# JTPRO: A Joint Tool–Prompt Reflective Optimization Framework for Language Agents

**Sandip Ghoshal, Anshul Mittal, Jyotika Singh, Miguel Ballesteros, Weiyi Sun, Fang Tu, Shailender Singh, Yassine Benajiba, Sujeeth Bharadwaj, Fahad Shah, Sujith Ravi, Dan Roth**
Oracle AI
**Correspondence:** sandip.ghoshal@oracle.com

## Abstract

Large language model (LLM) agents augmented with external tools often struggle as number of tools grow large and become domain-specific. In such settings, ambiguous tool descriptions and under-specified agent instructions frequently lead to tool mis-selection and incorrect slot/value instantiation. We hypothesize that this is due to two root causes: generic, one-size-fits-all prompts that ignore tool-specific nuances, and underspecified tool schemas that lack clear guidance on when and how to use each tool and how to format its parameters. We introduce **J**oint **T**ool-**P**rompt **R**eflective **O**ptimization (**JTPRO**), a framework for improving tool-calling reliability in *trace-supervised* settings by iteratively using rollout-driven reflection to co-optimize global instructions and per-tool schema/argument descriptions for accurate tool selection and argument instantiation in large tool inventories. JTPRO is designed to preserve only tool-local cues needed for correct disambiguation and slot filling. We evaluate JTPRO across multi-tool benchmarks, which account for different number of tools using three metrics: Tool Selection Accuracy (TSA), Slot Filling Accuracy(SFA), and Overall Success Rate(OSR) (correct tool + correct slots + correct values). JTPRO consistently outperforms strong baselines, including CoT-style agents, and reflective prompt optimizers such as GEPA by 5%–20% (relative) on OSR. Ablations show that joint optimization of instructions and tool schemas is more effective and robust than optimizing either component in isolation.

## 1 Introduction

Tool-augmented large language model (LLM) (Vaswani et al., 2017) agents extend their capabilities by invoking external tools for specialized operations and up-to-date information (Wang et al., 2024) and are an important real-world application (Singh, 2023) across domains (Zhang, 2024; Meghwani et al., 2025; Singh et al., 2025). In this work[1], we focus specifically on *trace-supervised tool-calling settings*, where the objective is reliable call-level execution: (i) select the correct tool among many conflicting options, (ii) instantiate correct arguments from natural language requests; both suffer when tool/slot descriptions are ambiguous or underspecified (Qin et al., 2023). **Figure 2** quantifies this scaling failure on ToolACE (Liu et al., 2025): (a) tool selection accuracy drops as the tool universe expands, (b) a basic retrieval filter (top-20) only partially mitigates the decline . Crucially, end-to-end success is often bottlenecked by *slot/value instantiation*: on ETID (Enterprise Tool Inventory Dataset, a synthetic dataset developed internally for this study), **Figure 1** shows that improving slot filling produces large gains in overall success. Accordingly, our problem setting centers on reliable tool invocation under large inventories, where success depends on both correct tool selec-

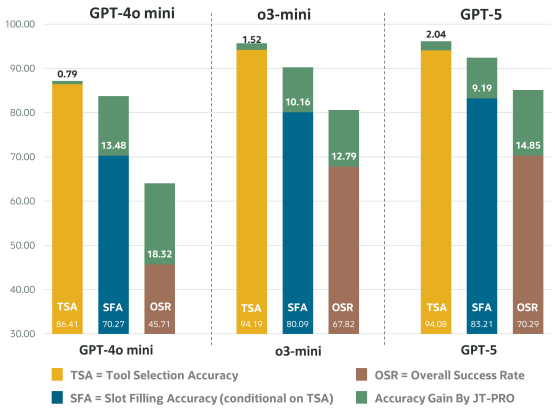


Figure 1: **Impact of slot-filling accuracy.** *Slot filling drives end-to-end success:* On the *Enterprise Tool-Inventory Dataset (ETID)* with complex schemas, we report TSA, SFA, and OSR; green overlays show absolute gains from JTPRO over baselines, highlighting that argument correctness is critical for OSR.

[1] www.sites.google.com/view/jtpro-etid/home



*Findings of the Association for Computational Linguistics: ACL 2026*, pages 40573–40595
July 2-7, 2026 

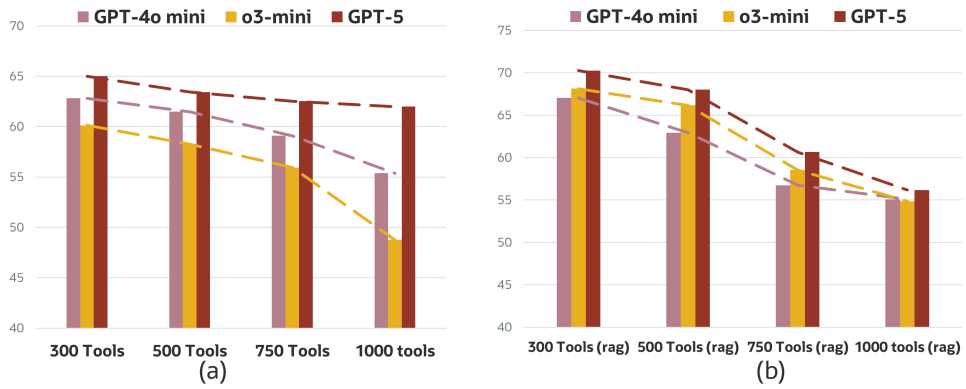


Figure 2: **Tool scaling failures and slot-filling impact. (a)** *All tools in context:* On ToolACE with an augmented inventory, tool selection accuracy drops as the tool set grows (300 to 1000), even for larger-context frontier models. **(b)** *Top-$k$ retrieval:* A basic RAG with reranker stage (top-20) does not remove the drop, indicating residual tool disambiguation/argument issues.

tion and correct argument instantiation.

Attempts to encode exhaustive tool and slot rules in lengthy global prompts are brittle, agents often fail to reliably follow extensive instructions, and maintaining cross-tool consistency becomes infeasible (Levy et al., 2024). **Figure 3** illustrates a representative tool-disambiguation failure that motivates JTPRO. In this example, two tools with overlapping descriptions, `get_all_countries` and `get_countries_list`, cause the baseline to mis-select the more generic tool for a request framed around investment analysis. After applying JTPRO, the tool descriptions are augmented with concise preference rules, specifying that `get_all_countries` should be used for general, non-investing requests, while `get_countries_list` should be preferred for investing or market-related queries. This resolves the ambiguity and yields the correct tool call, showing that targeted schema-level disambiguation can substantially improve tool selection in large tool inventories.

Prior work improves tool use via largely separate levers: model tuning, tuning-free prompting/-documentation, retrieval-based tool filtering, and prompt/context optimization. Tuning-free prompting (CoT (Wei et al., 2023), ReAct (Yao et al., 2023a)) and documentation refinement (DRAFT (Qu et al., 2025)) avoid weight updates but typically treat global instructions and tool schemas as static; retrieval-based selection reduces overload and iterative variants refine retrievers with agent feedback (Xu et al., 2024), yet retrieval alone does not fix downstream argument/format errors when slot semantics remain unclear. Prompt optimization and context evolution methods MIPRO (Opsahl-Ong et al., 2024); GEPA (Agrawal et al., 2025); AVATAR (Wu et al., 2024c); Dynamic Cheatsheet (Suzgun et al., 2025); ACE (Zhang et al., 2025) improve instruction-level behavior, but do not *jointly* adapt global decision rules and per-tool argument schemas at scale; similarly, Wu et al. (2025) refine prompts and tool descriptions but target efficiency rather than call-level tool/slot/value correctness under large domain tool stacks. JTPRO is best viewed as building on reflective optimization ideas and extending them to the *joint* optimization of multiple agent operating components, global instructions and per-tool schema/argument descriptions for reliable tool calling.

Our core contributions are as follows:

- **Joint optimization of tool/slot-schema and global instructions (JTPRO).** We formulate *joint* optimization of (i) the global instruction prompt $P$ and (ii) per-tool schema/argument descriptions $\{T_i\}$, targeting end-to-end invocation correctness (tool + slots + values) *without* model fine-tuning. This is critical as tool-use failures are inherently *coupled*: global policies depend on tool-local distinctions, and accurate slot/value instantiation relies on global conventions. Isolated optimization of $P$ or $\{T_i\}$ is insufficient to address these interdependent failure modes.

- **Reflection-driven, localized edits with controlled growth.** Inspired by reflection-augmented prompt engineering (Agrawal et al., 2025), JTPRO diagnoses systematic rollout failures (tool confusion, missing constraints, and formatting/-value errors) and issues targeted edits to both $P$ and the relevant tool/slot descriptions. To prevent

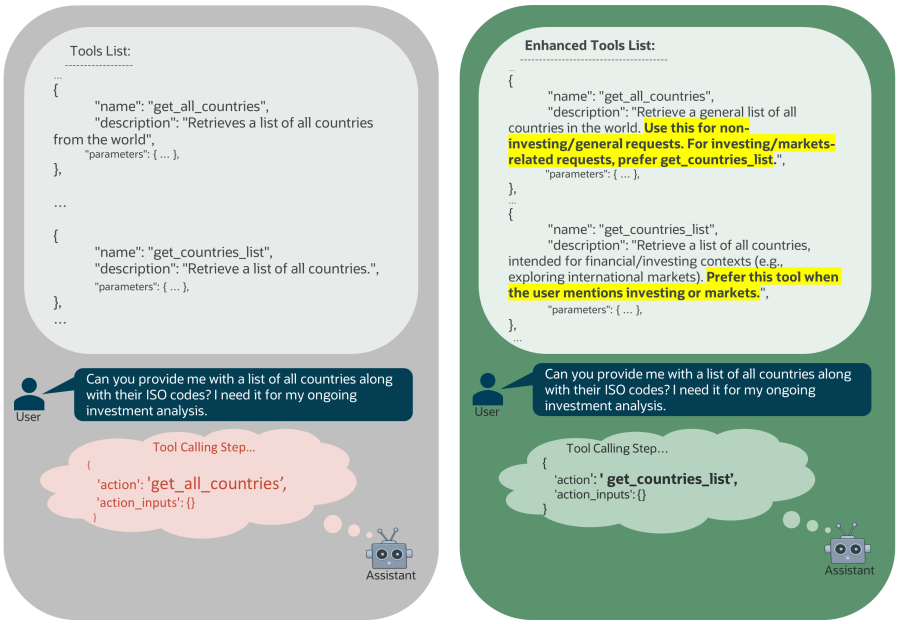


Figure 3: **Motivating context refinement with JTPRO, Tool disambiguation:** Baseline docs under-specify two similar tools, causing mis-selection; JTPRO adds brief per-tool decision rules (highlighted) to enable correct choice.

context bloat, we *globalize* recurring cross-tool slot semantics including date/time fields, numeric bounds, boolean parameters, sorting conventions, and currency/units into $P$, and replace redundant tool-local descriptions with short pointers to these shared rules. This reduces duplicated and potentially inconsistent schema text, while preserving tool-specific fields, exceptions, and disambiguation cues locally without merging or aliasing tools, which is important for real-world production systems; further details and examples are provided in Figures 9 and 10.

- **Empirical evaluation under realistic constraints.** We benchmark JTPRO in both single- and multi-tool environments with variable argument structures, reporting Tool Selection Accuracy, Slot Filling Accuracy (conditional on tool correctness), and Overall Success Rate. JTPRO demonstrates clear gains over strong baselines (such as baseline CoT, GEPA, and MIPRO) and further enhances retrieval-based pipelines by improving both retrieval and downstream slot filling.

## 2 Related Work

Tool-use learning spans (i) tuning-based adaptation, (ii) tuning-free prompting and documentation refinement, (iii) retrieval-based tool selection, and (iv) prompt/context optimization. Most methods improve *either* the global prompt *or* tool specifications, but rarely their *joint* co-adaptation under large, evolving tool inventories.

