Don't Adapt Small Language Models for Tools; Adapt Tool Schemas to the Models

Jonggeun Lee*, **Woojung Song***, **Jongwook Han, Haesung Pyun, Yohan Jo†**
Graduate School of Data Science, Seoul National University
{jonggeun.lee, opusdeisong, johnhan00, haesung.pyun, yohan.jo}@snu.ac.kr

Abstract

Small language models (SLMs) enable scalable tool-augmented multi-agent systems where multiple SLMs handle subtasks orchestrated by a powerful coordinator. However, they struggle with tool-use tasks, particularly in selecting appropriate tools and identifying correct parameters. A common failure mode is *schema misalignment*: models hallucinate plausible tool names that are absent from the provided tool schema, due to different naming conventions internalized during pretraining. Rather than training models to adapt to unfamiliar schemas, we propose adapting schemas to align with models' pretrained knowledge. We introduce **PA-Tool** (Pretraining-Aligned Tool Schema Generation), a training-free method that leverages peakedness, a signal used in contamination detection that indicates pretraining familiarity, to rename tool components. By generating multiple candidates and selecting the candidate with the highest peakedness, PA-Tool identifies pretraining-aligned naming patterns. Experiments on MetaTool and RoTBench show improvements of up to 17%, with schema misalignment errors reduced by 80%. PA-Tool enables small models to substantially improve tool-use accuracy without retraining, showing that schema-level interventions can unlock the tool-use potential of resource-efficient models. Our code is available at `https://github.com/holi-lab/PA-Tool`.

1 Introduction

Tool-augmented language models have become essential components of modern AI systems (Qu et al., 2025b). As these systems mature, there is growing interest in deploying small language models (SLMs, typically $\leq$ 8B) for tool use. This interest is driven by multi-agent architectures, where a powerful coordinator orchestrates reasoning while

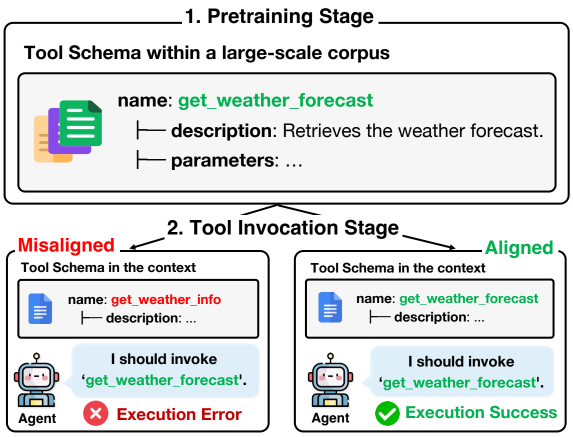

Figure 1: Effect of schema alignment on tool invocation. **Top**: Models learn tool schemas during pretraining. **Bottom-Left**: When schemas misalign, models generate plausible but non-existent tools. **Bottom-Right**: Schema alignment prevents such errors.

multiple SLMs handle subtasks (Belcak et al., 2025), and by edge deployment scenarios with strict resource constraints (Chen et al., 2025b). In these systems, SLMs must perform two critical operations: tool selection (identifying which API to call) and parameter identification (providing appropriate arguments). However, SLMs struggle significantly with these tasks, exhibiting severe performance degradation compared to larger models (Erdogan et al., 2024; Patil et al., 2025).

A common failure pattern is *schema misalignment*: even when appropriate tools are provided in context, models fail to invoke them and instead hallucinate plausible-looking components that are not in the provided schema[1] (Figure 1, Bottom-Left). This suggests that models fall back on familiar naming conventions internalized during pretraining when faced with unfamiliar schemas. Therefore, we hypothesize that aligning tool component names with patterns from the model's internalized knowledge can reduce this error pattern (Figure 1,

*Equal contribution.
†Corresponding author.

[1] In this work, a tool schema refers to a hierarchical structure of tools and parameters with descriptions (Figure 1, Top).

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 20695–20719
July 2-7, 2026

Bottom-Right).

To operationalize this idea, we propose PA-Tool (Pretraining-Aligned Tool Schema Generation), which generates a mapping between original tool component names and alternatives familiar to the models. To identify what naming conventions the model is familiar with, we leverage an insight from contamination detection method (Dong et al., 2024), which detect whether models encountered specific data during training. A key finding from this study is that patterns frequently seen during training exhibit *peakedness*: models generate highly similar outputs across multiple samples, creating a concentrated distribution. We adopt this *peakedness* as a signal for the model's familiarity with tool names. PA-Tool renames each component through three stages: (1) instructing the tool-using language model to generate multiple candidate names based on a description of the component (Chen et al., 2021), (2) computing the peakedness of each candidate, measured as the number of other candidates sufficiently similar under character-level edit distance (Levenshtein, 1965), and (3) selecting the candidate with the highest peakedness as the new name, as it is considered to best align with the model's internalized knowledge.

Experiments on MetaTool and RoTBench demonstrate PA-Tool's effectiveness across diverse scenarios. On MetaTool, PA-Tool achieves tool selection gains of up to 17% on Reliability (where models must identify that no suitable tool exists) and 9.6% on multi-tool selection scenarios. On RoTBench, PA-Tool yields tool selection gains of up to 10% (single-turn) and 6% (multi-turn), showing that alignment benefits persist across extended contexts, with consistent gains also appearing on parameter identification.

Our comprehensive analysis reveals that PA-Tool's effectiveness stems from directly addressing schema misalignment, the predominant failure mode in SLMs. Errors where models generate plausible but non-existent tool names decrease by 80.0% with PA-Tool, with accompanying reductions in other error types (18.8–24.0%). The underlying mechanism is further validated by showing that peakedness consistently increases as models encounter tool schemas during training, consistent with its role as a familiarity signal.

PA-Tool also demonstrates broad compatibility with existing approaches. For supervised fine-tuning, PA-Tool alone outperforms fine-tuned models on several subtasks without any training, and applying PA-Tool on top of fine-tuned models often yields further improvements. It also provides complementary gains when combined with training-free methods such as retrieval-based correction, constrained generation, and description enhancement. Beyond tool selection benchmarks, PA-Tool improves end-to-end task completion on API-Bank and τ-Bench, confirming that schema alignment translates to practical performance gains.

This training-free approach requires only a one-time schema mapping: rather than forcing models to conform to arbitrary schemas, we adapt the interface to match their internalized knowledge, enabling diverse tool use without model modification, retraining, or risk of catastrophic forgetting.

Our contributions are summarized as follows:

- We propose PA-Tool, a training-free schema optimization method that identifies pretraining-aligned tool names by repurposing peakedness, a signal from data contamination.
- We demonstrate improvements of up to 17% across SLMs on MetaTool and RoTBench, with benefits extending from tool selection to parameter identification in both single-turn and multi-turn settings.
- We demonstrate that through a simple, easily deployable mapping interface, SLMs can boost their tool-use accuracy while maintaining computational efficiency in resource-constrained systems.

2 Related Work

2.1 Emerging Agentic Frameworks

Recent advances in LLMs have enabled sophisticated agentic frameworks that decompose complex tasks into specialized modules, each coordinated by dedicated agents (Shinn et al., 2023; Agashe et al., 2025; Sapkota et al., 2026). In these multi-agent systems, SLMs are increasingly replacing larger models within individual modules to reduce computational costs and latency while maintaining specialized functionalities (Cheng et al., 2024; Belcak et al., 2025). However, these systems remain vulnerable to cascading failures when SLM agents malfunction in foundational tool interaction tasks such as tool selection and parameter identification (Erdogan et al., 2024; Patil et al., 2025). A particularly challenging failure mode is schema misalignment: SLM errors often manifest as plausible yet incorrect outputs, such as generating tool names that seem reasonable but do not exist in the actual schema. However, existing works primarily focus on improving agent architectures or coordination strategies, without addressing the schema-level mismatch that causes these failures.

2.2 Improving Tool Utilization in LLMs

As LLMs demonstrate the capability to interact with external tools (Schick et al., 2023; Hsieh et al., 2023), research has focused on evaluating and improving their tool-use capabilities. Evaluation efforts have developed benchmarks ranging from fine-grained assessments of tool selection and parameter identification (Huang et al., 2024; Li et al., 2023; Ye et al., 2024; Chen et al., 2024; Patil et al., 2025) to end-to-end multi-step evaluation (Qin et al., 2024; Trivedi et al., 2024; Yao et al., 2025; Shim et al., 2026; Seo et al., 2026).

Approaches to improving tool-use capabilities follow two main directions. Training-based methods employ supervised fine-tuning (Qin et al., 2024; Liu et al., 2025; Zhang et al., 2025) or reinforcement learning (Shi et al., 2024b; Qian et al., 2025; Chen et al., 2025a; Feng et al., 2026), but they require substantial data or computational resources. Training-free methods refine tool documentation (Yuan et al., 2025; Qu et al., 2025a) or leverage interaction histories (Fu et al., 2024; Wang et al., 2024; Zhao et al., 2024; Cui et al., 2025). However, existing training-free approaches primarily focus on improving descriptions or accumulating experiential knowledge through interactions rather than aligning the schema itself with model-preferred representations. Our work directly addresses this gap by generating schemas aligned with models' pretrained knowledge, targeting the root cause of schema misalignment.

2.3 Data Contamination Detection

Data contamination, defined as overlap between pretraining data and evaluation benchmarks, can inflate performance through memorization. Early detection methods use n-gram overlap (Brown et al., 2020), but these approaches fail to detect semantic paraphrasing. Probability-based approaches such as Min-k% Prob (Shi et al., 2024a) and perplexity analysis (Li, 2023) require token probabilities that are unavailable in closed-source models. LLM Decontaminator (Yang et al., 2023) uses auxiliary models for semantic similarity but introduces additional dependencies. Most relevant to our work, CDD (Dong et al., 2024) identifies memorized patterns by measuring peakedness in sampled candidates, making it applicable to any black-box model. We adapt this peakedness mechanism to identify pretraining-aligned schema patterns, transforming contamination detection into a constructive tool for schema optimization.

3 PA-Tool: Pretraining-Aligned Tool Schema Generation

We now formalize our approach to generating pretraining-aligned tool schemas. A tool schema is structured documentation that defines tools and parameters along with descriptions for each component. Our objective is to rename these components (i.e., tool names and parameter names) with ones that the model has frequently encountered during pretraining.

