm 9.5}$	$40.0_{\pm 6.7}$	$39.5_{\pm 6.7}$	**25.7**$_{\pm 6.0}$	$25.7_{\pm 8.6}$	**33.7**$_{\pm 3.5}$
Qwen3.5-4B						
Base	**53.3**$_{\pm 9.5}$	$41.9_{\pm 6.7}$	$38.1_{\pm 6.4}$	$32.9_{\pm 6.4}$	$28.6_{\pm 8.6}$	$36.3_{\pm 3.5}$
ToolRL (GRPO)	$42.9_{\pm 9.5}$	$33.8_{\pm 6.2}$	$27.6_{\pm 6.0}$	$27.1_{\pm 6.0}$	$21.9_{\pm 8.1}$	$28.4_{\pm 3.3}$
+ multi-interface	**53.3**$_{\pm 9.5}$	$49.5_{\pm 6.7}$	**51.0**$_{\pm 6.9}$	**39.5**$_{\pm 6.7}$	**39.0**$_{\pm 9.5}$	**45.6**$_{\pm 3.7}$
+ CIPR	$52.4_{\pm 9.5}$	**50.0**$_{\pm 6.7}$	$46.2_{\pm 6.7}$	$36.2_{\pm 6.7}$	$38.1_{\pm 9.5}$	$43.3_{\pm 3.5}$
+ CIPR across sizes						
Qwen3.5-2B	$41.9_{\pm 9.5}$	$29.5_{\pm 6.2}$	$25.2_{\pm 6.0}$	$21.9_{\pm 5.5}$	$21.9_{\pm 8.1}$	$25.0_{\pm 3.1}$
Qwen3.5-9B	**57.1**$_{\pm 9.5}$	**55.2**$_{\pm 6.7}$	**58.1**$_{\pm 6.7}$	**46.7**$_{\pm 6.7}$	**43.8**$_{\pm 9.5}$	**52.0**$_{\pm 3.7}$
Canonical names						
Qwen2.5-3B						
Base	$33.3_{\pm 9.0}$	$31.4_{\pm 6.2}$	$32.9_{\pm 6.7}$	$31.4_{\pm 6.2}$	$32.4_{\pm 8.6}$	$32.0_{\pm 3.3}$
ToolRL (GRPO)	$37.1_{\pm 9.5}$	$31.4_{\pm 6.2}$	**34.3**$_{\pm 6.4}$	$36.2_{\pm 6.7}$	$32.4_{\pm 8.6}$	$33.7_{\pm 3.3}$
+ multi-interface	$41.0_{\pm 9.5}$	$36.2_{\pm 6.4}$	$33.8_{\pm 6.7}$	**39.0**$_{\pm 6.7}$	**36.2**$_{\pm 9.5}$	**36.3**$_{\pm 3.4}$
+ CIPR	**43.8**$_{\pm 9.5}$	**37.1**$_{\pm 6.7}$	**34.3**$_{\pm 6.4}$	**39.0**$_{\pm 6.7}$	$33.3_{\pm 9.0}$	**36.3**$_{\pm 3.5}$
Qwen3.5-4B						
Base	$42.9_{\pm 9.5}$	$37.1_{\pm 6.7}$	$26.7_{\pm 5.7}$	$43.3_{\pm 6.7}$	$34.3_{\pm 8.6}$	$35.5_{\pm 3.5}$
ToolRL (GRPO)	$37.1_{\pm 9.0}$	$33.8_{\pm 6.2}$	$23.3_{\pm 5.5}$	$38.1_{\pm 6.4}$	$28.6_{\pm 8.6}$	$31.3_{\pm 3.5}$
+ multi-interface	**52.4**$_{\pm 9.5}$	**51.0**$_{\pm 6.7}$	**43.3**$_{\pm 6.7}$	**51.9**$_{\pm 6.9}$	**50.5**$_{\pm 9.5}$	**49.0**$_{\pm 3.6}$
+ CIPR	$43.8_{\pm 9.5}$	$43.3_{\pm 6.7}$	$35.7_{\pm 6.7}$	$40.5_{\pm 6.7}$	$39.0_{\pm 9.5}$	$39.7_{\pm 3.5}$
+ CIPR across sizes						
Qwen3.5-2B	$40.0_{\pm 9.5}$	$28.1_{\pm 6.0}$	$30.0_{\pm 6.0}$	$42.9_{\pm 6.7}$	$27.6_{\pm 8.6}$	$32.8_{\pm 3.4}$
Qwen3.5-9B	**46.7**$_{\pm 9.5}$	**44.3**$_{\pm 6.7}$	**46.2**$_{\pm 6.7}$	**47.6**$_{\pm 6.7}$	**48.6**$_{\pm 9.5}$	**46.4**$_{\pm 3.5}$