**Tuning-based tool learning.** Model-tuning methods learn tool use by updating parameters or adding trainable modules from tool traces or preference/reward signals, including supervised fine-tuning (Qin et al., 2024), (Liu et al., 2025), contrastive objectives (Wu et al., 2024a), reinforcement learning (Feng et al., 2025; Qian et al., 2025), and tool-token embedding extensions (Alazraki and Rei, 2025). While effective, these methods require retraining as the underlying tools/schemas evolve.

**Tuning-free prompting and documentation refinement.** Prompting approaches like CoT and ReAct and agentic planners like RestGPT (Song et al., 2023) and HuggingGPT (Shen et al., 2023) elicit multi-step reasoning without weight updates, but usually treat instructions and tool schemas as static. DRAFT (Qu et al., 2025) improves per-tool documentation via trial-and-error, yet does not optimize global instruction policies or multi-tool interactions mediated by shared prompt rules.

**Retriever-based tool selection.** Retriever-based pipelines filter candidates via lexical/dense retrieval and specialized rerankers e.g., CRAFT (Yuan et al., 2024), ToolRerank (Zheng et al., 2024), COLT (Qu et al., 2024), improving scalability but not resolving argument/format errors when slot semantics are unclear. Iterative retrieval refinement with agent feedback (Xu et al., 2024; Agarwal et al., 2025; Pattnayak et al., 2025) reduces retriever–agent mis-

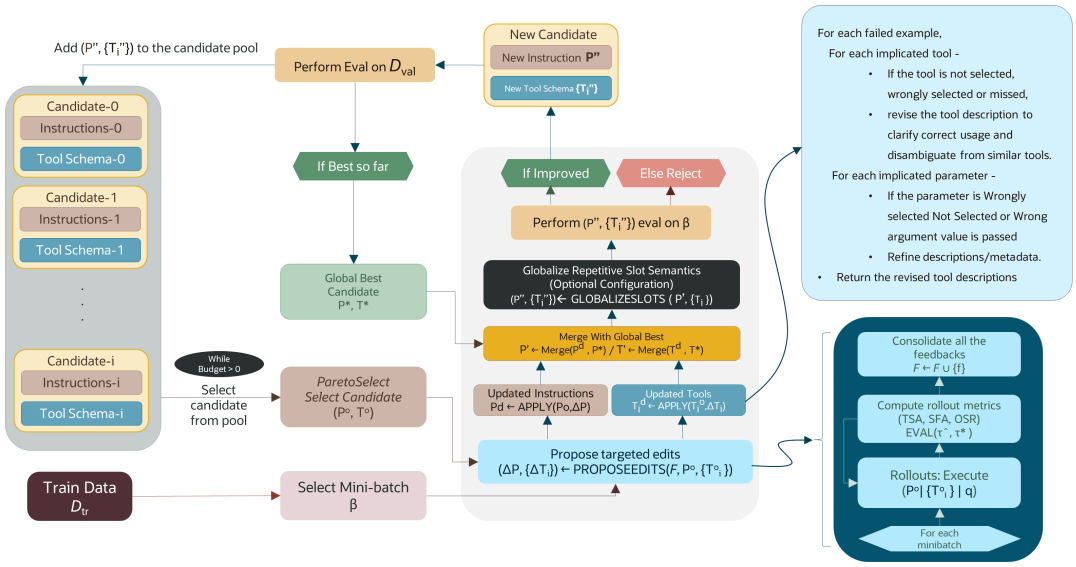


Figure 4: **JTPRO optimization loop (block-diagram view).** JTPRO maintains a pool of candidate contexts (global instructions $P$ and tool schemas $\{T_i\}$) and repeatedly (i) selects a candidate via Pareto-based sampling, (ii) runs minibatch rollouts on $\mathcal{D}_{tr}$ to compute tool-use metrics (TSA, SFA, OSR) and aggregate error feedback, and (iii) proposes localized edits to both $P$ and the implicated tool schemas. The edited instructions are merged with the current global-best $(P^\star, \{T_i^\star\})$, followed by optional globalization of repetitive slot semantics to avoid duplicated cross-tool parameter rules. Candidates that improve minibatch performance are validated on $\mathcal{D}_{val}$; improved candidates are added back to the pool, and the global best $(P^\star, \{T_i^\star\})$ is updated when a new highest validation score is observed.

match, but typically leaves the agent's instruction layer largely unchanged.

**Tool-using agents, tool construction, and prompt optimization.** Toolformer (Schick et al., 2023), ReAct (Yao et al., 2023b), and ReWOO (Xu et al., 2023) integrate tool calls into reasoning traces; DSPy (Khattab et al., 2024) and AutoPDL (Spiess et al., 2025) support declarative tool programs but assume static prompts/schemas. Other work constructs tools (TOOLMAKER (Wölflein et al., 2025)), optimizes tool-use prompts (AvaTaR (Wu et al., 2024c)), calibrates tool use (CITI (Hao et al., 2025), PROBECAL (Liu et al., 2024)), or improves tool policies via SFT/RL (Sullivan et al., 2025). Separately, self-refinement and prompt optimization (Madaan et al., 2023; Shin et al., 2020; Lester et al., 2021; Pryzant et al., 2023; Yuksekgonul et al., 2025; Singh et al., 2026; Zheng et al., 2026) and evolutionary search EvoPrompt (Guo et al., 2024) automate instruction improvement; MIPRO (Opsahl-Ong et al., 2024) optimizes module prompts and demonstrations, while GEPA (Agrawal et al., 2025) uses reflection over trajectories with Pareto selection, and AVATAR (Wu et al., 2024c) applies contrastive feedback. Dynamic Cheatsheet (Suzgun et al., 2025) and ACE (Zhang et al., 2025) motivate maintaining an evolving, curated context, but focus on strategy/memory rather than tool/argument schema co-adaptation.

**Reflective textual feedback for prompt/text optimization.** Recent work moves beyond scalar rewards (e.g., accuracy) by using *rich textual critiques* as the optimization signal, treating LLMs as optimizers that make targeted, gradient-like edits in text space. Maestro (Wang et al., 2025) places prompt optimization in a broader system loop, jointly updating agent graphs and configurations (including prompts) from reflective feedback over execution traces such as constraint violations and looping, improving reliability and sample efficiency. Feedback Descent (Lee et al., 2025) extends this view to an open-ended framework that uses pairwise comparisons, textual rationales, and accumulated feedback to iteratively revise prompts and other text artifacts. These approaches are complementary to tool-use settings: they show the value of structured, interpretable feedback for inference-time refinement, but do not directly address joint co-adaptation of *global instruction policies* and *per-tool argument/schema descriptions* in large tool inventories.

**Distinction.** In contrast to prior work that optimizes prompts or tool documentation separately, JTPRO jointly updates global instructions $P$ and

per-tool *tool/argument* schema descriptions $\{T_i\}$ using rollout-driven reflection, targeting call-level correctness (tool, slots, values) without model fine-tuning. JTPRO also reduces redundancy by abstracting shared slot conventions globally while preserving tool-specific details locally, leading to improved results in retrieval-based pipelines.

## 3 Problem Statement

We consider an LLM agent with access to a set of $N$ external tools (APIs/functions) $\{T_1, \ldots, T_N\}$. Each tool $T_i$ is specified by a schema/documentation entry describing its functionality and expected parameters (slots). Given a user query $Q$, the agent must produce an answer $A$, potentially by issuing one or more tool calls with structured arguments. The agent is guided by a global instruction prompt $P$ and the collection of tool schemas $\{T_i\}_{i=1}^N$.

For a query $Q$, the LLM is invoked with context

$$C(P, T, Q) \;=\; P \,\big|\, T_1 \,\big|\, \cdots \,\big|\, T_N \,\big|\, Q, \quad (1)$$

and produces a tool-call trace $\hat{\tau} = \hat{\tau}(P, T, Q)$.

Our objective is to optimize the textual content of $P$ and $\{T_i\}$ to maximize tool-use performance *without* model fine-tuning. Because tool identities and interfaces are typically fixed in production, we do *not* merge or alias tools. Instead, we allow edits to $P$ and each $T_i$, and we *globalize* recurring slot conventions (e.g., date/time formats, inclusive/exclusive bounds, currency/units) by lifting duplicated per-tool guidance into $P$.

We evaluate **call-level correctness**: correct tool selection and correct slot/value instantiation, summarized by **Tool Selection Accuracy**, **Slot Filling Accuracy** (conditional on correct tool), and **Overall Success Rate** (correct tool + correct slots + correct values). This emphasis matches deployments where executing tools and validating response-level correctness may be infeasible due to security, access control, rate limits, or non-deterministic backends.

Given a dataset $\mathcal{D} = \{(Q_j, \tau_j)\}_{j=1}^M$ with gold traces $\tau$, we optimize only two variables the global instructions $P$ and tool descriptions $T$ to maximize expected call-level correctness:

$$(P^\star, T^\star) = \arg\min_{P,T} \; \mathbb{E}_{(Q,\tau)\sim\mathcal{D}} \Big[\mathcal{L}\big(\hat{\tau}(P, T, Q), \tau\big)\Big]. \quad (2)$$

The loss function can be defined using tool selection, slot filling, and overall success:

$$\begin{aligned} \mathcal{L}(\hat{\tau}, \tau) &= \lambda_{\text{TSA}} \left(1 - \mathbb{I}[\hat{t} = t]\right) \\ &+ \lambda_{\text{SFA}} \, \mathbb{I}[\hat{t} = t] \left(1 - \text{Rec}(\hat{a}, a)\right) \\ &+ \lambda_{\text{OSR}} \left(1 - \mathbb{I}[\hat{t} = t \wedge \hat{a} = a]\right), \end{aligned} \quad (3)$$

where $\hat{t}$ and $t$ are the predicted and gold tool identifiers, $\mathbb{I}[\cdot]$ is the indicator function, $\hat{\mathbf{a}}$ and $\mathbf{a}$ are the predicted and gold argument structures, $\text{Rec}(\hat{a}, a)$ are slot/value recall conditional on $\hat{t} = t$, and $\lambda_{\text{TSA}}, \lambda_{\text{SFA}}, \lambda_{\text{OSR}}$ are nonnegative loss weights.

## 4 Technique

We present **Joint Tool–Prompt Reflective Optimization (JTPRO)**, a weight-free, context-level optimizer that iteratively updates (i) global agent instructions $P$ and (ii) per-tool schemas $\{T_i\}_{i=1}^N$ from labeled tool-call traces. Algorithm 1 summarizes the loop.

**Setup and objective** For each query $q$, the agent runs under $C(q) = P \,|\, T_1 \,|\, \cdots \,|\, T_N \,|\, q$ and produces a predicted trace $\hat{\tau}$. Given gold traces $\tau^\star$, JTPRO edits $P$ and $\{T_i\}$ to improve TSA, SFA (conditional on correct tool), and OSR (correct tool + correct slots + correct values).

**Candidate selection (Pareto)** JTPRO maintains a pool $\mathcal{C}$ of candidate contexts and uses GEPA-style Pareto selection: retain candidates that achieve the best score on at least one training instance, prune strictly dominated candidates, then sample a starting candidate with probability biased toward those that win on more instances.

**Rollouts, diagnostics, and localized edits** On a minibatch $\mathcal{B} \subset \mathcal{D}_{tr}$, we compute rollout metrics and extract structured failure signals $\mathcal{F}$ via $\textsc{Diagnose}(\hat{\tau}, \tau^\star)$ (e.g., tool confusions, missing required slots, formatting/value violations). A reflector proposes targeted edits $(\Delta P, \{\Delta T_i\}) \leftarrow \textsc{ProposeEdits}(\mathcal{F}, P^o, \{T_i^o\})$, which are applied to produce a draft context $P^d$ and $\{T_i^d\}$. Edits are localized to the implicated global rules and tool/slot descriptions.