3.1 Framework Overview

Let $\mathcal{M}$ denote a pretrained language model. Given a natural language description d of a schema component's functionality (e.g., a tool or a parameter description), our objective is to generate an optimal name s^* that represents the naming pattern most deeply internalized by $\mathcal{M}$. As illustrated in Figure 2, our approach operates in three stages: (1) we provide $\mathcal{M}$ with a description of a component and instruct it to generate its name multiple times (Stage 1), (2) we compute peakedness scores that measure how many similar candidates cluster around each candidate name (Stage 2), and (3) we select the candidate with the highest peakedness as the pretraining-aligned name (Stage 3). By applying this process to each component in the schema hierarchy, we construct a dictionary mapping from original names to pretraining-aligned names. The complete algorithm is provided in Appendix A.1.

3.2 Stage 1: Candidate Generation

In this stage, we collect diverse candidate names of a component that the model may have encountered during pretraining. As illustrated in Figure 2-1, given a component's description, we sample N candidate names $\mathcal{C} = \{s_1, s_2, \ldots, s_N\}$ from the language model with temperature $t \in (0, 1]$. This temperature-controlled sampling explores the model's learned distribution beyond the single greedy path, revealing diverse naming patterns (Chen et al., 2021). The detailed prompt structure is provided in Appendix L. We generate a reference name s_{ref} using greedy decoding ($t = 0$) for tie-breaking in the selection stage.

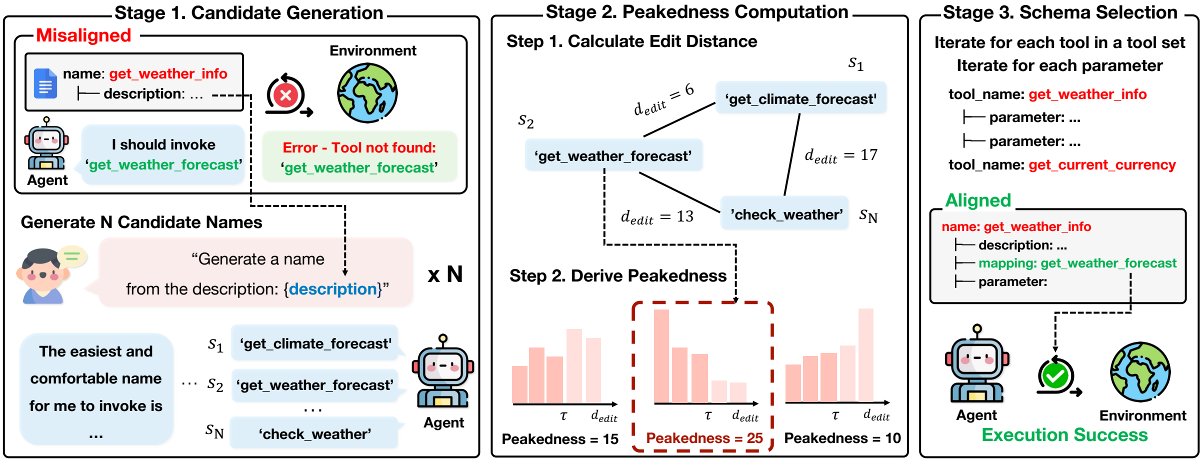

Figure 2: Overview of our PA-Tool framework.

3.3 Stage 2: Peakedness Computation

Following contamination detection principles (Dong et al., 2024), we analyze the local concentration of the output distribution to identify strongly memorized patterns. As illustrated in Figure 2-2, for each candidate name s_i, we compute its peakedness score by counting how many similar candidate names the model generates. The intuition is that candidate names with many similar variants indicate patterns frequently encountered during pretraining that the model generates consistently, while isolated candidates suggest less familiar patterns.

To quantify this clustering behavior, we first define a similarity threshold τ based on the maximum character length in the candidate set:

$$\tau = \alpha \cdot \ell_{\text{max}} \tag{1}$$

where $\ell_{\text{max}} = \max_{s_i \in \mathcal{C}} |s_i|$ is the maximum character length across all candidates, and $\alpha \in [0, 1]$ is a hyperparameter that controls the strictness of similarity. This length-adaptive threshold ensures that longer names are allowed proportionally more variation while maintaining consistent similarity criteria across different name lengths.

The peakedness score for each candidate s_i is then computed as:

$$\phi(s_i) = \sum_{j \neq i} \mathbb{I}(d_{\text{edit}}(s_i, s_j) \leq \tau) \tag{2}$$

where $d_{\text{edit}}(\cdot, \cdot)$ denotes the character-level edit distance using the Levenshtein algorithm (Levenshtein, 1965), and $\mathbb{I}(\cdot)$ is the indicator function that equals 1 when the condition is satisfied and 0 otherwise. This score counts the number of candidates that fall within the similarity threshold from s_i, reflecting how many similar names the model generates around this pattern.

3.4 Stage 3: Schema Selection

The representative name is selected as the candidate with the maximum peakedness:

$$s^* = \arg\max_{s_i \in \mathcal{C}} \phi(s_i) \tag{3}$$

This criterion identifies the naming pattern that the model generates most consistently across multiple sampling attempts, suggesting that this pattern is the most deeply internalized from the training data. When multiple candidates tie for maximum peakedness, we select the one with the minimum edit distance to the reference name:

$$s^* = \arg\min_{s_i \in \mathcal{C}^*} d_{\text{edit}}(s_i, s_{\text{ref}}) \tag{4}$$

where $\mathcal{C}^* = \{s_i \in \mathcal{C} : \phi(s_i) = \max_{s_j \in \mathcal{C}} \phi(s_j)\}$ contains all candidates with maximum peakedness.

This approach rests on the hypothesis that frequently occurring patterns in training data create local maxima in the model's output distribution. By identifying regions of high peakedness, we effectively locate these memorized naming conventions, which represent the most natural and well-formed component names according to the model's learned knowledge. Through iterative application across all components in a schema (Figure 2-3), we obtain the final pretraining-aligned schema. An example of a schema generated through this process is provided in Appendix A.2. We repeat our three-stage process for all tool sets within the system. When name collisions occur across different tools due to highly similar descriptions, we resolve them using the priority-based locking mechanism (Appendix A.3).

4 Experimental Setup

4.1 Benchmarks

MetaTool. MetaTool (Huang et al., 2024) evaluates tool selection capabilities across 4,287 test cases with 199 tools in four subtasks: (1) **Similar** tests semantic comprehension by distinguishing tools with overlapping functionalities (e.g., Sudoku vs. Tic-Tac-Toe); (2) **Scenario** selects appropriate tools based on user-specific contexts and requirements (e.g., software engineers, students); (3) **Reliability** assesses whether models can directly indicate when no suitable tool is available rather than hallucinating; and (4) **Multi-tool** measures whether models can correctly select multiple tools when tasks require composition of functionalities.

RoTBench. RoTBench (Ye et al., 2024) evaluates two tool-use capabilities across 105 test cases with 568 tools: (1) **Tool Selection** assesses whether models correctly identify the appropriate tool; and (2) **Parameter Identification** measures whether models accurately extract the required parameter set, conditioned on correct tool selection. We evaluate in both single-turn and multi-turn settings, where the latter provides two preceding interaction turns (e.g., a failed attempt with incorrect parameters) before evaluating on the third turn. RoTBench also introduces four levels of noise perturbation to tool component names (e.g., reversed names) to stress-test robustness; we focus on the Clean environment in our main experiments, with results across all levels reported in Appendix I.

4.2 Models and Baselines

We primarily focus on SLMs, evaluating four open-source models: Qwen2.5-3B/7B (Qwen et al., 2025), Llama3.1-8B (Meta AI, 2024a), and Llama3.2-3B (Meta AI, 2024b). For each model, we evaluate five configurations: (1) **Base** uses the original schema in the benchmarks without modifications; (2) **Greedy** uses greedy decoding to generate deterministic schema names; (3) **MostFreq** selects the most frequently generated candidate from our sampling process; (4) **Human** uses schemas where two PhD-level annotators with software engineering backgrounds manually renamed tools to more intuitive names (evaluated only on MetaTool due to RoTBench's large tool sets); (5) **PA-Tool** applies our peakedness-based method. To contextualize SLM performance and quantify PA-Tool's improvements, we also compare against three state-of-the-art closed-source models: GPT-4.1-mini (OpenAI, 2025), Gemini-2.5-Flash (Basu Mallick et al., 2025), and Claude-Sonnet-4.5 (Anthropic, 2025).

4.3 Implementation Details

When generating schemas with PA-Tool, we use 32 candidates at temperature 0.4 with $\alpha = 0.2$ (sensitivity analysis in Appendix A.4). Each component's description is provided individually (see Figure 2-1; prompt template in Appendix L).

For benchmark inference, we use temperature 0 to ensure reproducible results across all experiments. We use accuracy as the primary evaluation metric across both benchmarks, measuring the percentage of test cases where the model's predictions exactly match the ground-truth labels. Detailed evaluation protocols are in Appendix B.

5 Main Results

Table 1 presents comprehensive results across MetaTool and RoTBench benchmarks, demonstrating PA-Tool's effectiveness.

MetaTool. PA-Tool substantially improves performance over Base models across most MetaTool subtasks. The most substantial gains appear in Reliability (up to 17.0%, e.g., Llama3.2-3B: 43.6→60.6%), where models must recognize when no suitable tool exists, a task critically dependent on clearly understanding available options. In Multi-tool, gains reach 9.6% (Llama3.1-8B: 78.7→88.3%). When tasks require identifying multiple tools simultaneously, schema misalignment compounds across each selection, making alignment particularly critical. Similar and Scenario tasks show improvements up to 10.7%.

PA-Tool outperforms training-free alternatives on most tasks. Greedy occasionally underperforms Base (Llama3.2-3B Reliability: 39.8% vs. 43.6%) as it generates only a single candidate, limiting schema space exploration. While MostFreq sometimes achieves competitive results by capturing frequency, PA-Tool shows more consistent improvements by measuring distributional concentration. Compared to Human-designed schemas, PA-Tool achieves comparable or superior performance (Llama3.2-3B Similar: 65.7% vs. 58.6%), demonstrating automated alignment can match human intuition while being scalable.