**Merge-with-best for incremental tool adaptation** JTPRO tracks a validation-best context $C^\star = (P^\star, \{T_i^\star\})$. After editing, we merge $P^d$ with $P^\star$ using $\textsc{Merge}(P^d, P^\star)$ to form $P'$, and $\textsc{Merge}(T^d, T^\star)$ to form $T'$, implementing a "growing playbook" that preserves cross-cutting rules while adding new, rollout-driven guidance. This accumulation also supports incremental

**Algorithm 1:** JTPRO: Reflective Schema–Instruction Co-Optimization with Slot-Semantics Globalization

**Input:** initial global instructions $P^{(0)}$, initial tool schemas $\{T_i^{(0)}\}_{i=1}^N$, labeled training set $\mathcal{D}_{tr} = \{(q, \tau^\star)\}$, labeled validation set $\mathcal{D}_{val}$, max iterations $I$, batch size $B$
**Output:** optimized global instructions $P^\star$, optimized tool schemas $\{T_i^\star\}_{i=1}^N$

Initialize $P \leftarrow P^{(0)}$ and $T_i \leftarrow T_i^{(0)}$ for all $i \in \{1, \ldots, N\}$
Initialize best context $C^\star \leftarrow (P, \{T_i\}_{i=1}^N)$ and best validation score $s^\star \leftarrow -\infty$
Initialize pool $\mathcal{C} \leftarrow \{(P, \{T_i\}_{i=1}^N)\}$ `// candidate contexts`
**for** $t \leftarrow 1$ **to** $I$ **do** `// main optimization loop`
  Sample a minibatch $\mathcal{B} \subset \mathcal{D}_{tr}$ of size $B$
  $(P^o, \{T_i^o\}_{i=1}^N) \leftarrow \textsc{ParetoSelect}(\mathcal{C})$
  $s^o_{\mathcal{B}} \leftarrow \textsc{Score}((P^o, \{T_i^o\}_{i=1}^N), \mathcal{B})$
  Initialize aggregated feedback $\mathcal{F} \leftarrow \emptyset$
  **foreach** $(q, \tau^\star) \in \mathcal{B}$ **do**
    Construct context $C(q) \leftarrow P^o \,|\, T_1^o \,|\, \cdots \,|\, T_N^o \,|\, q$
    Run agent to obtain predicted trace $\hat{\tau} \leftarrow \textsc{Agent}(C(q))$
    Compute rollout metrics $(\text{TSA}, \text{SFA}, \text{OSR}) \leftarrow \textsc{Eval}(\hat{\tau}, \tau^\star)$
    Extract error signals $f \leftarrow \textsc{Diagnose}(\hat{\tau}, \tau^\star)$
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{f\}$
  $(\Delta P, \{\Delta T_i\}_{i=1}^N) \leftarrow \textsc{ProposeEdits}(\mathcal{F}, P^o, \{T_i^o\}_{i=1}^N)$
  $P^d \leftarrow \textsc{Apply}(P^o, \Delta P)$
  $T_i^d \leftarrow \textsc{Apply}(T_i^o, \Delta T_i)$ for all $i \in \{1, \ldots, N\}$
  $P' \leftarrow \textsc{Merge}(P^d, P^\star)$
  $T_i' \leftarrow \textsc{Merge}(T_i^d, T_i^\star)$ for all $i \in \{1, \ldots, N\}$
  $(P'', \{T_i''\}_{i=1}^N) \leftarrow \textsc{GlobalizeSlots}(P', \{T_i'\}_{i=1}^N)$
  $s''_{\mathcal{B}} \leftarrow \textsc{Score}((P'', \{T_i''\}_{i=1}^N), \mathcal{B})$
  **if** $s''_{\mathcal{B}} > s^o_{\mathcal{B}}$ **then**
    $s''_{\text{val}} \leftarrow \textsc{Score}((P'', \{T_i''\}_{i=1}^N), \mathcal{D}_{val})$
    **if** $s''_{\text{val}} \geq s^\star$ **then**
      $\mathcal{C} \leftarrow \textsc{AddToPool}(\mathcal{C}, (P'', \{T_i''\}_{i=1}^N), K)$
      **if** $s''_{\text{val}} > s^\star$ **then**
        $C^\star \leftarrow (P'', \{T_i''\}_{i=1}^N)$
        $(P^\star, \{T_i^\star\}_{i=1}^N) \leftarrow (P'', \{T_i''\}_{i=1}^N)$
        $s^\star \leftarrow s''_{\text{val}}$
**return** $C^\star$ *as* $(P^\star, \{T_i^\star\}_{i=1}^N)$

toolset expansion: when new $T_i$ are appended, stable global conventions remain intact and new tool-triggered rules are integrated without re-optimizing from scratch.

**Globalizing repetitive slot semantics** To reduce duplicated schema text, JTPRO applies $\textsc{GlobalizeSlots}(P', \{T_i'\}) \mapsto (P'', \{T_i''\})$, which identifies recurring cross-tool slot conventions and lifts them into named rules in $P''$ while replacing redundant tool-local descriptions with short pointers to those rules. Figure 9 motivates this step: in ETID, a small number of slot families, especially identifiers and date/time fields, but also numeric bounds, boolean flags, sorting parameters, and currency/unit fields—recur across many tools (up to 77/124), producing substantial repetition in per-tool schemas. Figure 10 illustrates the resulting two-level organization: shared fields such as `startDate`, `endDate`, `rangeMinimum`, `rangeMaximum`, and their inclusive flags point to global rules (e.g., *DateTime Fields*, *Numeric Bounds*, and *Boolean Parameters*), while tool-specific fields and local overrides remain in $T_i''$. This improves slot filling by enforcing consistent semantics across tools, reducing duplicated and potentially conflicting wording, and preserving schema space for tool-specific exceptions and disambiguation rules.

**Acceptance and pool update** We score $(P'', \{T_i''\})$ on $\mathcal{B}$ and, if improved, evaluate on $\mathcal{D}_{val}$. Improved candidates are added to $\mathcal{C}$ (bounded size $K$), and if a candidate is best on validation we update $C^\star$ accordingly.

**Summary** JTPRO combines Pareto-selected candidate search, reflection-driven localized edits to $P$ and $\{T_i\}$, and globalization of shared slot semantics to improve both tool selection and argument correctness in large tool inventories.

## 5 Datasets and Evaluation

### 5.1 Datasets

We evaluate JTPRO on three complementary benchmarks that stress different failure modes in tool-using agents: (i) complex, domain-specific slot filling with a moderate tool inventory, (ii) tool selection under toolset scaling, and (iii) a *multi-tool calling* setting where a single query may require invoking multiple tools in parallel and correctly instantiating arguments at each step. Our bench-

mark choices are matched to JTPRO's core setting: reusable tool schemas and stable train/validation/test distributions that permit transferable prompt and schema refinement, particularly in large tool inventories where tool selection and schema-constrained argument filling are the dominant bottlenecks. We discuss benchmark suitability, including the omission of other popular datasets, as well as the current scope of our evaluation with respect to sequential tool dependencies, in Appendix C.1.

**Enterprise Tool-Inventory Dataset (ETID).** ETID is a domain-specific tool-calling dataset designed to evaluate *argument correctness* under complex, enterprise-style schemas. It contains 124 tools with an average of 3.4 parameters per tool (maximum 12) and approximately 13 labeled examples per tool (minimum 10). We evaluate both an *all-tools* setting and *value-stream* subsets. To study data efficiency, we use intent-aligned regimes `Train-`$N$`ex`, where each tool contributes $N$ training and $N$ validation examples (for a total of $124 \times N$ examples per split), and report results for `Train-1ex`, `Train-2ex`, and `Train-4ex`. The test set is fixed at 404 queries. More details on dataset construction are provided in Section B and in the project repository[2].

**ToolACE (tool scaling).** ToolACE evaluates performance degradation as the tool universe expands. We use fixed splits (Train = 199, Validation = 76, Test = 121) and augment the tool inventory to create `ToolACE-300/500/750/1000` variants.

**SEAL-Tools (parallel multi-tool calling).** SEAL-Tools (Wu et al., 2024b) benchmarks *parallel* multi-tool calling across diverse domains. We use a curated `multiple-overlap` subset with 1,138 tools and 2,743 arguments, split into Train = 600, Validation = 100, Test = 100. Each query requires 3.2 parallel tool calls on average (typically 3), with 5.8 arguments filled per query, stressing joint multi-tool selection and argument filling.

We use a curated `multiple-overlap` subset with 1,138 tools and 2,743 arguments. The split is Train = 600, Validation = 100, and Test = 100 examples. Each query requires an average of 3.2 parallel tool calls (77% require exactly 3 tools) and 5.8 filled arguments. Tool overlap across splits ensures evaluation tools are seen during training. This setting isolates the challenge of *joint* tool selection and slot filling at scale.

[2] www.sites.google.com/view/jtpro-etid/home

| Dataset | #Tools | Total Args | | Required Args | |
|---|---|---|---|---|---|
| | | Avg | Max | Avg | Max |
| ETID | 124 | 3.4 | 12 | 0.81 | 6 |
| ToolACE-300 | 336 | 2.05 | 14 | 1.20 | 6 |
| ToolACE-500 | 536 | 2.14 | 17 | 1.20 | 7 |
| ToolACE-750 | 786 | 2.17 | 23 | 1.21 | 7 |
| ToolACE-1000 | 1036 | 2.10 | 23 | 1.21 | 7 |
| SEAL-Tools | 1138 | 2.41 | 8 | 1.60 | 5 |

Table 1: Dataset statistics. "#Tools" denotes the size of the tool universe available at inference time. "Total Args" counts all schema parameters per tool, and "Required Args" counts mandatory parameters.

### 5.2 Evaluation Metrics

Following prior tool-use evaluations, we measure call-level correctness rather than answer accuracy. Specifically, we report:

- **Tool Selection Accuracy (TSA)**: fraction of queries for which the agent chose the correct tool(s) required (including choosing none if no tool needed).
- **Slot Filling Accuracy (SFA)**: recall of correct slot/value assignments *conditional on correct tool selection*.
- **Overall Success Rate (OSR)**: (correct tool + correct slots + correct values).

This evaluation reflects practical deployments where executing the true tool backend may be infeasible (e.g., security constraints, access controls, rate limits, or non-deterministic systems), so correctness must be assessed at the tool-call level.

## 6 Results and Analysis

### 6.1 ToolACE: Scaling the Tool Universe

Table 2 reports Tool Selection Accuracy (TSA), Slot Filling Accuracy (SFA; conditional on correct tool), and Overall Success Rate (OSR; correct tool + correct slots + correct values) for ToolACE with 500 and 1000 tools.

As the tool inventory grows, baseline performance drops primarily in TSA, which cascades to lower OSR even when SFA remains high. GEPA improves TSA in most settings, but gains in OSR are limited because failures often stem from tool-specific disambiguation and argument constraints that global instruction refinement alone cannot resolve.

| Model | #Tools | TSA (%) | | | SFA (%) ↑ \| TSA | | | OSR (%) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Base | GEPA | JTPRO | Base | GEPA | JTPRO | Base | GEPA | JTPRO |
| GPT-4o mini | 500 | 63.832 | 73.33 | **75.25** | 86.996 | 85.27 | **88.12** | 60.00 | 61.98 | **69.42** |
| GPT-4o mini | 1000 | 61.115 | 73.39 | **75.13** | 86.67 | 83.36 | **83.59** | 58.18 | 60.33 | **63.64** |
| o3-mini | 500 | 70.78 | 73.33 | **76.19** | 84.994 | 85.27 | **88.52** | 59.454 | 61.98 | **65.29** |
| o3-mini | 1000 | 58.916 | 70.11 | **71.48** | 85.036 | 86.59 | **87.46** | 51.272 | 58.68 | **64.46** |
| GPT-5 | 500 | 73.02 | 77.17 | **82.28** | 84.785 | 85.75 | **90.00** | 62.73 | 66.12 | **74.38** |
| GPT-5 | 1000 | 67.658 | 75.13 | **78.72** | 87.35 | 86.40 | **89.26** | 62.366 | 67.77 | **73.55** |

Table 2: **ToolACE results under tool-universe scaling (500 vs. 1000 tools).** JTPRO achieves the strongest end-to-end performance (OSR) by jointly improving tool selection (TSA) and argument correctness (SFA), with the largest gains appearing in the 1000-tool regime where tool confusions are most frequent.

JTPRO consistently achieves the highest TSA and OSR across all models and tool counts. Gains are especially pronounced in the 1000-tool setting (e.g., +13.2 OSR points for `o3-mini` over baseline). While SFA is already strong, JTPRO further boosts end-to-end success by reducing tool confusions and encoding missing slot/value conventions. These results show that, on ToolACE, OSR improvements are primarily driven by better tool selection, emphasizing that accurate TSA is critical for downstream argument correctness.

### 6.2 ETID: Complex Slot Filling with Moderate Tool Counts

We evaluate on the *Enterprise Tool-Inventory Dataset (ETID)*, which features complex multi-argument schemas (avg. 3.4 parameters/intent; max 12) and measures correctness at the *call level* (tool + slots + values). Table 3 reports results under low-supervision regimes (Train-1/2/4 examples per intent; fixed test set of 404 queries).

Two trends stand out. First, **slot/value correctness is the main bottleneck**. Baseline TSA is high (85–94%), yet OSR remains much lower, showing that SFA errors dominate once the correct tool is chosen. JTPRO addresses this directly, improving SFA and boosting OSR—e.g., for `GPT-4o mini`, OSR rises from 44.8→60.15 (+15.35) in Train-1ex and 46.53→66.83 (+20.30) in Train-4ex, despite similar TSA.

Second, **JTPRO delivers robust gains across models and training regimes**. For `gpto3-mini`, OSR improves over both baseline and GEPA in all regimes, with larger gains as supervision increases (Train-4ex: 67.33→82.67). For `GPT-5`, GEPA raises TSA, but JTPRO achieves the highest OSR by combining strong tool selection with higher SFA (Train-2ex: SFA 92.77, OSR 85.64). Overall, ETID shows that optimizing tool selection alone is insufficient; joint refinement of instructions and tool/slot descriptions is necessary to convert high TSA into end-to-end success.

### 6.3 SEAL-Tools: Multi-Tool Calling

We evaluate on the SEAL-Tools multi-tool subset, which contains 1,138 tools and requires an average of 3.2 parallel tool calls per query. This makes both tool disambiguation and argument instantiation challenging, since success depends on selecting the correct tools and correctly filling their arguments across multiple calls.

Table 4 shows that JTPRO consistently improves SFA and OSR across all three models, while keeping TSA stable or slightly improving it. For **GPT-4o**, TSA changes only slightly from 81.0 to 82.3, while SFA rises from 40.4 to 53.51 and OSR from 23.0 to 27.5. For **o3-mini**, TSA improves from 82.2 to 83.9, with SFA increasing from 52.3 to 60.1 and OSR from 26.3 to 30.1. For **GPT-5**, TSA increases from 84.5 to 86.5, alongside gains in SFA from 56.3 to 65.2 and OSR from 28.8 to 33.6.

Overall, the results reinforce our central finding: in large, schema-rich multi-tool settings, improving tool selection alone is not enough. Joint refinement of agent instructions and per-tool schema descriptions is needed to translate strong TSA into better end-to-end success.

### 6.4 Instance-Level Slot Corrections and Tool Disambiguation

JTPRO improves slot/value instantiation at the example level while also making semantically similar tools easier to distinguish. Figure 12 shows that for GPT-5 on **ToolACE-500**, JTPRO corrects pre-

| Model | Train | TSA (%) | | | SFA (%) ↑ \| TSA | | | OSR (%) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Base | GEPA | JTPRO | Base | GEPA | JTPRO | Base | GEPA | JTPRO |
| GPT-4o mini | Train-1ex | 85.91 | 86.23 | 83.14 | 69.81 | 78.71 | **82.72** | 44.80 | 50.19 | **60.15** |
| GPT-4o mini | Train-2ex | 86.36 | 85.32 | 88.27 | 70.90 | 77.12 | **83.37** | 45.79 | 52.7 | **65.10** |
| GPT-4o mini | Train-4ex | 86.96 | 88.38 | 87.18 | 70.09 | 74.93 | **85.16** | 46.53 | 54.34 | **66.83** |
| o3-mini | Train-1ex | 94.00 | 95.08 | 95.78 | 80.48 | 83.73 | **89.77** | 68.81 | 75.00 | **79.46** |
| o3-mini | Train-2ex | 94.15 | 94.40 | 95.40 | 80.20 | 85.30 | **90.01** | 67.33 | 73.02 | **79.70** |
| o3-mini | Train-4ex | 94.41 | 94.90 | 95.95 | 79.60 | 87.455 | **90.98** | 67.325 | 77.725 | **82.67** |
| GPT-5 | Train-1ex | 94.06 | 97.27 | 95.76 | 82.45 | 90.21 | **92.15** | 68.81 | 80.20 | **84.65** |
| GPT-5 | Train-2ex | 94.03 | 97.91 | 95.80 | 83.90 | 89.89 | **92.77** | 71.53 | 79.18 | **85.64** |
| GPT-5 | Train-4ex | 94.16 | 98.47 | 96.81 | 83.29 | 88.80 | **92.30** | 70.54 | 80.69 | **85.15** |

Table 3: **ETID results under low-supervision training regimes.** JTPRO yields the most consistent OSR improvements, indicating that improved argument semantics are crucial for complex enterprise schemas.

| Model | TSA (%) | | | SFA (%) ↑ \| TSA | | | OSR (%) | | |
|---|---|---|---|---|---|---|---|---|---|
| | Base | GEPA | JTPRO | Base | GEPA | JTPRO | Base | GEPA | JTPRO |
| GPT-4o | 81 | 82.11 | 82.3 | 40.4 | 43.51 | **53.51** | 23 | 24.51 | **27.5** |
| o3-mini | 82.2 | 82.1 | 83.9 | 52.3 | 56.9 | **60.1** | 26.3 | 27.5 | **30.1** |
| GPT-5 | 84.5 | 85.2 | 86.5 | 56.3 | 61.3 | **65.2** | 28.8 | 31.1 | **33.6** |

Table 4: **SEAL-Tools results (multi-tool calling).** JTPRO consistently improves SFA and OSR across all models, while keeping TSA stable or slightly improved.

viously incorrect slot/value assignments on **26 of 121** test examples (**21.48%**), while on **ETID** (Figure 13) it improves slot correctness on **94 of 403** examples (**23.33%**). Because these gains are distributed across many test instances, they indicate that JTPRO's improvements are broad rather than driven by a small number of outliers.

Another key reason of these gains is improved tool description disambiguation. On ToolACE-500, JTPRO updated **11%** of tool descriptions (**55/500**) with explicit cues that better separate confusable tools. We quantify this effect using intra-group cosine similarity across **37** groups of semantically similar tools; the group with the largest improvement reduced similarity from **0.668** to **0.502** (**-16.6%**), indicating clearer differentiation.

## 7 Conclusion

We presented **Joint Tool–Prompt Reflective Optimization (JTPRO)**, a weight-free reflective context optimization framework for improving tool-calling reliability in trace-supervised settings by jointly refining global agent instructions and per-tool schema/argument descriptions from rollout-driven feedback. JTPRO targets the two dominant failure modes in large, domain-specific tool inventories: tool mis-selection and argument mis-instantiation. The method uses reflective diagnostics to produce localized edits, maintains a candidate pool with Pareto-style selection to preserve diverse effective behaviors, and prevents context bloat by *globalizing* recurring slot semantics into the instruction layer while retaining tool-specific disambiguation cues in local schemas. Across ToolACE tool-scaling experiments, ETID enterprise slot-filling tasks, and SEAL-Tools multi-tool calling, JTPRO improves tool selection, slot filling, and overall success relative to strong baselines including reflective prompt optimizers (e.g., GEPA), highlighting that accurate argument semantics are necessary to translate high tool selection accuracy into end-to-end tool-use success. Overall, these results position JTPRO as a practical reflective optimization approach for jointly adapting multiple agent operating components, global instructions and tool schemas, to improve reliable tool invocation under evolving tool inventories without model fine-tuning.

## Limitations

Our study has limitations that motivate future work. First, our experimental scope is centered on trace-supervised tool-calling reliability. We evaluate (i) single-tool, single-slot/value cases and (ii) multi-tool *parallel* calling with single-slot/value instantiation; we do not evaluate *sequential* multi-tool workflows that require multi-step dependencies, intermediate state, or long-horizon planning (e.g., tool chains where earlier outputs condition later calls). Extending JTPRO to such settings will require modeling stepwise credit assignment across tool sequences and validating robustness under longer rollouts.

Second, while ETID captures complex multi-argument schemas, our current evaluation does not systematically stress *deeply nested* argument structures (e.g., multi-layer JSON objects, lists-of-objects with constraints, or schema-dependent composition rules) at scale; future benchmarks should include nested-slot correctness and structure-aware metrics beyond scalar slot/value matching.

Third, our evaluation focuses on call-level correctness (tool, slots, values) in trace-supervised settings rather than executing tools and verifying response-level correctness; when tool execution is available, future experiments should extend JTPRO to end-to-end evaluation that includes tool responses and downstream post-processing logic (e.g., response parsing, aggregation, and business-rule enforcement), since these components can introduce additional failure modes beyond argument correctness.

Fourth, ETID is currently not publicly released, which limits external reproducibility and benchmarking by the community.

Finally, our empirical study is limited to 3 benchmarks; future work should broaden coverage to additional public and proprietary tool-use datasets spanning more domains, tool granularity, and interaction styles, to better characterize generalization under diverse tool inventories and schema conventions.

## References


Amit Agarwal, Hansa Meghwani, Hitesh Laxmichand Patel, Tao Sheng, Sujith Ravi, and Dan Roth. 2025. Aligning LLMs for multilingual consistency in enterprise applications. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 117–137, Suzhou (China). Association for Computational Linguistics.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2025. Gepa: Reflective prompt evolution can outperform reinforcement learning. *Preprint*, arXiv:2507.19457.

Lisa Alazraki and Marek Rei. 2025. Meta-reasoning improves tool use in large language models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, page 7885–7897. Association for Computational Linguistics.

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

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2024. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*.

Yupu Hao, Pengfei Cao, Zhuoran Jin, Huanxuan Liao, Yubo Chen, Kang Liu, and Jun Zhao. 2025. Citi: enhancing tool utilizing ability in large language models without sacrificing general performance. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. Dspy: Compiling declarative language model calls into self-improving pipelines.

Yoonho Lee, Joseph Boen, and Chelsea Finn. 2025. Feedback descent: Open-ended text optimization via pairwise comparison. *Preprint*, arXiv:2511.07919.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Mosh Levy, Alon Jacoby, and Yoav Goldberg. 2024. Same task, more tokens: the impact of input length on the reasoning performance of large language models.

Hao Liu, Zi-Yi Dou, Yixin Wang, Nanyun Peng, and Yisong Yue. 2024. Uncertainty calibration for tool-using language agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 16781–16805, Miami, Florida, USA. Association for Computational Linguistics.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, and 8 others. 2025. Toolace: Winning the points of llm function calling. *Preprint*, arXiv:2409.00920.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: iterative refinement with self-feedback. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Hansa Meghwani, Amit Agarwal, Priyaranjan Pattnayak, Hitesh Laxmichand Patel, and Srikant Panda. 2025. Hard negative mining for domain-specific retrieval in enterprise systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 1013–1026.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9340–9366, Miami, Florida, USA. Association for Computational Linguistics.

Priyaranjan Pattnayak, Amit Agarwal, Hansa Meghwani, Hitesh Laxmichand Patel, and Srikant Panda. 2025. Hybrid ai for responsive multi-turn online conversations with novel dynamic routing and feedback adaptation. In *Proceedings of the 4th International Workshop on Knowledge-Augmented Methods for Natural Language Processing*, pages 215–229.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7957–7968, Singapore. Association for Computational Linguistics.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. Toolrl: Reward is all tool learning needs. *Preprint*, arXiv:2504.13958.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, and 22 others. 2024. Tool learning with foundation models. *Preprint*, arXiv:2304.08354.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Towards completeness-oriented tool retrieval for large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, page 1930–1940. ACM.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2025. From exploration to mastery: Enabling llms to master tools via self-driven interactions.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Preprint*, arXiv:2303.17580.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online. Association for Computational Linguistics.