RoTBench. RoTBench evaluates models in single-turn and multi-turn settings across tool selection and parameter identification. PA-Tool demonstrates consistent improvements in single-turn tool

Model	Method	MetaTool				RoTBench			
		Tool Selection				Single-turn		Multi-turn	
		Similar	Scenario	Reliability	Multi-tool	Tool Sel.	Param Iden.	Tool Sel.	Param Iden.
Closed-Source Models									
GPT-4.1-mini	Base	**79.6**	**84.3**	76.3	72.2	79.1	58.1	**71.4**	**61.4**
Gemini-2.5-Flash	Base	70.0	79.8	**89.2**	77.3	82.9	56.2	60.0	54.3
Claude-Sonnet-4.5	Base	75.6	83.0	84.3	**85.1**	**83.3**	**67.6**	64.3	58.6
Small Language Models									
Qwen2.5-3B	Base	48.7	55.3	83.6	**75.1**	12.4	7.6	10.0	10.0
	Greedy	49.1	57.0	82.9	63.4	13.3	7.6	11.4	7.1
	MostFreq	50.5	56.3	84.3	73.0	14.3	8.6	12.9	8.6
	Human	**54.6**	**62.9**	80.6	72.6	-	-	-	-
	PA-Tool	50.0	58.8	**86.2**	72.6	**18.1**	**10.5**	**15.7**	**14.3**
Qwen2.5-7B	Base	59.6	74.4	78.3	78.3	49.5	20.0	21.4	21.4
	Greedy	60.6	75.4	85.4	82.7	50.5	21.0	18.6	15.7
	MostFreq	65.3	75.5	**90.0**	**86.1**	50.5	**23.8**	24.3	20.0
	Human	**65.8**	76.7	78.6	82.1	-	-	-	-
	PA-Tool	64.1	**78.4**	88.2	84.9	**55.2**	21.9	**27.1**	**22.9**
Llama3.2-3B	Base	55.0	58.6	43.6	79.1	56.2	20.0	32.9	27.1
	Greedy	57.7	58.9	39.8	70.4	59.1	20.0	32.9	25.7
	MostFreq	61.3	64.2	**65.7**	**81.3**	60.0	18.1	32.9	27.1
	Human	58.6	60.9	55.2	72.6	-	-	-	-
	PA-Tool	**65.7**	**67.7**	60.6	80.5	**62.9**	**21.9**	**34.3**	**28.6**
Llama3.1-8B	Base	61.5	73.9	53.5	78.7	58.1	17.1	42.8	34.3
	Greedy	64.6	72.9	51.5	78.9	63.8	**18.1**	44.3	32.9
	MostFreq	68.8	79.3	**66.4**	85.7	66.7	**18.1**	45.7	34.3
	Human	69.3	78.9	63.5	86.5	-	-	-	-
	PA-Tool	**70.4**	**79.9**	66.0	**88.3**	**68.6**	**18.1**	**48.6**	**35.7**

Table 1: Performance comparison on MetaTool and RoTBench. All metrics are reported as accuracy (%). **Bold** indicates the best performance within each open-source model and among all closed-source models.

selection across all models, with gains ranging from 5.7% to 10.5% (Llama3.1-8B: 58.1→68.6%). In multi-turn settings, improvements of up to 6% show that alignment benefits persist across extended contexts. Notably, improvements extend beyond tool selection to parameter identification, with gains of up to 4.3% (Qwen2.5-3B multi-turn: 10.0→14.3%), confirming that schema alignment improves overall tool-use accuracy.

Comparison with Closed-source Models. While a gap remains against closed-source models (Claude-Sonnet-4.5: 83.3% tool selection on RoTBench single-turn), PA-Tool enables small models to achieve competitive or superior performance on specific subtasks. In MetaTool's Multi-tool, Llama3.1-8B with PA-Tool (88.3%) surpasses all closed-source models including Claude-Sonnet-4.5 (85.1%). In Reliability, Qwen2.5-7B with PA-Tool (88.2%) approaches Gemini-2.5-Flash (89.2%). These results suggest that targeted schema alignment can narrow the gap with larger models on subtasks where tool naming is a key factor.

Generalization to Diverse Models. Beyond the SLMs examined here, we apply PA-Tool to diverse model families and scales, including SLMs (Ministral-8B, GPT-4.1-nano, Gemini-2.5-Flash-Lite), LLMs (Llama3.3-70B, GPT-4.1-mini, Gemini-2.5-Flash), and reasoning models (Qwen3-1.7B and Qwen3-4B in thinking mode). As shown in Table 11, PA-Tool improves performance across most models and settings, with larger gains for SLMs where schema misalignment is a dominant failure mode (e.g., Ministral-8B gains 8.5% on RoTBench single-turn tool selection). For LLMs, gains are smaller on average but remain substantial where misalignment compounds, such as multi-tool composition (e.g., GPT-4.1-mini gains 12.1% on MetaTool Multi-tool). For reasoning models, PA-Tool provides gains of up to 10.5% on RoTBench (Qwen3-1.7B single-turn parameter identification) despite strong baselines from chain-of-thought, confirming that reasoning and schema alignment address complementary failure modes (see Appendix C for details).

6 Analysis

We analyze PA-Tool along three directions. We first ask *why* it works, identifying the error modes it addresses, validating that peakedness reflects pretraining familiarity, and isolating which schema components drive the gains (§6.1–6.3). We then

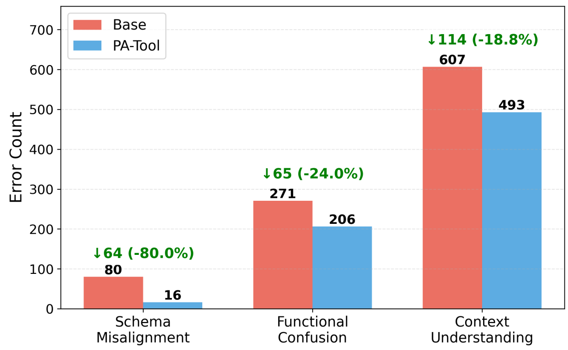

Figure 3: Error count distribution for Llama3.1-8B on MetaTool tool selection tasks.

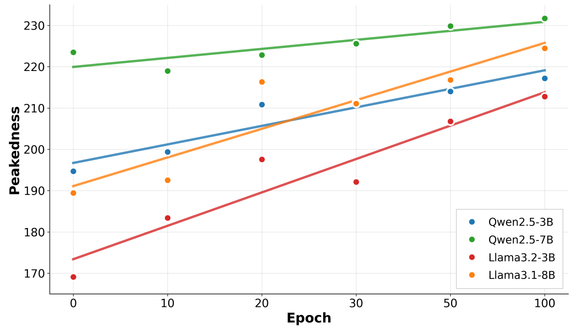

Figure 4: Peakedness across training epochs.

Model	Config.	Single-turn		Multi-turn	
		Tool Sel.	Param Id.	Tool Sel.	Param Id.
Llama3.2-3B	Base	56.2	20.0	32.9	27.1
	Tool-only	57.1	21.9	**35.7**	25.7
	Param-only	58.1	**23.8**	31.4	**28.6**
	Both (PA-Tool)	**62.9**	21.9	34.3	**28.6**
Llama3.1-8B	Base	58.1	17.1	42.8	34.3
	Tool-only	62.9	14.3	47.1	34.3
	Param-only	56.2	17.1	45.7	34.3
	Both (PA-Tool)	**68.6**	**18.1**	**48.6**	**35.7**

Table 2: Ablation of tool name and parameter name alignment on RoTBench.

ask how it relates to existing tool-use approaches, showing that it complements both training-based fine-tuning and other training-free methods rather than competing with them (§6.4–6.5). Finally, we ask how far its benefits extend, testing generalization to end-to-end benchmarks, human perception, and cross-model schema transfer (§6.6–6.8).

6.1 Error Analysis

To understand how PA-Tool addresses different failure modes, we analyze error distributions in Llama3.1-8B before and after applying PA-Tool. We categorize incorrect predictions into three types: *Schema Misalignment Error* (generating non-existent but plausible tools), *Functional Confusion Error* (selecting wrong tools with similar functionality), and *Context Understanding Error* (selecting functionally unrelated tools). Detailed definitions and settings are in Appendix E.

Figure 3 shows how PA-Tool affects different error types. PA-Tool substantially reduces all error types, with particularly strong impact on Schema Misalignment (80.0% reduction) and meaningful improvements on Functional Confusion (24.0%) and Context Understanding (18.8%). This demonstrates that schema alignment not only reduces naming-related failures but also indirectly benefits other aspects of tool selection, suggesting that misaligned names contribute to other error sources.

6.2 Validating Peakedness

PA-Tool assumes that peakedness reflects models' familiarity with naming patterns. To validate this, we simulate pretraining conditions where models are repeatedly exposed to tool schemas and measure whether peakedness increases accordingly.

We use the same MetaTool data and settings from Section 6.4, which contains user queries paired with tool schemas (names and descriptions). The key difference from SFT is the loss computation: we compute loss on all tokens rather than only on assistant responses, to simulate pretraining-style exposure. We train four models (Qwen2.5-3B/7B, Llama3.2-3B, Llama3.1-8B), measuring peakedness at epochs 0, 10, 20, 30, 50, and 100 using 256 candidates per tool component for stable measurements. Figure 4 shows that peakedness consistently increases with training epochs across all models, with gains up to +25.8% (Llama3.2-3B). This supports our hypothesis that peakedness increases with training exposure, consistent with its use in PA-Tool as a familiarity signal.

6.3 Component Ablation

To isolate the contributions of tool name and parameter name alignment, we conduct an ablation on RoTBench, which evaluates both capabilities.

Table 2 shows that each alignment component tends to benefit its target capability: for example, tool name alignment improves tool selection on Llama3.1-8B (single-turn: 58.1→62.9%), while parameter name alignment improves parameter identification on Llama3.2-3B (single-turn: 20.0→23.8%). Combining both yields the most balanced results across settings: for Llama3.2-3B, it attains the highest single-turn tool selection while remaining competitive elsewhere, and for Llama3.1-8B, the joint configuration achieves the best performance across all settings. We therefore

Configuration	MetaTool				RoTBench
	Similar	Scenario	Reliab.	Multi.	Tool Sel.
Base	65.2	71.1	55.1	79.8	58.1
+ PA-Tool	72.7	75.9	**67.2**	**89.9**	**68.6**
+ SFT1	71.2	77.6	57.1	82.8	61.0
+ SFT1 + PA-Tool	72.7	**80.8**	57.1	**89.9**	65.7
+ SFT2 (2× data)	71.2	80.3	58.1	81.8	59.1
+ SFT2 + PA-Tool	**73.2**	79.2	55.6	86.9	61.9

Table 3: Results of fine-tuning and PA-Tool on 20% random sample from MetaTool and full RoTBench.

adopt the joint configuration as the default.

6.4 Integration with Supervised Fine-tuning

While PA-Tool is training-free, we examine how it compares to and combines with supervised fine-tuning (SFT) on MetaTool. We compare six configurations using Llama3.1-8B: (1) Base, (2) Base + PA-Tool, (3) SFT1 ($\sim$2.5K samples), (4) SFT1 + PA-Tool, (5) SFT2 (2$\times$ data, $\sim$5K samples), and (6) SFT2 + PA-Tool. Although SFT is trained on MetaTool data, PA-Tool is applied independently to each benchmark's own tool schemas. Training details and additional experiments on other models (i.e., Qwen2.5-3B/7B) are in Appendix D.