Jyotika Singh. 2023. *Natural Language Processing in the Real World: Text Processing, Analytics, and Classification*. Chapman and Hall/CRC.

Jyotika Singh, Weiyi Sun, Amit Agarwal, Viji Krishnamurthy, Yassine Benajiba, Sujith Ravi, and Dan Roth. 2025. Can LLMs narrate tabular data? an evaluation framework for natural language representations of text-to-SQL system outputs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 883–902, Suzhou (China). Association for Computational Linguistics.

Jyotika Singh, Fang Tu, Miguel Ballesteros, Weiyi Sun, Sandip Ghoshal, Michelle Yuan, Yassine Benajiba, Sujith Ravi, and Dan Roth. 2026. Mt-osc: Path for llms that get lost in multi-turn conversation. *Preprint*, arXiv:2604.08782.

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, Ye Tian, and Sujian Li. 2023. Restgpt: Connecting large language models with real-world restful apis. *Preprint*, arXiv:2306.06624.

Claudio Spiess, Mandana Vaziri, Louis Mandel, and Martin Hirzel. 2025. AutoPDL: Automatic Prompt Optimization for LLM Agents. *arXiv e-prints*, arXiv:2504.04365.

Michael Sullivan, Mareike Hartmann, and Alexander Koller. 2025. Procedural environment generation for tool-use agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 18555–18573, Suzhou, China. Association for Computational Linguistics.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. 2025. Dynamic cheatsheet: Test-time learning with adaptive memory. *Preprint*, arXiv:2504.07952.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6).

Wenxiao Wang, Priyatham Kattakinda, and Soheil Feizi. 2025. Maestro: Joint graph & config optimization for reliable ai agents. *Preprint*, arXiv:2509.04642.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Georg Wölflein, Dyke Ferber, Daniel Truhn, Ognjen Arandjelovic, and Jakob Nikolas Kather. 2025. LLM agents making agent tools. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 26092–26130, Vienna, Austria. Association for Computational Linguistics.

Bin Wu, Edgar Meij, and Emine Yilmaz. 2025. A joint optimization framework for enhancing efficiency of tool utilization in LLM agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22361–22373, Vienna, Austria. Association for Computational Linguistics.

Jiayi Wu, Renyu Zhu, Nuo Chen, Qiushi Sun, Xiang Li, and Ming Gao. 2024a. Structure-aware fine-tuning for code pre-trained models. *Preprint*, arXiv:2404.07471.

Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. 2024b. Seal-tools: Self-instruct tool learning dataset for agent tuning and detailed benchmark. *arXiv preprint arXiv:2405.08355*.

Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis N. Ioannidis, Karthik Subbian, Jure Leskovec, and James Zou. 2024c. Avatar: Optimizing LLM agents for tool usage via contrastive reasoning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. 2023. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *Preprint*, arXiv:2305.18323.

Qiancheng Xu, Yongqi Li, Heming Xia, and Wenjie Li. 2024. Enhancing tool retrieval with iterative feedback from large language models.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023a. React: Synergizing reasoning and acting in language models.

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

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R. Fung, Hao Peng, and Heng Ji. 2024. Craft: Customizing llms by creating and retrieving from specialized toolsets. *Preprint*, arXiv:2309.17428.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. 2025. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639:609–616.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. 2025. Agentic context engineering: Evolving contexts for self-improving language models. *Preprint*, arXiv:2510.04618.

Sida Zhang. 2024. Agentic ai across domains: A comprehensive review of capabilities, applications, and future directions. *Journal of Computing Innovations and Applications*, 2:86–98.

Caleb Zheng, Jyotika Singh, Fang Tu, Weiyi Sun, Sujeeth Bharadwaj, Yassine Benajiba, Sujith Ravi, Eli Shlizerman, and Dan Roth. 2026. Diffumask: Diffusion language model for token-level prompt pruning. *Preprint*, arXiv:2604.06627.

Yuanhang Zheng, Peng Li, Wei Liu, Yang Liu, Jian Luan, and Bin Wang. 2024. Toolrerank: Adaptive and hierarchy-aware reranking for tool retrieval. *Preprint*, arXiv:2403.06551.

# A Method prompts and details

## A.1 Update Global Instructions and Tool Revisions Prompt

**Prompt Template: Propose Updated Global Instructions and Tool Revisions (JT-PRO Reflector)**

*Goal: produce clean, minimal updates to the global instructions and only the tools that require revision, based on the feedback trace.*

**Task.** Update the global instructions and tool descriptions using the feedback on the current context.

**Current Global Instructions**
`<curr_instructions>`

**Current Tool Definitions (Full List)**
`<tools_list_to_update>`

**Objective.** Produce:

- `global_instructions`: updated system-level guidance.
- `example_specific_instructions`: batch-specific guidance to append to prior examples/hints.
- `tool_revisions`: only the tools you modified (not the entire tool list).

**Feedback Trace**
`<dataset_with_feedback>`
For each example, the feedback indicates whether the model:

- failed to call a tool that should have been called, or
- called the wrong tool instead of the correct one, or
- selected the correct tool but produced incorrect `action_inputs` (missing/wrong parameters), or
- selected the correct parameters but assigned incorrect slot names/values (formatting/value errors).

**Instructions Revision Rules (Important)**

- The instructions may already contain prior revisions and examples.
- Always preserve previously incorporated `example_specific_instructions` and example hints; do not alter them.
- Modify existing `example_specific_instructions` only if there is a direct conflict with the current feedback.
- Add new guidance as bullet points appended to the existing list under `example_specific_instructions`.

**Tool Revision Rules (Important)**

- If an answer is marked wrong, check two cases:

- **Case 1 (Model error):** If a tool-selection error occurred (a tool should have been chosen but was not, or was chosen but should not have been), you *must* revise the relevant tool description(s) to make correct usage clearer. Otherwise, you may leave the tool unchanged.
- **Case 2 (Documentation/data issue):** Tool documentation may be ambiguous or incomplete, and ground-truth traces may contain incorrect tool arguments or values. If you detect such issues, revise the tool documentation to remove ambiguity so future runs avoid the same failure.

- Return only the tools you modified from the provided tool list.

**Output Format (Strict)**

- Return a single JSON object immediately after the literal text `Answer:`
- Do not add any extra text before or after the JSON.

**Required JSON Schema**

```
{
"global_instructions": "UPDATED
GLOBAL INSTRUCTIONS HERE",
"example_specific_instructions":
"UPDATED INSTRUCTIONS AS PER THE
CURRENT BATCH",
"tool_revisions": [
{
"name": "<tool1_name>",
"description":
"<tool1_description>",
"parameters": {
"type": "dict",
"properties": {
"<property1>":
{"description": "<updated
tool1_property1_description>",
"type": "<property1_type; same as
original>"},
"<property2>":
{"description": "<updated
tool1_property2_description>",
"type": "<property2_type; same as
original>"}
},
"required": ["<property1;
required parameters; identical to
original>"]
},
"required": null
}
]
}
```

**Final constraint:** `Answer:` must be followed by *only* the JSON object.

Figure 5: **JTPRO reflector prompt for proposing new global instructions and tool candidates.** The reflector updates system-level guidance and selectively revises only the implicated tools/slots based on rollout feedback, while preserving prior batch-specific examples and enforcing a strict JSON-only output format.

**Prompt Template: Merge Draft Instructions with the Global Best (MergeWithBest)**

**Role.** You are the *instruction merger* in agent. Your job is to combine a draft update with the current best global instructions into a single improved instruction prompt.

**Inputs**

- **Global best instructions** ($P^\star$): `<best_global_instructions>`
- **Draft instructions from current rollout** ($P^d$): `<draft_global_instructions>`
- **(Optional) Newly added tools since** $P^\star$: `<new_tools_summary>` (Names + 1–2 lines per tool describing the new capability.)

**Objective.** Produce merged instructions $P'$ that:

- preserve stable, broadly useful guidance from $P^\star$ (the "growing playbook"),
- incorporate *new* and *validated* guidance from $P^d$ *additively*,
- remain concise and non-redundant (avoid restating the same rule twice),
- support incremental toolset growth: keep cross-cutting rules stable while adding any new decision rules introduced by newly appended tools.

**Merge Rules (Strict)**

- **Do not overwrite**: never delete a rule from $P^\star$ unless $P^d$ provides a clearly conflicting correction.
- **Prefer generalization**: if $P^d$ adds a rule that generalizes an existing one, keep the generalized version and remove the narrower duplicate.
- **Resolve conflicts explicitly**: if a draft rule contradicts an existing rule, keep the version that is more precise and operational (clear triggers, clear expected behavior), and remove the other.
- **Tool-growth compatibility**: if a draft rule is specific to a newly added tool, include it only if it can be stated as a general decision rule (when-to-use / how-to-fill), otherwise keep it minimal and non-invasive.
- **No tool merging**: do not rename, alias, or merge tools; only adjust global instruction text.

**Output Format (Strict)**

- Return *only* the merged global instructions $P'$ as plain text.
- No JSON. No commentary. No additional sections.

Figure 6: **MergeWithBest prompt.** The merger composes rollout-specific draft instructions $P^d$ with the current global best $P^\star$ to form $P'$, preserving stable cross-cutting guidance while integrating validated new rules. This "growing playbook" mechanism supports incremental toolset expansion by keeping existing conventions stable and appending new decision rules when new tools are introduced.

**Prompt Template: Slot-Semantics Globalization (Two-Level Context Editing)**

**Role.** You are a context editor for a tool-using LLM agent. You may revise (i) *Global Instructions* $P$ and (ii) per-tool schemas $\{T_i\}$ (tool and argument descriptions).
**Objective.** Reduce repeated slot/argument guidance across tools while preserving tool-specific distinctions needed for correct tool selection and slot filling.

**Step 1: Scan for repeated slot semantics.** Read each tool schema $T_i$ and its argument descriptions carefully. Identify *recurring* slot conventions that appear across many tools, such as: date/time windows, identifier formatting, numeric bounds (inclusive/exclusive), units/currency normalization, boolean/defaulting rules, pagination parameters, and sorting conventions.

**Step 2: Globalize shared rules.** For each repeated convention, write a *single*, canonical rule in the Global Instructions $P$ that: (a) states the default interpretation and formatting requirements, and (b) specifies when to apply default values versus using user-provided constraints. The global rule should be phrased generically so it applies to any tool that contains the relevant slot(s).

**Step 3: Keep exceptions local.** Do *not* merge, alias, or rename tools. For each tool schema $T_i$:

- Remove redundant restatements of globalized rules and replace them with a short pointer (e.g., "See Global Instructions: *[Rule Name]*" ).
- If a tool requires different semantics (e.g., a different date format, special rounding, a stricter constraint), keep that information *locally* in $T_i$ and explicitly mark it as an **override** of the global rule.

**Constraints.**

- Do not change tool interfaces: do **not** add/remove arguments or invent fields.
- Prefer minimal, high-impact edits: globalize only clearly repetitive conventions; keep tool-unique decision rules and edge cases local.

**Output.**

- An updated Global Instructions block to append to $P$ (named rules + concise definitions).
- Updated schemas for only the tools you modified (short pointers + explicit overrides).

## B Enterprise Tool-Inventory Dataset (ETID): Synthesis Methodology

**Design objective.** ETID can be synthesized as a privacy-preserving benchmark for tool use in realistic enterprise settings, where agents must operate over a large catalog of domain-specific tools with heterogeneous schemas, overlapping capabilities, and multi-argument interfaces. Rather than relying on proprietary traces, the dataset can be constructed from an abstracted enterprise tool inventory that preserves only the structural properties needed for evaluation: tool granularity, schema complexity, required/optional argument patterns, and regions of semantic overlap.