Table 3 shows PA-Tool provides gains in most configurations. Notably, PA-Tool alone outperforms both SFT1 and SFT2 on Reliability and Multi-tool without any training, and combining both achieves the best results, with SFT1 + PA-Tool reaching 80.8% on Scenario. That PA-Tool remains effective even after fine-tuning suggests that SFT teaches tool-use reasoning but does not fully resolve the model's pretrained naming preferences, leaving room for PA-Tool to provide additional gains. SFT2 shows marginal gains over SFT1 on MetaTool but degrades on RoTBench relative to SFT1 (59.1% vs. 61.0%), suggesting that additional domain-specific training narrows the model's generalization to unseen tools. PA-Tool avoids this risk entirely as it requires no training.

6.5 Integration with Training-Free Methods

Beyond schema renaming, we evaluate PA-Tool's synergy with other training-free approaches. We test three categories of methods: (1) **Retrieval-based correction**, which maps misaligned outputs to valid tools post-hoc using BM25 (Robertson and Zaragoza, 2009) or ToolLLM (Qin et al., 2024); (2) **Constrained generation**, which enforces valid tool names through JSON schema constraints during decoding; and (3) **Description enhancement** via EasyTool (Yuan et al., 2025), which rewrites tool descriptions for clarity without modifying names. These methods address different aspects of tool-use and are compatible with PA-Tool's schema alignment approach. We evaluate both standalone and combined configurations (Method + PA-Tool), with detailed settings presented in Appendix G.

Table 15 shows that retrieval methods yield limited gains (<3% improvement), as post-hoc matching often fails to recover the model's intended tool. Constrained generation substantially improves RoTBench (Qwen2.5-7B: 78.1% vs. Base 49.5%) by eliminating format errors, but shows mixed results on MetaTool (Table 16). Combining PA-Tool with retrieval or constrained generation improves results on most subtasks (Tables 15, 16), confirming these approaches are complementary.

For description enhancement (Table 17), PA-Tool and EasyTool show complementary strengths: PA-Tool achieves larger gains on tool selection for Llama models (e.g., Llama3.2-3B Similar: 65.7% vs. 45.3%), while EasyTool is more effective on some Qwen configurations. Combining both yields the best performance on RoTBench for most models (e.g., Qwen2.5-3B single-turn: 7.6%→19.1%), as name alignment and description clarity address orthogonal axis of schema quality.

6.6 Additional Benchmarks

To demonstrate that PA-Tool generalizes across different evaluation settings, we evaluate on API-Bank (Li et al., 2023) and τ-Bench (Yao et al., 2025). API-Bank evaluates tool-use accuracy by measuring exact matches between predicted and ground-truth tools and parameters. τ-Bench is an end-to-end benchmark that assesses agent performance through multi-turn dialogues where agents must complete tasks through interaction with users. Detailed settings are in Appendix B.

API-Bank. PA-Tool consistently improves performance in most settings (Table 4). Qwen2.5-3B shows substantial gains on the Call task (18.0%→28.5%), while improvements persist on Call+Retrieve, which requires both tool selection and information retrieval. This confirms that schema alignment enhances tool-use accuracy.

τ-Bench. To verify that improved tool selection translates to better task completion, we evaluate on τ-Bench (Retail). We run each evaluation 5 times to mitigate variance from the user simulator. PA-Tool consistently improves end-to-end task completion across all models (Table 4). Analyzing Llama3.1-8B's tool-use trajectories reveals that

Model	Call		Call+Retrieve		τ-Bench	
	Base	PA-Tool	Base	PA-Tool	Base	PA-Tool
Qwen2.5-3B	18.0	**28.5**	18.5	**21.9**	3.7	**3.8**
Qwen2.5-7B	25.7	**34.7**	23.5	**26.9**	6.8	**9.7**
Llama3.2-3B	5.1	**5.4**	**10.9**	9.2	4.5	**5.6**
Llama3.1-8B	28.0	**29.8**	21.0	**22.4**	9.7	**11.1**

Table 4: Performance on API-Bank and end-to-end task completion (%) on τ-Bench (Retail, N=5).

Dimension	Rating		Δ	Preference	
	Orig	PA		Orig	PA
Understand.	2.72	**3.41**	+.69	10.2	**52.3**
Func. match	2.78	**3.44**	+.66	10.4	**50.9**

Table 5: Human evaluation of PA-Tool names vs. original names (3 annotators $\times$ 199 tools).

schema misalignment errors decreased from 115 to 98 cases, demonstrating that addressing schema misalignment contributes substantially to the observed performance gains.

6.7 Human Evaluation of Renamed Schemas

While PA-Tool improves model performance, a natural concern is whether optimizing schemas for model familiarity produces names that are unintuitive or obscure to human developers. To address this, we conduct a human evaluation comparing PA-Tool names with the original names.

Three annotators with software development experience evaluated all 199 MetaTool tools. Each annotator was presented with a tool's functional description alongside the original name and the PA-Tool name (generated by Llama3.1-8B) in a blind setup (labeled "Name A" and "Name B" in randomized order). For each pair, annotators (1) rated both names on a 1–5 scale for *ease of understanding* and *match to functionality*, and (2) indicated which name they preferred. This yielded 597 responses.

As shown in Table 5, PA-Tool names receive higher ratings on both dimensions, with improvements that are statistically significant (paired t-test, $p < 10^{-35}$). Over 50% of responses prefer the PA-Tool name, while only about 10% prefer the original. Table 6 illustrates representative cases. The largest clarity gains occur when original names are brand-specific or opaque (e.g., `Figlet`$\rightarrow$`ascii_converter`, Δ=+3.00), while the few cases where PA-Tool names are rated lower involve originals that are already intuitive compound words (e.g., `ProductComparison`$\rightarrow$`compare_options`, Δ=-1.67), where renaming sacrifices specificity.

Original	PA-Tool	Δ	Description
PA-Tool improves clarity			
`Figlet`	`ascii_converter`	+3.00	Convert text into ASCII fonts
`ad4mat`	`track_traffic`	+2.67	Monetize traffic via tracking links
`universal`	`web_analyzer`	+2.67	Access web pages, analyze PDFs, etc.
PA-Tool reduces clarity			
`ProductComparison`	`compare_options`	-1.67	Compare product options
`StrologyTool`	`astro_services`	-1.00	Provide astrology services
`ShoppingAssistant`	`cart_qr_generator`	-1.00	Manage cart and display QR codes

Table 6: Representative examples of clarity changes.

These results indicate that PA-Tool's alignment with pretrained knowledge also yields names that are more readable and functionally descriptive for human developers. PA-Tool can therefore be deployed without sacrificing human interpretability.

6.8 Cross-Model Schema Transfer

In multi-agent systems where multiple SLMs handle different subtasks, generating a separate schema for each model could become burdensome. We test whether schemas aligned to one model can also benefit a different model (full results in Appendix H). Cross-model schemas generally improve over unaligned baselines, even across model families (e.g., Llama3.1-8B with Qwen2.5-7B schemas gains +8.1% on Similar; see Table 18), suggesting partially overlapping naming conventions from similar training corpora. Self-generated schemas are optimal in most settings, but the gap is small on most tasks, and larger models' schemas can benefit smaller models within the same family. Given the negligible one-time cost per model (Appendix F), model-specific generation remains the preferred option, though cross-model transfer provides a practical fallback when this is not possible.

7 Conclusion

We introduced PA-Tool, a training-free method that aligns tool schemas with models' pretrained knowledge by leveraging peakedness as a pretraining familiarity signal. Experiments demonstrate improvements of up to 17% with schema misalignment errors reduced by 80%, validating schema adaptation as an effective strategy for enhancing tool use in small language models. PA-Tool's practical advantages make it particularly valuable for resource-constrained deployments—as a simple schema-level intervention, it can be applied without model training or fine-tuning, requiring only straightforward name mapping. By bridging pretrained knowledge and real-world tool interfaces, PA-Tool unlocks small models' potential for tool-augmented applications while preserving computational efficiency.

Limitations

PA-Tool primarily targets SLMs, where schema misalignment is a dominant failure mode, and consistently improves performance in this regime. As model capacity increases, schema misalignment becomes less severe due to stronger reasoning abilities, and gains naturally diminish. This reflects the reduced prevalence of the target problem rather than a limitation of the approach itself; even at larger scales, PA-Tool provides meaningful improvements in settings where misalignment compounds, such as multi-tool composition tasks (e.g., GPT-4.1-mini gains 12% on MetaTool Multi-tool).

Our reliance on peakedness as an alignment signal assumes this metric reliably indicates pre-training familiarity. While validated across our experiments, this relationship may vary for models with substantially different training distributions. Additionally, our evaluation focuses on English-language schemas; the effectiveness of character-level metrics may differ for non-Latin scripts or morphologically complex languages.

PA-Tool operates on tool and parameter names, leaving descriptions unchanged. Extending the peakedness mechanism to descriptions risks semantic drift, where the generated description may diverge from the tool's actual functionality. Separately, PA-Tool is compatible with description enhancement approaches that operate on an orthogonal axis. As demonstrated in our EasyTool and EasyTool with PA-Tool experiments (§6.5), combining PA-Tool with description enhancement yields complementary gains, suggesting that integration with interaction-driven description refinement methods (Qu et al., 2025a; Wang et al., 2024; Fang et al., 2025) is a promising direction that we leave to future work.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) under the grant RS-2024-00333484 and by the Institute of Information & Communications Technology Planning & Evaluation (IITP) under the Leading Generative AI Human Resources Development grant IITP-2026-RS-2024-00397085 and the grant RS-2025-02215122 (Development and Demonstration of Lightweight AI Model for Smart Homes), all funded by the Korean government (MSIT). This research was also supported by the Korea Institute of Science and Technology Information (KISTI) in 2026 (No. (KISTI)K26L3M1C1), aimed at developing KONI (KISTI Open Neural Intelligence), a large language model specialized in science and technology.

References

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025. Agent s: An open agentic framework that uses computers like a human. In *The Thirteenth International Conference on Learning Representations*.

Anthropic. 2025. Introducing claude sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5. Accessed 2026-04-19.

Shrestha Basu Mallick, Sid Lall, Zach Gleicher, and Kate Olszewska. 2025. Continuing to bring you our latest models, with an improved gemini 2.5 flash and flash-lite release. https://developers.googleblog.com/en/continuing-to-bring-you-our-latest-models-with-an-improved-gemini-2-5-flash-and-flash-lite-release/. Google Developers Blog; Accessed 2026-04-19.

Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. 2025. Small language models are the future of agentic ai. *Preprint*, arXiv:2506.02153.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie

Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Wei Chen, Zhiyuan Li, and Mingyuan Ma. 2025b. Octopus: On-device language model for function calling of software APIs. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pages 329–339, Albuquerque, New Mexico. Association for Computational Linguistics.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. 2024. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9510–9529, Bangkok, Thailand. Association for Computational Linguistics.

Xiaoxue Cheng, Junyi Li, Xin Zhao, Hongzhi Zhang, Fuzheng Zhang, Di Zhang, Kun Gai, and Ji-Rong Wen. 2024. Small agent can also rock! empowering small language models as hallucination detector. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14600–14615, Miami, Florida, USA. Association for Computational Linguistics.

Yue Cui, Liuyi Yao, Shuchang Tao, Weijie Shi, Yaliang Li, Bolin Ding, and Xiaofang Zhou. 2025. Enhancing tool learning in large language models with hierarchical error checklists. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 16357–16375, Vienna, Austria. Association for Computational Linguistics.

Yihong Dong, Xue Jiang, Huanyu Liu, Zhi Jin, Bin Gu, Mengfei Yang, and Ge Li. 2024. Generalization or memorization: Data contamination and trustworthy evaluation for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 12039–12050, Bangkok, Thailand. Association for Computational Linguistics.

Lutfi Eren Erdogan, Nicholas Lee, Siddharth Jha, Sehoon Kim, Ryan Tabrizi, Suhong Moon, Coleman Richard Charles Hooper, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2024. TinyAgent: Function calling at the edge. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 80–88, Miami, Florida, USA. Association for Computational Linguistics.

Wei Fang, Yang Zhang, Kaizhi Qian, James R. Glass, and Yada Zhu. 2025. PLAY2PROMPT: Zero-shot tool instruction optimization for LLM agents via tool play. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 26274–26290, Vienna, Austria. Association for Computational Linguistics.

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

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. 2023. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675*.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. 2024. Metatool benchmark for large language models: Deciding whether to use tools and which to use. In *The Twelfth International Conference on Learning Representations*.

Vladimir I. Levenshtein. 1965. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady*, 10:707–710.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.

Yucheng Li. 2023. Estimating contamination via perplexity: Quantifying memorisation in language model evaluation. *Preprint*, arXiv:2309.10677.

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, Wang Xinzhi, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. 2025. ToolACE: Winning the points of LLM function calling. In *The Thirteenth International Conference on Learning Representations*.

Meta AI. 2024a. Introducing llama 3.1: Our most capable models to date. `https://ai.meta.com/blog/meta-llama-3-1/`. Accessed 2026-04-19.

Meta AI. 2024b. Llama 3.2: Revolutionizing edge ai and vision with open, efficient models. https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/. Accessed 2026-04-19.

OpenAI. 2025. Gpt-4.1 mini. https://openai.com/index/gpt-4-1/. Accessed 2026-04-19.

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

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. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

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. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2025a. From exploration to mastery: Enabling LLMs to master tools via self-driven interactions. In *The Thirteenth International Conference on Learning Representations*.

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

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

Ranjan Sapkota, Konstantinos I. Roumeliotis, and Manoj Karkee. 2026. Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges. *Information Fusion*, 126:103599.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Gyuhyeon Seo, Jungwoo Yang, Junseong Pyo, Nalim Kim, Jonggeun Lee, and Yohan Jo. 2026. Simuhome: A temporal- and environment-aware benchmark for smart home LLM agents. In *The Fourteenth International Conference on Learning Representations*.

Weijia Shi, Anirudh Ajith, Mengzhou Xia, Yangsibo Huang, Daogao Liu, Terra Blevins, Danqi Chen, and Luke Zettlemoyer. 2024a. Detecting pretraining data from large language models. In *The Twelfth International Conference on Learning Representations*.

Wentao Shi, Mengqi Yuan, Junkang Wu, Qifan Wang, and Fuli Feng. 2024b. Direct multi-turn preference optimization for language agents. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2312–2324, Miami, Florida, USA. Association for Computational Linguistics.

Jeonghoon Shim, Woojung Song, Cheyon Jin, Seungwon Kook, and Yohan Jo. 2026. Non-collaborative user simulators for tool agents. In *The Fourteenth International Conference on Learning Representations*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024. LLMs in the imaginarium: Tool learning through simulated trial and error. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10583–10604, Bangkok, Thailand. Association for Computational Linguistics.

Shuo Yang, Wei-Lin Chiang, Lianmin Zheng, Joseph E. Gonzalez, and Ion Stoica. 2023. Rethinking benchmark and contamination for language models with rephrased samples. *Preprint*, arXiv:2311.04850.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. 2025. {τ}-bench: A benchmark for \underline{T}ool-\underline{A}gent-\underline{U}ser interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*.

Junjie Ye, Yilong Wu, Songyang Gao, Caishuang Huang, Sixian Li, Guanyu Li, Xiaoran Fan, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024. 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*, pages 313–333, Miami, Florida, USA. Association for Computational Linguistics.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Kan Ren, Dongsheng Li, and Deqing Yang. 2025. EASYTOOL: Enhancing LLM-based agents with concise tool instruction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 951–972, Albuquerque, New Mexico. Association for Computational Linguistics.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Quoc Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Manoj Awalgaonkar, Rithesh R N, Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. 2025. xLAM: A family of large action models to empower AI agent systems. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11583–11597, Albuquerque, New Mexico. Association for Computational Linguistics.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

A PA-Tool

A.1 Detailed Algorithm for PA-Tool

Algorithm 1 Pseudocode for PA-Tool

Require: Language model $\mathcal{M}$, Component description d, samples N, temperature t, hyperparameter α
Ensure: Representative name s^*

// Stage 1: Candidate Generation
$s_{\text{ref}} \leftarrow$ Generate name from $\mathcal{M}$ with temperature 0
$\mathcal{C} \leftarrow \{\}$
for $i = 1$ **to** N **do**
  $s_i \leftarrow$ Generate name from $\mathcal{M}$ with temperature t
  Add s_i to $\mathcal{C}$
end for
// Stage 2: Peakedness Computation
$\ell_{\max} \leftarrow$ maximum character length of name in $\mathcal{C}$
$\tau \leftarrow \alpha \cdot \ell_{\max}$ ▷ Eq. (1)
for each s_i in $\mathcal{C}$ **do**
  $\phi(s_i) \leftarrow$ count of $s_j \in \mathcal{C}$ with $j \neq i$ and edit distance to $s_i \leq \tau$ ▷ Eq. (2)
end for
// Stage 3: Schema Selection
$\mathcal{C}^* \leftarrow$ names in $\mathcal{C}$ with maximum peakedness ▷ Eq. (3)
if $|\mathcal{C}^*| == 1$ **then**
  $s^* \leftarrow$ the unique name in $\mathcal{C}^*$
else
  $s^* \leftarrow$ name in $\mathcal{C}^*$ closest to s_{ref} ▷ Eq. (4)
end if
return s^*

A.2 Example of Schema Generation Process

We provide a concrete example of how PA-Tool generates pretraining-aligned schemas. Given the original tool name `DietTool` with description "A tool that simplifies calorie counting, tracks diet, and provides insights from many restaurants and grocery stores...", PA-Tool generates 32 candidate names by sampling at temperature 0.4.

The top candidates by frequency are:

- `diet_tracker`: 5 occurrences
- `diet_insights`: 4 occurrences
- `calorie_tracker`: 3 occurrences
- `nutri_guide`: 3 occurrences
- `eatwise`: 3 occurrences
- Others: `nutrify`, `nutri_navigator`, `diet_planner`, etc. (1-2 occurrences each)

Importantly, PA-Tool does not simply select the most frequent candidate. Instead, it computes peakedness by measuring how many similar candidates cluster around each option using edit distance. In this case, PA-Tool selects `diet_insights` (peakedness=4) rather than the most frequent diet_tracker (5 occurrences), as the former has a tighter cluster of similar variants indicating stronger distributional concentration.

It is also worth noting that greedy decoding (temperature 0) produces nutri_guide, which differs from both the most frequent candidate and PA-Tool's selection. This illustrates three distinct outcomes: (1) PA-Tool's peakedness-based selection (diet_insights), (2) the most frequent candidate (diet_tracker), and (3) greedy decoding's output (nutri_guide). These differences highlight that PA-Tool's selection mechanism considers distributional concentration rather than simple frequency or single-sample generation.

A.3 Name Collision Resolution

When tools have highly similar descriptions, PA-Tool may generate identical names for different components. We resolve this through iterative priority-based locking: in each round, the component with the highest peakedness for a contested name acquires it, while others cascade to their next candidate. This process repeats until all components have unique names.

Model	MetaTool	RoTBench
Qwen2.5-3B	2 (1.0%)	13 (4.2%)
Qwen2.5-7B	0 (0.0%)	15 (4.8%)
Llama3.2-3B	1 (0.5%)	18 (5.8%)
Llama3.1-8B	1 (0.5%)	17 (5.5%)

Table 7: Name collision statistics across benchmarks.

Table 7 shows collision frequencies across benchmarks. Collisions remain rare across all models (less than 6%), with MetaTool showing particularly low rates (0-1.0%) and RoTBench showing slightly higher but still modest rates (4.2-5.8%). RoTBench's higher collision rate stems from greater lexical overlap among tool descriptions (average Jaccard similarity of 0.025 vs. 0.011 for MetaTool).

A.4 Effect of Hyperparameters

We investigate the effect of PA-Tool's three key hyperparameters on performance: the number of candidates (N), the similarity threshold (α), and the sampling temperature (t). All experiments are conducted on the MetaTool benchmark.

Number of Candidates (N). Table 8 shows that smaller models (3B) achieve stable performance with 16–32 candidates, while larger models (7–8B) require 32–64 candidates before performance

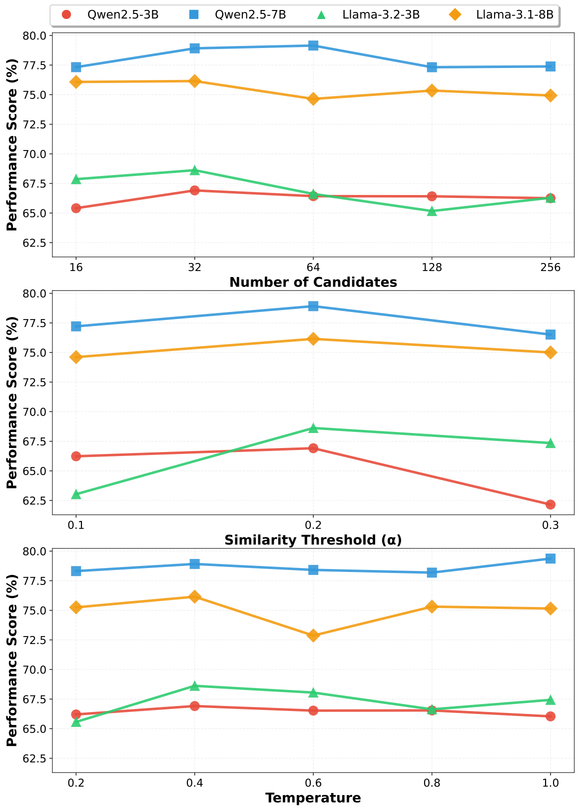

Figure 5: Effect of hyperparameters on PA-Tool across different models. All results are averaged across four MetaTool subtasks. **Top:** Effect of the number of candidates (N). **Middle:** Effect of similarity threshold (α). **Bottom:** Effect of sampling temperature (t).