**Constructing an enterprise-style tool inventory.** A practical synthesis recipe begins by assembling tools from multiple enterprise workflows, for example, search, analytics, reporting, inventory, finance, support, scheduling, and operations. Each tool is represented by a name, a concise description, and a typed parameter schema. The inventory should be synthesized to exhibit a realistic argument distribution, with many tools requiring only a small number of fields and a meaningful long tail of more complex schemas. To preserve realistic routing failures, semantically adjacent tools should remain distinct rather than being merged or aliased; instead, they should differ in scope, triggering conditions, or argument constraints so that tool disambiguation remains a central challenge.

**Injecting overlapping field semantics.** To reproduce the schema redundancy typical of enterprise tool stacks, the synthesis process should explicitly reuse recurring slot families across many tools. As discussed in the global slot-semantics analysis, these families include identifier fields, date/time windows, numeric bounds, boolean flags, sorting controls, and currency/unit parameters. One effective procedure is to define a library of canonical slot families and instantiate them repeatedly across tools with minor naming variation and tool-specific overrides. For example, many tools may expose fields analogous to `startDate`, `endDate`, `rangeMinimum`, `rangeMaximum`, and associated inclusivity flags, while retaining local fields such as item names, location names, account identifiers, or business-specific selectors. This creates the heavy-tailed overlap pattern that motivates globalizing shared semantics while keeping tool-specific exceptions local.

**Generating user requests and gold traces.** Given the synthesized inventory, natural-language requests can be generated by sampling intents per tool and paraphrasing them across diverse linguistic forms. To make the benchmark challenging, the generator should include (i) direct requests, (ii) ambiguous requests that could plausibly match multiple nearby tools, and (iii) requests that require normalization of dates, identifiers, numeric ranges, booleans, or units. Gold traces are then constructed as structured tool calls with the correct tool and fully specified arguments, including canonical formatting and defaulting rules for recurring slot families.

**Privacy and validation.** Because the benchmark is synthetic, all values can be produced from controlled non-sensitive templates or vocabularies, with explicit exclusion of personally identifiable information (PII), proprietary identifiers, and real customer artifacts. A final validation stage should verify schema consistency, argument-type correctness, intended ambiguity among overlapping tools, and coverage across train, validation, and test splits. Low-shot regimes can then be created by allocating a small number of labeled examples per tool while maintaining broad tool coverage in evaluation.

# C Experimental Setup

## C.1 Benchmark Suitability and Scope of Evaluation

Our benchmark selection is guided by the core design assumption of JTPRO: prompt and schema refinements should be learned over a reusable tool set with stable schema semantics, so that improvements transfer across train, validation, and test splits rather than overfitting to individual instances. This makes benchmarks with persistent tools and large schema-rich tool inventories especially suitable for evaluating our method.

**Why we do not use BFCL.** BFCL defines tools on a per-instance basis rather than as a single consistent tool inventory. In practice, we observed cases where the same tool name appears with different parameter definitions across examples. This makes it difficult to construct the stable, reusable tool universe required by JTPRO without substantially modifying the dataset. For this reason, we considered BFCL unsuitable for our study setting.

**Why $\tau$-bench is only partially aligned.** $\tau$-bench is closer to our setting, but it uses relatively small tool inventories (e.g., 15 APIs in Retail and 13 in Airline). By contrast, our main claim concerns settings with large toolsets, where tool selection and schema-constrained slot filling become the dominant sources of failure. We therefore prioritize benchmarks that more directly stress this scaling regime.

**Scope with respect to sequential planning.** Our current evaluation focuses on single-tool, parallel multi-tool, and schema-constrained argument instantiation settings. It does not yet evaluate long-horizon sequential workflows in which the output of one tool determines the choice or arguments of a subsequent tool. We view this as an important next step, but also as a distinct source of difficulty from the large-inventory tool selection and slot-filling problems targeted by JTPRO. Extending the framework to sequential planning benchmarks is a natural direction for future work.

## C.2 Evaluation protocol and reporting.

To reduce variance from stochastic decoding and optimization dynamics, all results are aggregated over multiple independent runs. Unless otherwise stated, each experiment is repeated **5–10 times** and we report the **average** performance across runs.

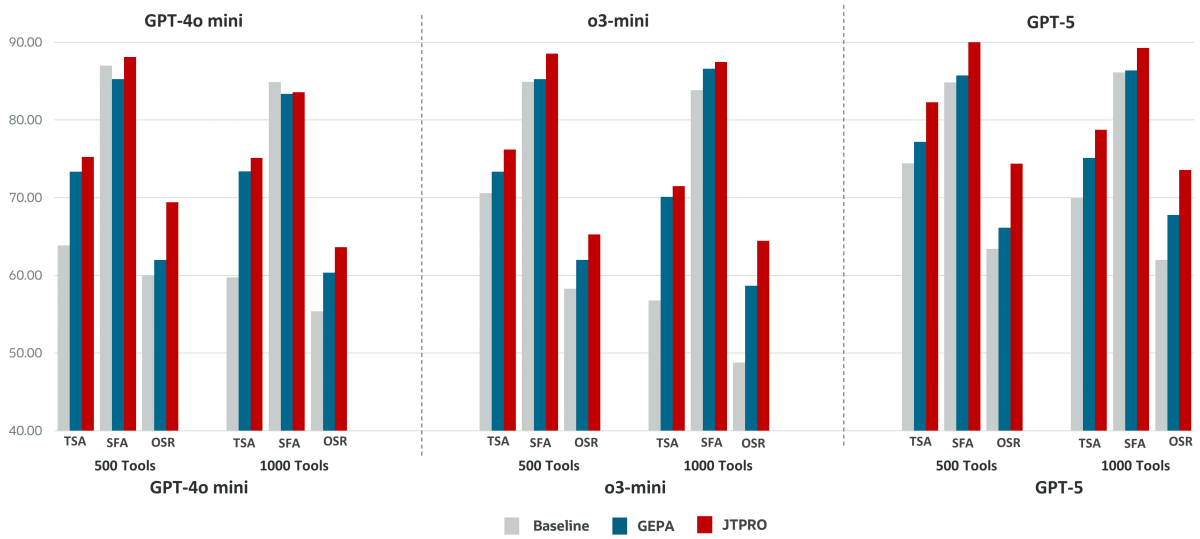


Figure 7: **ToolACE scaling results across models and metrics.** For each model, we report TSA, SFA (conditional on correct tool), and OSR at 500 and 1000 tools. Tool-universe growth primarily reduces TSA for the baseline, which cascades to lower OSR; GEPA partially mitigates this via global instruction refinement, while JTPRO provides the most consistent improvements in OSR by jointly refining global instructions and tool/slot descriptions.

For fair comparison, we run **JTPRO and GEPA under matched optimization budgets**: both methods use the same maximum number of rollouts and identical optimization settings (including the same minibatch size and the same reflector configuration). Across all experiments, we use **o3-mini** as the reflector model and set the **LLM temperature parameter to 1**.

## D Additional Figure Discussion

### D.1 Repetitive slot semantics across tools

Figures 9 and 10 motivate GLOBALIZESLOTS. Figure 9 shows a heavy-tailed distribution of parameter families across the tool inventory: a small number of recurring semantic classes, especially identifier and date/time fields, appear across a large fraction of tools (up to 77/124), while numeric bounds, boolean flags, sorting parameters, and related fields also recur repeatedly. This pattern indicates that many tools restate nearly identical guidance for formatting and interpretation, such as ISO-8601 date handling, inclusive versus exclusive numeric bounds, defaulting behavior, boolean semantics, and currency normalization.

Figure 10 shows how JTPRO converts this redundancy into a compact global rule layer. Instead of repeating the same instructions inside each tool schema, JTPRO lifts shared conventions into named global rules such as *DateTime Fields*, *Numeric Bounds*, *Boolean Parameters*, *Sorting Conventions*, and *Currency and Units*, and replaces duplicated local text with short pointers to those rules. In the illustrated `ItemOverageRequest` schema, for example, `startDate` and `endDate` now refer to the global date/time rule, `rangeMinimum` and `rangeMaximum` refer to the global numeric-bounds rule, and `rangeMinimumInclusive` and `rangeMaximumInclusive` refer to the global boolean rule, while tool-specific fields such as `locationName` and `itemName` remain local. This two-level organization improves slot filling by enforcing consistent semantics across tools, reducing duplicated and potentially conflicting wording, and preserving schema space for the tool-specific exceptions and disambiguation cues that matter for correct invocation.

### D.2 Figure 7: ToolACE scaling results

Figure 7 evaluates robustness under tool-universe growth (500 vs. 1000 tools). The dominant failure mode under scaling is reduced TSA: as inventories expand, overlapping tool descriptions and increased distractors cause more routing errors, which then cascade into lower OSR. GEPA partially mitigates this via global instruction refinement, but it does not directly repair tool-local ambiguity or slot semantics. JTPRO delivers the most consistent OSR gains because it **jointly** revises global policies *and* the specific tool/slot descriptions implicated by observed failures, improving both selection and downstream argument correct-

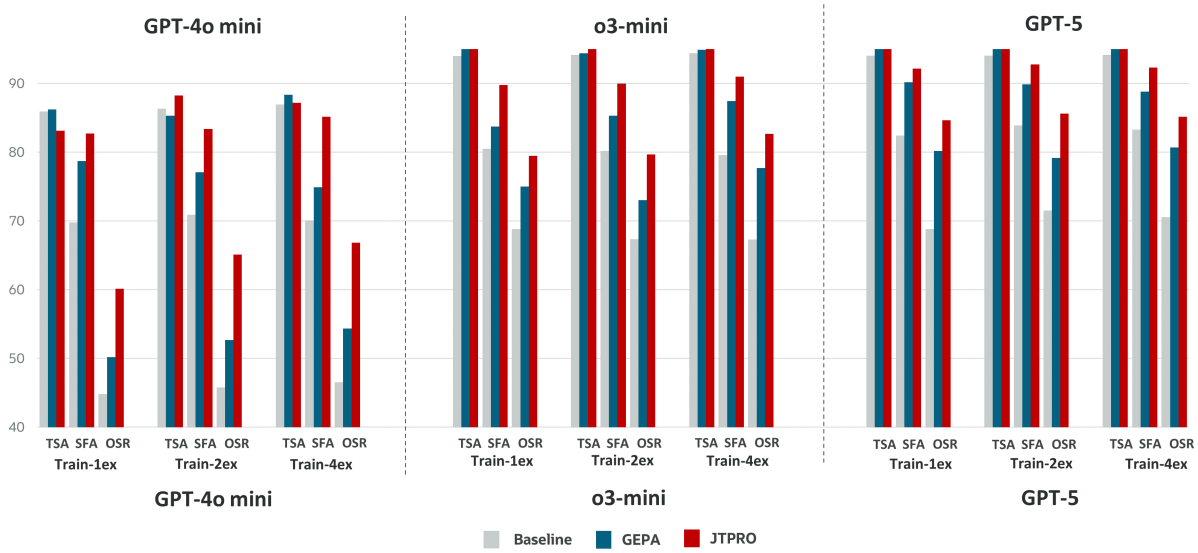


Figure 8: **ETID performance across supervision levels.** Grouped bars show TSA, SFA (conditional on TSA), and OSR for three models under Train-1ex/2ex/4ex regimes. Baselines achieve high TSA but substantially lower OSR, revealing slot/value errors as the dominant failure mode; JTPRO improves SFA and therefore OSR consistently across regimes, while GEPA primarily improves TSA for larger models.

ness.

### D.3 Figure 8: ETID performance across supervision levels

Figure 8 studies data efficiency on ETID under Train-1ex/2ex/4ex regimes. Across models, baselines often achieve relatively strong TSA but substantially lower OSR, indicating that **slot/value instantiation** is the primary bottleneck under complex schemas. JTPRO improves SFA (conditional on TSA) and therefore consistently lifts OSR across supervision levels, reflecting that many ETID failures stem from underspecified or inconsistent argument semantics that can be corrected through targeted tool/slot documentation edits plus strengthened global tool-calling rules.