Model	N	MetaTool			
		Similar	Scenario	Reliab.	Multi.
Qwen2.5-3B	16	50.5	59.1	85.7	66.4
	32	50.0	58.8	**86.2**	**72.6**
	64	**50.9**	**59.1**	86.1	69.6
	128	50.5	58.6	86.0	70.6
	256	50.2	**59.1**	83.2	72.4
Qwen2.5-7B	16	63.2	74.9	86.8	84.3
	32	**64.1**	**78.4**	**88.2**	84.9
	64	62.9	77.7	**88.2**	**87.7**
	128	62.3	77.4	85.4	84.1
	256	63.0	**78.4**	81.0	87.1
Llama3.2-3B	16	64.2	65.9	60.4	**80.9**
	32	**65.7**	**67.7**	**60.6**	80.5
	64	64.3	65.2	57.5	79.5
	128	63.7	65.2	52.5	79.3
	256	64.8	65.5	54.2	80.7
Llama3.1-8B	16	68.9	79.5	**66.9**	**88.9**
	32	**70.4**	**79.9**	66.0	88.3
	64	68.7	77.8	64.5	87.5
	128	67.9	79.4	65.1	**88.9**
	256	67.6	79.8	63.9	88.3

Table 8: Effect of Number of Candidates.

Model	α	MetaTool			
		Similar	Scenario	Reliab.	Multi.
Qwen2.5-3B	0.1	50.5	58.3	84.1	72.0
	0.2	50.0	**58.8**	**86.2**	**72.6**
	0.3	**51.0**	58.6	78.9	60.2
Qwen2.5-7B	0.1	62.7	76.4	84.8	**84.9**
	0.2	**64.1**	**78.4**	**88.2**	**84.9**
	0.3	63.2	75.8	84.7	82.3
Llama3.2-3B	0.1	62.8	66.3	54.8	68.2
	0.2	**65.7**	**67.7**	**60.6**	**80.5**
	0.3	65.0	65.5	59.0	79.9
Llama3.1-8B	0.1	68.3	76.9	65.0	88.1
	0.2	**70.4**	**79.9**	**66.0**	88.3
	0.3	68.1	78.7	64.4	**88.7**

Table 9: Effect of Similarity Threshold.

Model	Temp.	MetaTool			
		Similar	Scenario	Reliab.	Multi.
Qwen2.5-3B	0.2	50.2	58.5	85.0	71.0
	0.4	50.0	58.8	**86.2**	**72.6**
	0.6	51.5	58.9	85.5	70.2
	0.8	50.5	59.7	83.9	72.0
	1.0	**51.7**	**60.2**	85.1	67.2
Qwen2.5-7B	0.2	62.2	75.3	88.8	**86.9**
	0.4	**64.1**	**78.4**	88.2	84.9
	0.6	62.3	75.8	88.6	**86.9**
	0.8	61.4	77.7	87.5	86.1
	1.0	63.1	77.4	**90.2**	86.7
Llama3.2-3B	0.2	64.1	68.3	55.6	74.2
	0.4	**65.7**	67.7	60.6	**80.5**
	0.6	61.3	66.4	**64.6**	79.9
	0.8	63.5	**68.7**	54.1	80.3
	1.0	64.0	66.3	61.5	77.9
Llama3.1-8B	0.2	69.5	79.5	63.5	88.5
	0.4	**70.4**	**79.9**	**66.0**	88.3
	0.6	68.4	79.8	62.3	80.9
	0.8	67.8	79.4	65.2	**88.7**
	1.0	69.5	76.9	65.6	88.5

Table 10: Effect of Sampling Temperature.

plateaus. Beyond these ranges, additional candidates provide minimal gains.

Similarity Threshold (α). Table 9 shows that performance peaks at $\alpha = 0.2$ consistently across all models. At $\alpha = 0.1$, performance drops by 2–3%, while $\alpha = 0.3$ shows similar degradation.

Sampling Temperature (t). Table 10 shows that performance remains stable across temperatures $t \in [0.2, 1.0]$, varying within 1–2%. Moderate temperatures ($t = 0.4$–0.6) show slightly better results, though the differences are marginal.

B Evaluation Protocol

B.1 MetaTool

In MetaTool evaluation, the agent receives (1) a user query, (2) a list of candidate tools with their names and descriptions, and (3) few-shot examples. The agent must select the appropriate tool name(s) from the provided list. For Similar, Scenario, and Multi-tool subtasks, the agent selects one or more ground-truth tools; for Reliability, the agent must output "None" when no suitable tool exists. Models must reason about and distinguish between all available tools in the candidate list, regardless of their names. The prompt is shown in Figure 8.

Scoring Methodology. We revised the original MetaTool scoring methodology, which relied heavily on manual analysis and did not adequately handle ambiguous cases where models produced partially correct outputs. For single-tool tasks (Similar and Scenario), when exactly one tool match is found, we verify that it corresponds to the ground-truth label. If the keyword "None" also appears, the response is marked correct only when the matched tool appears before "None", as the MetaTool prompt requires the selected tool to be stated first. For cases with two or more matches, we replaced manual analysis with automated evaluation using GPT-4.1-mini, which assesses whether the model's output selected only the ground-truth tools. For Reliability tasks, we added logic to verify correct "None" predictions: a response is marked correct if no matches are found and "None" is present, or if matches exist but "None" appears before them. This distinguishes appropriate rejections from cases where models incorrectly included tool names alongside "None". For multi-tool tasks, responses with exactly one match are now explicitly marked incorrect, as these tasks require selecting multiple tools. Cases with more than two matches are evaluated using GPT-4.1-mini. These refinements enable consistent and scalable automated scoring across all instances, which was difficult to achieve with the original manual methodology.

B.2 RoTBench

In RoTBench evaluation, the agent receives (1) a system message describing the task format, (2) a list of available tools with complete schemas including names, descriptions, and parameter specifications, and (3) a user query. The agent must provide both tool selection and parameter identification in a structured format. The prompt is shown in Figure 7.

Evaluation Metrics. Tool Selection accuracy is measured by whether the model selects the correct tool name from the available list. Parameter Identification accuracy is computed conditionally: only cases where the model correctly selected the tool are evaluated for whether the model also identified the correct set of required parameters. This two-stage evaluation isolates the impact of schema alignment on each capability, while preserving the multi-candidate selection challenge where models must reason across multiple tool options.

B.3 API-Bank

API-Bank (Li et al., 2023) is a comprehensive benchmark for evaluating tool-augmented language models across realistic API usage scenarios. The benchmark implements 73 runnable APIs spanning diverse domains and evaluates models on 314 manually annotated dialogues containing 753 API calls.

API-Bank assesses two fundamental capabilities: (1) **Call**, where models must correctly invoke APIs with appropriate parameters when APIs are provided in the context, and (2) **Retrieval+Call**, where models must first retrieve relevant APIs from a large pool using an API Search function, then call the selected APIs. Both tasks require exact matching of tool names and parameters with ground-truth annotations for correctness. We evaluate on both capabilities to assess whether PA-Tool improves tool selection when schemas are either explicitly provided (Call) or must be retrieved (Retrieval+Call). The prompt is shown in Figure 9.

B.4 τ-Bench

τ-Bench (Yao et al., 2025) evaluates agents on realistic, multi-turn task-oriented dialogues where they must interact with simulated users while following domain-specific policies. Unlike traditional benchmarks that provide all information upfront, τ-Bench requires agents to incrementally gather information through conversation, consult policy guidelines, and execute appropriate API calls to reach a target database state.

We evaluate on the Retail domain, which contains realistic customer service scenarios. Success requires: (1) correctly invoking APIs with write capabilities (e.g., `cancel_order`, `update_shipping`) to modify the database state, (2) adhering to domain-specific policies, and (3) managing multi-turn interactions to gather necessary information from users. The benchmark measures end-to-end task completion by comparing the final database state with the ground-truth state, directly assessing whether improved tool selection translates to successful task completion. We fix the user simulator to Llama3.1-8B-Instruct and use temperature 0 for deterministic evaluation, reporting the mean pass rate over 5 independent runs (N=5) to account for stochasticity in user simulation. The prompt is shown in Figure 10.

B.5 PA-Tool Integration

Within these evaluation protocols, PA-Tool operates by replacing original tool names and parameter names in the provided schemas with pretraining-aligned alternatives. Critically, all other components remain identical: tool descriptions, parameter descriptions, example demonstrations, and task instructions are unchanged. This ensures that PA-Tool's improvements stem solely from schema alignment, not from modifications to semantic information or task structure. For instance, if the original schema contains `get_weather_info` but PA-Tool determines that `get_weather_forecast` better aligns with the model's pretrained knowledge, only the tool name in the tool schema is replaced. The tool's description, available parameters, and all contextual information remain identical across Base and PA-Tool conditions, ensuring fair and comparable evaluation.

C Generalization to Diverse Models Details

To assess whether PA-Tool generalizes across different model families and scales, we conduct additional experiments on MetaTool and RoTBench using eight models beyond the Qwen2.5 and Llama families examined in our main results, bringing the total to 12 models across 5 families. We group these models into three categories. (1) **SLMs**: Ministral-8B, GPT-4.1-nano, and Gemini-2.5-Flash-Lite; (2) **LLMs**: Llama3.3-70B, GPT-4.1-mini, and Gemini-2.5-Flash, to test whether PA-Tool remains effective at larger scales; and (3) **reasoning models**: Qwen3-1.7B and Qwen3-4B in thinking mode, to examine whether inference-time chain-of-thought reasoning can independently resolve schema misalignment.

Results on SLMs. Consistent with our main findings, PA-Tool improves performance over the Base

Model	Method	MetaTool				RoTBench			
		Tool Selection				Single-turn		Multi-turn	
		Similar	Scenario	Reliability	Multi-tool	Tool Sel.	Param Iden.	Tool Sel.	Param Iden.
Small Language Models									
Ministral-8B	Base	65.3	81.2	42.5	**93.1**	66.7	28.6	**51.4**	38.6
	PA-Tool	**70.5**	**82.6**	**44.7**	92.4	**75.2**	**31.4**	**51.4**	**41.4**
GPT-4.1-nano	Base	56.8	53.1	99.0	**81.5**	71.4	34.3	**51.4**	**44.3**
	PA-Tool	**60.6**	**54.0**	**99.5**	77.1	**72.4**	**44.8**	47.1	42.9
Gemini-2.5-Flash-Lite	Base	79.3	86.5	65.9	**77.7**	72.4	63.8	**65.7**	**58.6**
	PA-Tool	**80.8**	**87.1**	**70.5**	72.0	**73.3**	**65.7**	64.3	**58.6**
Large Language Models									
Llama3.3-70B	Base	**80.4**	**85.2**	63.7	**82.9**	**75.2**	**27.6**	60.0	47.1
	PA-Tool	79.9	**85.2**	**66.2**	81.3	74.3	26.7	**64.3**	**48.6**
GPT-4.1-mini	Base	79.6	84.3	76.3	72.2	79.1	58.1	**71.4**	**61.4**
	PA-Tool	**80.5**	**84.5**	**80.0**	**84.3**	**82.9**	**59.1**	70.0	**61.4**
Gemini-2.5-Flash	Base	70.0	79.8	89.2	77.3	82.9	56.2	60.0	54.3
	PA-Tool	**72.6**	**82.7**	**92.4**	**78.5**	**84.8**	**62.9**	**72.9**	**64.3**
Reasoning Models (thinking mode)									
Qwen3-1.7B	Base	66.2	**73.2**	93.2	**76.1**	58.1	32.4	38.6	**38.6**
	PA-Tool	**68.7**	69.8	**96.6**	73.2	**62.9**	**42.9**	**45.7**	**38.6**
Qwen3-4B	Base	72.0	**81.9**	84.2	**88.1**	65.7	**30.5**	57.1	52.9
	PA-Tool	**73.5**	81.2	**91.2**	86.5	**69.5**	27.6	**65.7**	**61.4**

Table 11: Performance comparison on MetaTool and RoTBench with additional models, grouped by model type. **Bold** indicates the better result between Base and PA-Tool per row group.

in most settings. Gains are most pronounced for Ministral-8B, with a +8.5% improvement on RoTBench single-turn tool selection (66.7%→75.2%) and +5.2% on MetaTool Similar (65.3%→70.5%). For GPT-4.1-nano and Gemini-2.5-Flash-Lite, improvements are more task-dependent, with notable gains on MetaTool Reliability (+4.6% for Gemini) and RoTBench parameter identification (+10.5% for GPT-4.1-nano), though gains on other subtasks are smaller or absent.

Results on LLMs. PA-Tool also improves larger models, though average gains are smaller than for SLMs, consistent with reduced prevalence of schema misalignment at larger scales. Gains remain substantial where misalignment compounds, such as multi-tool composition, where GPT-4.1-mini improves by +12.1% on MetaTool Multi-tool (72.2%→84.3%). Llama3.3-70B shows the most modest improvements, suggesting that some larger open-source models may already handle misaligned schemas reasonably well.

Results on Reasoning Models. Despite strong baselines from chain-of-thought reasoning (e.g., Qwen3-1.7B Reliability: 93.2%), PA-Tool yields gains of up to +10.5% on RoTBench single-turn parameter identification and +7.1% on multi-turn tool selection for Qwen3-1.7B. Slight decreases on near-saturated subtasks (e.g., Qwen3-4B Multi-tool: 88.1%→86.5%) mirror patterns observed for larger non-reasoning models, where baselines are already strong. These results confirm that reasoning and schema alignment address complementary failure modes rather than competing solutions.

Together, these results demonstrate that PA-Tool generalizes across diverse model families, architectures, and scales.

D Integration with Supervised Fine-tuning Details

Setup. As MetaTool does not include a predefined training split, we construct train/validation /test sets by randomly sampling 60%/20%/20% of all task instances. Since the dataset provides only ground-truth labels without reasoning traces, we use GPT-4.1-mini to generate reasoning trajectories for each training instance (prompt in Figure 12), which we use to train the SFT1 baseline. SFT2 doubles this training data to test whether PA-Tool remains effective under larger training sets. We fine-tune Llama3.1-8B with the hyperparameters in Table 13, and additionally fine-tune Qwen2.5-3B and Qwen2.5-7B under the SFT1 configuration to verify generalization across model families.

Model	Config	Similar	Scenario	Reliab.	Multi.
Qwen2.5-3B	Base	50.3	61.2	84.9	63.6
	PA-Tool	50.8	**64.4**	83.9	68.7
	SFT	50.8	62.3	85.4	64.6
	SFT + PA-Tool	**51.8**	59.8	**89.5**	**77.8**
Qwen2.5-7B	Base	**61.8**	**80.3**	80.9	79.8
	PA-Tool	60.8	78.6	84.9	85.9
	SFT	60.8	80.2	81.9	82.8
	SFT + PA-Tool	**61.8**	78.6	**86.9**	**86.9**

Table 12: SFT comparison on Qwen2.5 models.

Results. Table 12 presents the performance on Qwen2.5 models. The results confirm the key findings from Llama3.1-8B (Table 3): SFT strengthens tool-use reasoning through training data, while PA-Tool resolves naming-level misalignment that persists even after fine-tuning. SFT + PA-Tool achieves the best results on most subtasks (e.g., Qwen2.5-3B Reliability: 89.5%, Multi-tool: 77.8%), generalizing the complementary gains observed on Llama3.1-8B to the Qwen2.5 family.

Hyperparameter	Value
LoRA Rank	32
LoRA Alpha	64
LoRA Dropout	0.05
Learning Rate	5e-5
Batch Size	16
Epochs	5
GPU	NVIDIA A100 80GB PCIe

Table 13: Training hyperparameters for SFT.

E Error Analysis Details

Error Taxonomy. We categorize tool selection errors into three types based on their underlying failure modes:

(1) Schema Misalignment Error: the model generates a plausible but non-existent tool name, following its pretrained naming conventions rather than the provided schema.

(2) Functional Confusion Error: the model selects an existing tool whose functionality is similar to the correct one, indicating it understands the query but confuses related tools. For instance, when asked to send an email notification, the model might select `send_sms` instead of `send_email`.

(3) Context Understanding Error: the model selects a functionally unrelated tool, indicating a failure to comprehend the query's intent. For example, when asked to delete a user account, the model might select `create_user` or `list_products`.

Experimental Setup. We use GPT-4.1-mini (OpenAI, 2025) as an error analyzer to classify errors from Llama3.1-8B in both Base and PA-Tool configurations. We analyze MetaTool's three tool selection subtasks—Similar, Scenario, and Multi-tool—excluding Reliability as it specifically tests the "no suitable tool" scenario rather than tool selection errors. The prompt is provided in Figure 11.

F Computational Time

Model	MetaTool	RoTBench
Qwen2.5-3B	8.0 sec	13.9 sec
Qwen2.5-7B	11.2 sec	15.0 sec
Llama3.2-3B	8.2 sec	12.4 sec
Llama3.1-8B	11.6 sec	15.9 sec

Table 14: Schema generation time for PA-Tool.

Table 14 shows the wall-clock time required for one-time schema generation on both MetaTool (199 tools) and RoTBench (568 tools). Schema generation is a one-time preprocessing cost, requiring 8–16 seconds depending on model size and number of tools. Once generated, the pretraining-aligned schema can be reused indefinitely without additional overhead during inference. All experiments were conducted on Intel Xeon Gold 6230 CPU @ 2.10GHz (38 cores), NVIDIA A100 80GB PCIe, 450GB RAM, with Python 3.11.14, vLLM 0.10.2, and CUDA 12.8.

G Integration with Training-free Methods Details

G.1 Retrieval-based Correction

Retrieval-based methods recover from schema misalignment by mapping invalid model outputs to valid tool names post-hoc. When a model prediction is not found in the candidate list (case-insensitive check), we retrieve the nearest valid tool; predictions already within the candidate list are left unchanged, even if semantically incorrect. This distinction is important: retrieval can only recover schema-violating outputs, not wrong selections among valid tools.

BM25. We retrieve the most similar valid tool using BM25 scoring (Robertson and Zaragoza, 2009) based on lexical overlap.

ToolLLM Embedding. We employ ToolLLM's tool-specialized embedding model (Qin et al., 2024) and retrieve the nearest valid tool in embedding space.

Model	Method	MetaTool				RoTBench			
		Tool Selection				Single-turn		Multi-turn	
		Similar	Scenario	Reliability	Multi-tool	Tool Sel.	Param Iden.	Tool Sel.	Param Iden.
Qwen2.5-3B	Base	48.7	55.3	83.6	**75.1**	12.4	7.6	10.0	10.0
	BM25	48.7	55.3	83.6	**75.1**	14.3	8.6	12.9	11.4
	ToolLLM	48.7	55.3	83.7	**75.1**	4.8	1.9	12.9	11.4
	PA-Tool	50.0	58.8	**86.2**	72.6	18.1	10.5	15.7	14.3
	PA-Tool + BM25	**50.3**	60.2	**86.2**	72.8	**21.9**	**13.3**	20.0	17.1
	PA-Tool + ToolLLM	**50.3**	**60.3**	**86.2**	73.4	20.9	**13.3**	**21.4**	**20.0**
Qwen2.5-7B	Base	59.6	74.4	78.3	78.3	49.5	20.0	21.4	21.4
	BM25	59.8	74.6	78.3	78.3	49.5	20.9	21.4	21.4
	ToolLLM	59.8	74.6	78.3	78.3	49.5	20.9	21.4	21.4
	PA-Tool	**64.1**	78.4	**88.2**	84.9	55.2	**21.9**	**27.1**	**22.9**
	PA-Tool + BM25	**64.1**	**78.5**	**88.2**	84.9	**56.2**	**21.9**	**27.1**	**22.9**
	PA-Tool + ToolLLM	**64.1**	**78.5**	**88.2**	**85.5**	55.2	**21.9**	**27.1**	**22.9**
Llama3.2-3B	Base	55.0	58.6	43.6	79.1	56.2	20.0	32.9	27.1
	BM25	55.7	58.7	43.6	79.1	56.2	20.9	32.9	27.1
	ToolLLM	55.4	58.8	43.6	79.1	56.2	**22.9**	32.9	27.1
	PA-Tool	**65.7**	**67.7**	**60.6**	**80.5**	62.9	21.9	**34.3**	**28.6**
	PA-Tool + BM25	**65.7**	**67.7**	**60.6**	**80.5**	**63.8**	21.9	**34.3**	**28.6**
	PA-Tool + ToolLLM	**65.7**	**67.7**	**60.6**	**80.5**	62.9	21.9	**34.3**	**28.6**
Llama3.1-8B	Base	61.5	73.9	53.5	78.7	58.1	17.1	42.8	34.3
	BM25	61.8	74.0	53.5	78.7	58.1	17.1	42.8	34.3
	ToolLLM	61.6	74.0	53.5	78.7	58.1	17.1	42.8	34.3
	PA-Tool	**70.4**	**79.9**	**66.0**	**88.3**	**68.6**	**18.1**	**48.6**	35.7
	PA-Tool + BM25	**70.4**	**79.9**	**66.0**	**88.3**	**68.6**	**18.1**	**48.6**	**37.1**
	PA-Tool + ToolLLM	**70.4**	**79.9**	**66.0**	**88.3**	**68.6**	**18.1**	**48.6**	**37.1**

Table 15: Performance of retrieval-based post-hoc correction methods (BM25, ToolLLM) on MetaTool and RoTBench.