### D.4 Figure 11: Per-example slot-filling improvement rate

Figure 11 reports the fraction of test instances for which JTPRO improves per-query slot/value correctness over the baseline. The gains are larger on ETID (complex schemas) than on ToolACE, aligning with the hypothesis that real-world OSR is often bottlenecked by argument instantiation even after correct tool selection. Importantly, the improvements occur on a non-trivial fraction of held-out examples across all evaluated models, suggesting that joint context refinement yields robust, example-level corrections rather than isolated wins.

### D.5 Figure 12: ToolACE-500 example-wise corrections (GPT-5)

Figure 12 provides an example-wise view of slot/-value corrections on ToolACE-500 for GPT-5. Each bar corresponds to a test instance, contrasting baseline vs. JTPRO slot correctness. Overall, JTPRO improves slot correctness on **26/121** examples (**21.48%**). This plot highlights that improvements are distributed across the test set (rather than concentrated in a single cluster), consistent with JTPRO correcting recurring slot conventions and tool-specific documentation ambiguities that manifest in diverse queries.

### D.6 Figure 13: ETID example-wise corrections (GPT-5)

Figure 13 shows the analogous example-wise comparison for ETID (GPT-5). JTPRO improves slot correctness on **94/403** examples (**23.33%**), reinforcing that complex multi-argument schemas benefit strongly from (i) tightening global tool-calling policies (e.g., required-field completeness, no hallucinated keys) and (ii) clarifying per-tool parameter semantics. Together with Figures 8 and 11, this example-level view supports the claim that SFA is a major driver of end-to-end OSR gains on ETID.

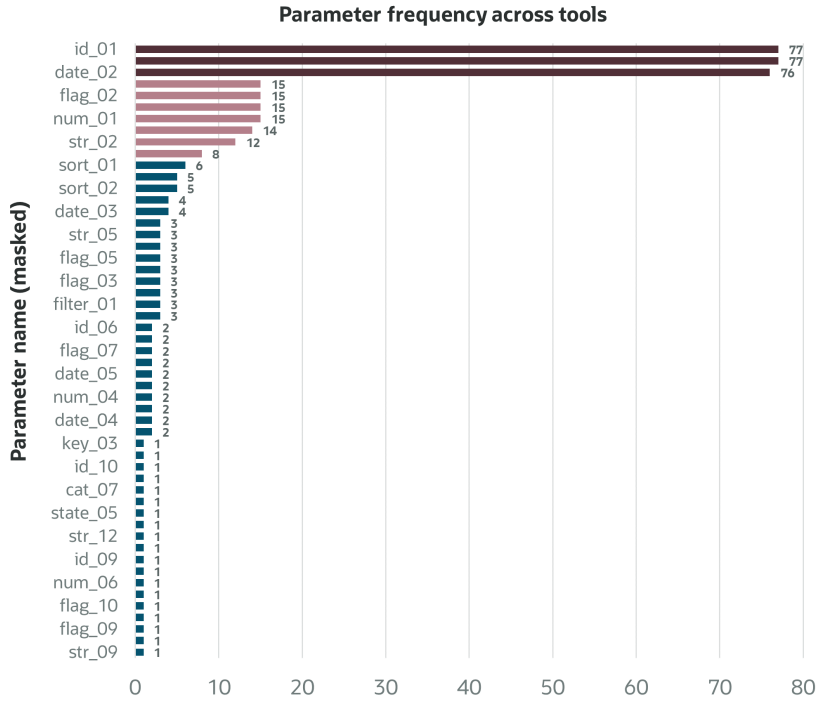


Figure 9: Parameter frequency across tools. The distribution is heavy-tailed: a small number of parameter families, especially identifier and date/time fields, recur across a large fraction of the tool inventory (up to 77/124 tools), while numeric, boolean, sorting, and related fields also appear repeatedly. This motivates lifting shared slot semantics into a global instruction layer rather than restating them in each tool schema.

**Global Rules:**

1. DateTime Fields: All date/time values (fields such as startDate, endDate, asOfDate, etc.) must follow the ISO 8601 standard. Date-time fields use the format 'YYYY-MM-DDTHH:MM:SSZ'. When a field is left blank, the system context default applies unless explicitly overridden locally.
2. Numeric Bounds: Any numeric parameters named rangeMinimum, rangeMaximum, and their respective inclusive flags (rangeMinimumInclusive, rangeMaximumInclusive) are to be interpreted using the provided numeric examples. The inclusive flags indicate whether the endpoint is included. See Global Instructions: Numeric Bounds for details.
3. Boolean Parameters: All boolean fields (e.g., isLate, isPicked) must be interpreted as true/false. Default values apply when provided or when not explicitly set in the query.
4. Sorting Conventions: Sorting parameters (such as prioritySorting, valueSorting, and limitSorting) must adhere to the enumerated values specified. When no override is provided, the default enumerated value should be used. See Global Instructions: Sorting Conventions for guidance.
5. Currency and Units: Monetary values must use standardized 3-letter ISO currency codes (e.g., USD, EUR, GBP).


```
{
    "name": "ItemOverageRequest",
    "description": "Fetch detailed information on inventory overages linked to a specific item and location. (Updated: Redundant date/time and numeric bound instructions have been replaced by pointers to Global Instructions.)",
    "parameters": {
      "type": "object",
      "properties": {
        "rangeMaximum": {
          "type": "integer",
          "description": "Numeric bound for maximum overage amount. See Global Instructions: Numeric Bounds. Example: 500"
        },
        "rangeMaximumInclusive": {
          "type": "boolean",
          "description": "Boolean flag for inclusive maximum. See Global Instructions: Boolean Parameters."
        },
        "rangeMinimum": {
          "type": "integer",
          "description": "Numeric bound for minimum overage amount. See Global Instructions: Numeric Bounds. Example: 500"
        },
        "rangeMinimumInclusive": {
          "type": "boolean",
          "description": "Boolean flag for inclusive minimum. See Global Instructions: Boolean Parameters."
        },
        "startDate": {
          "type": "string",
          "format": "date-time",
          "description": "Start date in ISO 8601 format. See Global Instructions: DateTime Fields. (Override: if a default value is provided, it must be used.)"
        },
        "endDate": {
          "type": "string",
          "format": "date-time",
          "description": "End date in ISO 8601 format. See Global Instructions: DateTime Fields. (Override: if a default value is provided, it must be used.)"
        },
        "locationName": {
          "type": "string",
          "description": "The location for which the item overage is queried. Example: New York Warehouse"
        },
        "itemName": {
          "type": "string",
          "description": "The name of the item for which the overage is queried. Example: Adult Bicycle, Three Wheel Bicycle"
        }
      }
    }
  },
```


Figure 10: Example of globalizing slot semantics. JTPRO moves repeated guidance for date/time formatting, numeric bounds, boolean interpretation, sorting conventions, and currency normalization into named global rules, and replaces redundant tool-local descriptions with short pointers to those rules. In the example `ItemOverageRequest` schema, shared fields such as `startDate`, `endDate`, `rangeMinimum`, `rangeMaximum`, and their inclusive flags reference global instructions, while tool-specific fields and local overrides remain in the tool schema.

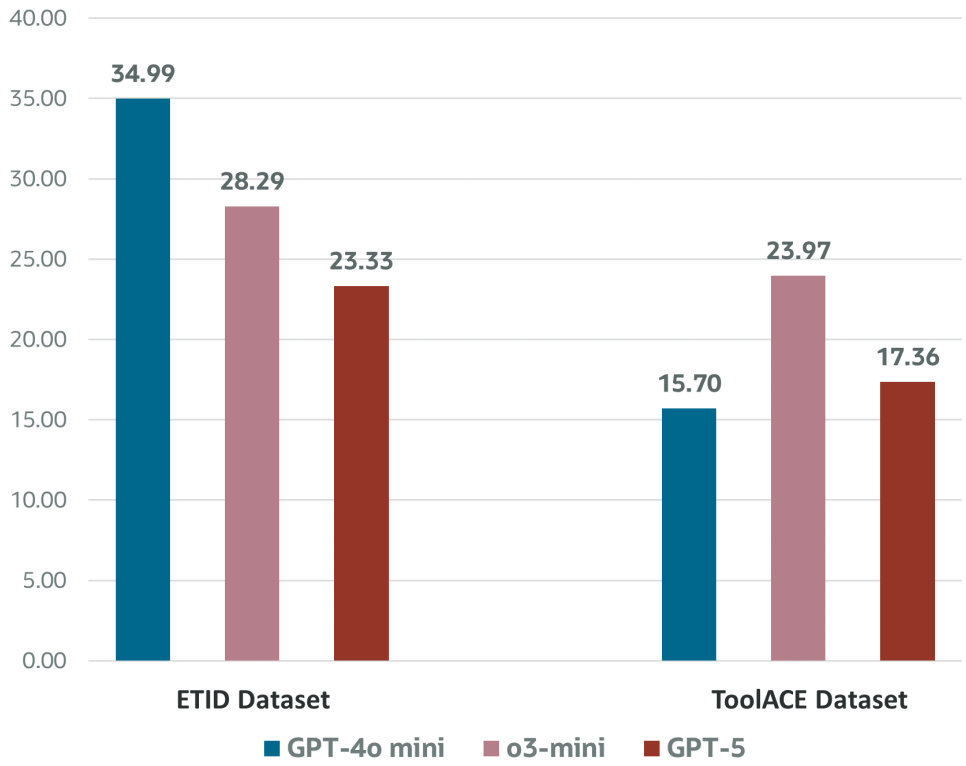


Figure 11: **Per-example slot-filling improvements from JTPRO.** For each base model and dataset, we report the *average percentage of test instances* on which slot filling is more accurate after JTPRO optimization than the corresponding baseline (i.e., per-query slot/value correctness improves). Gains are larger on the complex slot-filling ETID benchmark (e.g., 34.99% for GPT-4o mini) and remain substantial on ToolACE (e.g., 23.97% for o3-mini), indicating that JTPRO improves argument instantiation on a non-trivial fraction of held-out queries across models.

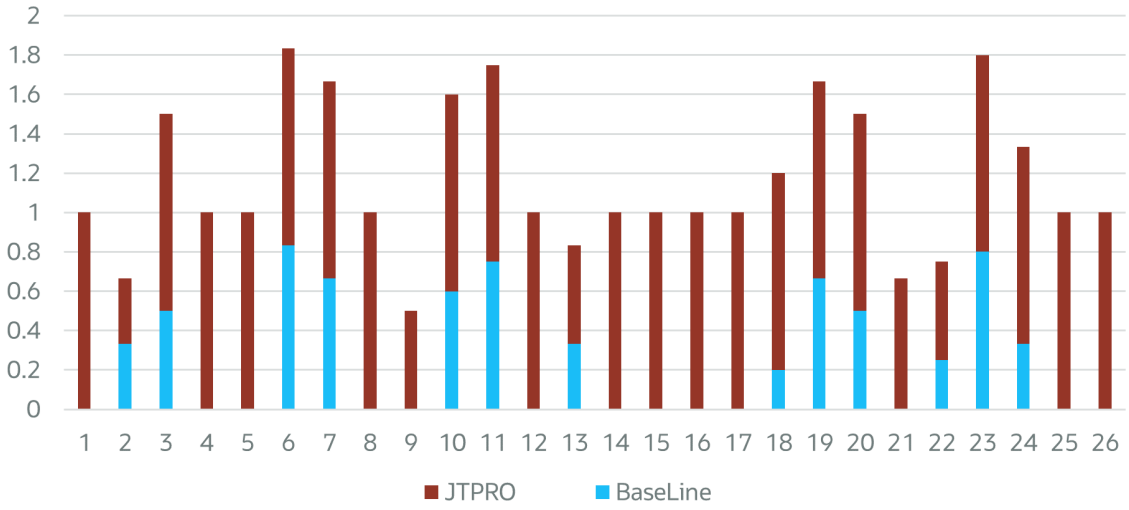


Figure 12: **Per-example slot/value corrections after JTPRO (ToolACE-500, GPT-5).** Example-wise comparison of slot-filling outcomes on the ToolACE test set with 500 tools for GPT-5: each bar corresponds to a test instance (x-axis indices), highlighting instances where JTPRO fixes previously incorrect slot/value instantiations relative to the baseline. Overall, JTPRO improves slot correctness on **26 out of 121** test examples (**21.48%**).

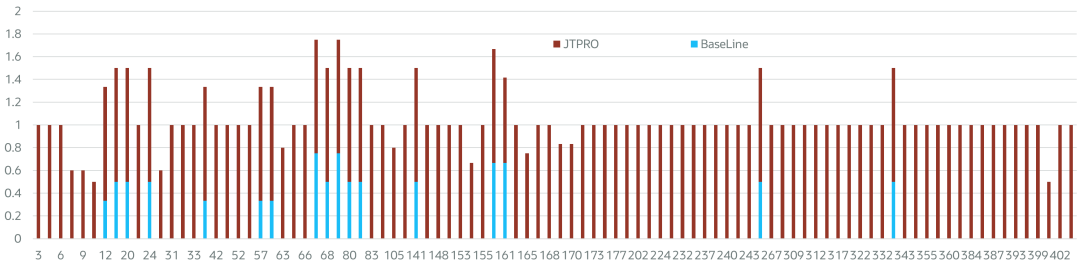


Figure 13: **Per-example slot/value corrections after JTPRO (ETID, GPT-5).** Example-wise comparison of slot-filling outcomes on the ETID test set for GPT-5: each bar corresponds to a test instance (x-axis indices), highlighting instances where JTPRO fixes previously incorrect slot/value instantiations relative to the baseline. Overall, JTPRO improves slot correctness on **94 out of 403** test examples (**23.33%**).

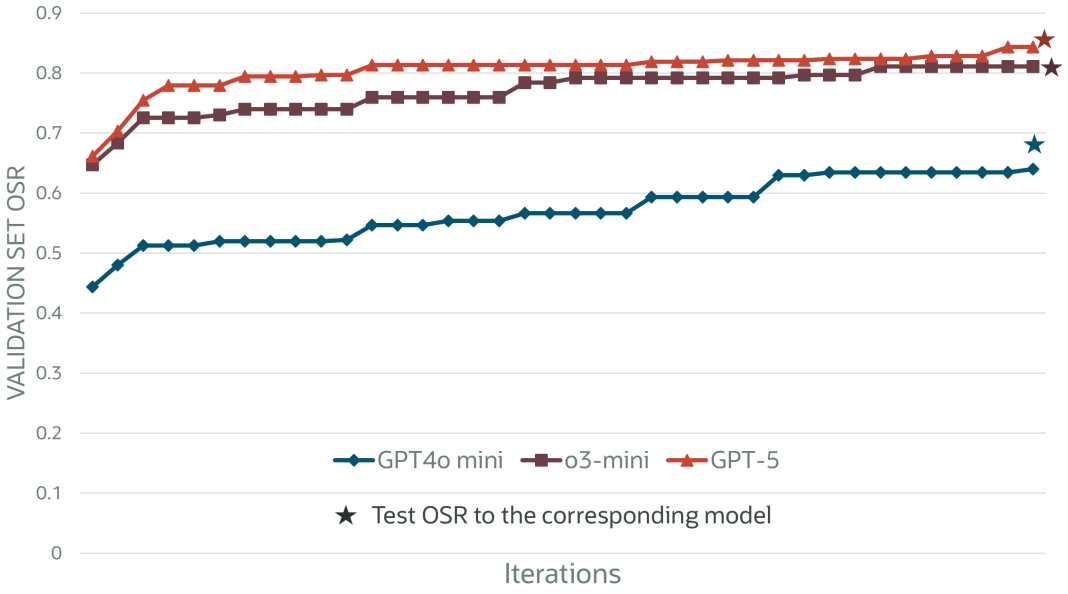


Figure 14: **JTPRO convergence on validation OSR.** Validation OSR over optimization iterations for three base models (GPT-4o mini, o3-mini, GPT-5); $\star$ marks final test OSR for the best validation-selected context. OSR rises quickly in early iterations and then plateaus, indicating rapid correction of high-impact errors followed by smaller refinements that transfer to held-out data.

### D.7 Figure 14: Convergence behavior

Figure 14 plots validation OSR over JTPRO iterations for three base models, with $\star$ denoting the final test OSR obtained using the best validation-selected context. The curves show rapid early gains followed by saturation, consistent with the reflector first correcting high-impact, systematic errors (e.g., frequent tool confusions, missing required slots, formatting/defaulting mistakes) and later iterations focusing on smaller refinements. The separation between validation trajectories and the final test markers indicates that improvements transfer to held-out queries rather than merely optimizing minibatch idiosyncrasies.

#### D.7.1 Embedding-Based Disambiguation Metric

### D.8 Tool Description Disambiguation Analysis

We study how joint optimization improves tool selection by analyzing semantic changes in tool descriptions on the ToolAce-500 benchmark.

#### D.8.1 Description Enrichment

The optimizer modifies 55 out of 500 tool descriptions (11%), increasing the average description length from 86.1 to 100.1 characters (+16.3%). These edits primarily target *disambiguation*, explicitly differentiating tools with overlapping semantics.

**Example: `search` vs. `web_search`**

- **search (before):** "Perform Google search and get results."
- **search (after):** "Perform Google search with advanced locale controls (gl/hl), country restrictions (cr), and time filters (tbs). *NOT for general web article or paper discovery, prefer* `web_search` *for generic queries.*"
- **web_search (after):** "Search the web for relevant pages. *PREFERRED for general-purpose web, article, and paper discovery. Do not confuse with similarly named* `search` *tools.*"

#### D.8.2 Confusable Tool Groups

We identify tools sharing common name prefixes (e.g., `get_user_*`, `get_all_*`) as potentially confusable. This yields 37 groups comprising 109 tools, representing high-risk ambiguity regions where models frequently select semantically similar but incorrect tools.

To quantify disambiguation quality, we compute pairwise cosine similarity between tool descriptions within each confusable group using sentence embeddings (`all-MiniLM-L6-v2`). Lower intra-group similarity indicates stronger semantic separation.

| Tool Group | Before | After | Δ |
|---|---|---|---|
| `get_ip_*` (2) | 0.668 | 0.502 | −0.166 |
| `get_page_*` (2) | 0.849 | 0.736 | −0.113 |
| `get_languages_*` (2) | 0.676 | 0.584 | −0.092 |
| `get_trending_*` (3) | 0.437 | 0.377 | −0.060 |
| `search_by_*` (2) | 0.281 | 0.265 | −0.016 |

Table 5: Intra-group cosine similarity (lower indicates better disambiguation) for the most improved confusable tool groups. Parentheses denote group size.

| Metric | Value |
|---|---|
| Total tools analyzed | 500 |
| Modified descriptions | 55 (11.0%) |
| Avg. length (before) | 86.1 chars |
| Avg. length (after) | 100.1 chars |
| Relative increase | +16.3% |
| Confusable groups | 37 |
| Confusable tools | 109 |
| Groups improved | 6 (16.2%) |
| Avg. similarity reduction | 0.012 |

Table 6: Summary of tool description disambiguation on ToolAce-500.

The `get_ip_*` group exhibits the largest improvement, with similarity reduced from 0.668 to 0.502. This change reflects the addition of explicit preference guidance, e.g., "*Preferred for general IP geolocation requests. Use this instead of* `get_geolocation_by_ip` *unless extended fields are required.*"

#### D.8.3 Disambiguation Patterns Learned

Across the 55 modified tools, we observe four recurring disambiguation strategies:

1. **Parameter format guidance** (28 tools): e.g., "Pass `country_code` as a 2-letter lowercase ISO code."
2. **Explicit preference signals** (9 tools): e.g., "PREFERRED for…"
3. **Negative constraints** (5 tools): e.g., "NOT for general web article discovery."
4. **Cross-tool references** (3 tools): e.g., "Use this instead of `get_geolocation_by_ip`."

#### D.8.4 Summary Statistics

Overall, joint optimization learns to resolve tool ambiguity through targeted description edits, complementing instruction-level optimization and improving tool selection robustness.

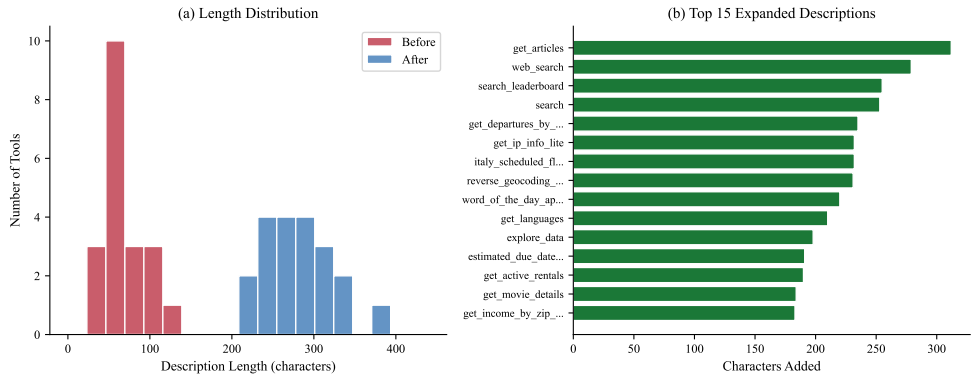


Figure 15: Tool description length analysis. (a) Distribution of description lengths before and after optimization. (b) Per-tool length changes for the 55 modified tools.

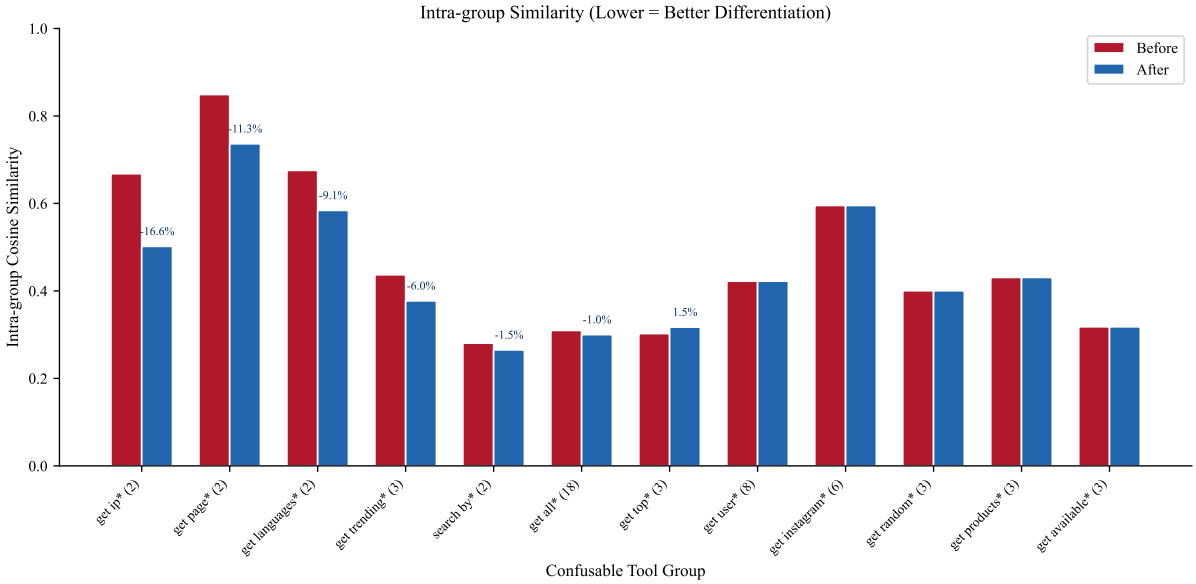


Figure 16: Intra-group cosine similarity for the top 15 confusable tool groups. Lower values indicate stronger semantic differentiation.