Results. As shown in Table 15, retrieval-based correction yields limited improvements on its own. On MetaTool, BM25 and ToolLLM produce marginal or no gains in most subtasks, with accuracy improvements rarely exceeding 1%. On RoTBench, gains are slightly larger but remain inconsistent across models and settings. When combined with PA-Tool, however, retrieval methods yield further improvements on top of PA-Tool's gains, indicating that schema alignment and post-hoc retrieval address different aspects of tool-use and can be combined for additional benefit.

G.2 Constrained Generation

Constrained generation methods enforce valid outputs during the decoding process, preventing schema-violating generations.

JSON Schema Constraints. We implement constrained decoding using JSON Schema with enum restrictions, which masks logits corresponding to tokens that would lead to invalid tool names during generation. For MetaTool, the schema restricts outputs to `{"selected_tool": <enum>}`, while for RoTBench it enforces `{"action": <enum>, "action_input": <object>}`, where the enum contains all valid tool names plus special tokens (e.g., `None`, `finish`).

Results. As shown in Table 16, constrained generation substantially improves RoTBench performance by eliminating format errors (e.g., Qwen2.5-7B single-turn tool selection: 49.5%→78.1%). On MetaTool, however, results are more mixed: constrained generation sometimes underperforms PA-Tool alone, and can even fall below Base on Reliability (e.g., Qwen2.5-3B: 83.6%→60.9%), likely because enum restrictions bias the model toward emitting a tool even when none is appropriate. Combining constrained generation with PA-Tool (Constrained-PA) mitigates this: the approach retains format guarantees while leveraging PA-Tool's schema alignment, yielding the best parameter identification results on most models (e.g., Qwen2.5-3B single-turn: 16.2%→26.7%). This suggests that PA-Tool's aligned representations help the model select correctly even when the output space is explicitly constrained.

G.3 Description Enhancement

Description enhancement methods improve tool understanding by rewriting documentation for clarity,

Model	Method	MetaTool				RoTBench			
		Tool Selection				Single-turn		Multi-turn	
		Similar	Scenario	Reliability	Multi-tool	Tool Sel.	Param Iden.	Tool Sel.	Param Iden.
Qwen2.5-3B	Base	48.7	55.3	83.6	75.1	12.4	7.6	10.0	10.0
	PA-Tool	50.0	58.8	86.2	72.6	18.1	10.5	15.7	14.3
	Constrained	65.9	**80.3**	60.9	**77.3**	**65.7**	16.2	**42.9**	**32.9**
	Constrained-PA	**66.1**	77.9	**74.5**	76.5	61.9	**26.7**	37.1	30.0
Qwen2.5-7B	Base	59.6	74.4	78.3	78.3	49.5	20.0	21.4	21.4
	PA-Tool	64.1	78.4	**88.2**	**84.9**	55.2	21.9	27.1	22.9
	Constrained	64.2	**81.0**	78.4	83.3	**78.1**	**31.4**	**71.4**	51.4
	Constrained-PA	**67.9**	79.3	84.0	84.1	76.2	**31.4**	67.1	**54.3**
Llama3.2-3B	Base	55.0	58.6	43.6	79.1	56.2	20.0	32.9	27.1
	PA-Tool	**65.7**	67.7	60.6	80.5	62.9	**21.9**	34.3	28.6
	Constrained	55.1	**70.2**	**72.3**	**89.5**	62.9	20.0	41.4	31.4
	Constrained-PA	59.2	69.8	65.5	86.9	**68.6**	**21.9**	**44.3**	**40.0**
Llama3.1-8B	Base	61.5	73.9	53.5	78.7	58.1	17.1	42.8	34.3
	PA-Tool	**70.4**	79.9	66.0	**88.3**	**68.6**	18.1	**48.6**	**35.7**
	Constrained	66.3	**81.7**	66.2	74.3	64.8	16.2	44.3	**35.7**
	Constrained-PA	69.8	81.2	**66.3**	79.9	63.8	**20.0**	45.7	**35.7**

Table 16: Performance of **Constrained** generation using JSON Schema enum on MetaTool and RoTBench

Model	Method	MetaTool				RoTBench			
		Tool Selection				Single-turn		Multi-turn	
		Similar	Scenario	Reliability	Multi-tool	Tool Sel.	Param Iden.	Tool Sel.	Param Iden.
Qwen2.5-3B	Base	48.7	55.3	83.6	**75.1**	12.4	7.6	10.0	10.0
	PA-Tool	50.0	58.8	86.2	72.6	18.1	10.5	15.7	**14.3**
	EasyTool	**52.3**	**64.3**	91.5	70.6	19.1	12.4	14.3	11.4
	EasyTool+PA-Tool	48.0	47.7	**93.4**	46.1	**22.9**	**19.1**	**17.1**	**14.3**
Qwen2.5-7B	Base	59.6	74.4	78.3	78.3	49.5	20.0	21.4	21.4
	PA-Tool	64.1	78.4	88.2	84.9	55.2	21.9	**27.1**	22.9
	EasyTool	**65.1**	**78.9**	83.0	**85.1**	50.5	**26.7**	25.7	24.3
	EasyTool+PA-Tool	64.5	73.5	**90.5**	79.3	**52.4**	**26.7**	**27.1**	**25.7**
Llama3.2-3B	Base	55.0	58.6	43.6	79.1	56.2	20.0	32.9	27.1
	PA-Tool	**65.7**	**67.7**	60.6	**80.5**	62.9	21.9	34.3	28.6
	EasyTool	45.3	65.3	64.1	79.5	56.2	20.9	35.7	28.6
	EasyTool+PA-Tool	42.0	58.3	**80.5**	66.6	**66.7**	**23.8**	**38.6**	**30.0**
Llama3.1-8B	Base	61.5	73.9	53.5	78.7	58.1	17.1	42.8	34.3
	PA-Tool	**70.4**	**79.9**	66.0	**88.3**	**68.6**	18.1	**48.6**	35.7
	EasyTool	61.1	77.2	**76.9**	72.2	59.0	14.3	40.0	31.4
	EasyTool+PA-Tool	66.2	77.9	73.8	68.6	64.8	**21.9**	41.4	**37.1**

Table 17: Performance of description enhancement (**EasyTool**) on MetaTool and RoTBench

without modifying the tool naming schema.

EasyTool. EasyTool (Yuan et al., 2025) creates unified tool instructions by restructuring verbose API documentation into clear, consistent formats. We adapt this methodology with two modifications. First, to ensure fair comparison without introducing external model capabilities, we use the *target model itself* rather than GPT-4 to generate descriptions. Second, we simplify the output to concise 1–2 sentence descriptions with action-oriented verbs (e.g., "Retrieves", "Calculates"). The full prompt template follows EasyTool's principles of removing redundancy and improving clarity (Figure 13).

Results. As shown in Table 17, PA-Tool and EasyTool show complementary strengths across models. On Llama models, PA-Tool substantially outperforms EasyTool (e.g., Llama3.2-3B Similar: 65.7% vs. 45.3%; Llama3.1-8B Multi-tool: 88.3% vs. 72.2%), suggesting that naming alignment drives larger gains than description enhancement for these models. On Qwen models, EasyTool is competitive and sometimes stronger, particularly for Qwen2.5-3B on MetaTool subtasks (e.g., Scenario: 64.3% vs. 58.8%), indicating that different models benefit from different axes of schema improvement. Importantly, the two approaches are

Target	Schema Source	Similar	Scenario	Reliab.	Multi.
Llama3.2-3B	Base (no PA-Tool)	55.0	58.6	43.6	79.1
	Qwen2.5-7B	61.6	59.3	**65.5**	80.7
	Qwen2.5-3B	58.4	60.2	67.6	79.3
	Llama3.1-8B	63.3	65.9	67.5	**82.7**
	Self	**65.7**	**67.7**	60.6	80.5
Llama3.1-8B	Base (no PA-Tool)	61.5	73.9	53.5	78.7
	Qwen2.5-7B	69.6	77.4	54.1	83.1
	Qwen2.5-3B	68.9	78.2	54.7	82.5
	Llama3.2-3B	69.2	79.2	51.2	85.7
	Self	**70.4**	**79.9**	**66.0**	**88.3**

Table 18: Cross-model schema transfer on MetaTool. *Self* denotes the target model's own PA-Tool schema.

compatible: combining EasyTool with PA-Tool achieves the best Reliability scores on three of four models (e.g., Qwen2.5-3B: 93.4%), and yields the strongest overall parameter identification on RoTBench (e.g., Qwen2.5-3B single-turn: 19.1%), showing that name alignment and description clarity address orthogonal aspects of schema quality.

H Cross-Model Schema Transfer

PA-Tool generates model-specific schemas, raising a practical question: in multi-agent systems with heterogeneous SLMs, must each model use its own schema? We investigate cross-model schema transfer by evaluating target models with schemas generated by different source models.

Results. Table 18 presents cross-model schema transfer results on MetaTool. We observe three findings. First, cross-model schemas improve over unaligned baselines, even across model families (e.g., Llama3.1-8B with Qwen2.5-7B schemas achieves 69.6% on Similar, +8.1% over Base), suggesting partially overlapping naming conventions due to shared training corpora. Second, self-generated schemas are optimal in most settings, but cross-model gaps are task-dependent: on Similar and Scenario, gaps are within 1–3%, while Reliability shows larger gaps (e.g., 54.1% vs. 66.0% for Llama3.1-8B), indicating that tasks requiring precise schema understanding are more sensitive to model-specific alignment. Third, larger models' schemas can benefit smaller models within the same family; Llama3.2-3B with Llama3.1-8B schemas achieves the highest scores on Multi-tool among cross-model configurations.

We recommend generating model-specific schemas given the negligible one-time cost (Appendix F). When this is infeasible, cross-model schemas—particularly from a larger model in the same family—provide a viable fallback that still outperforms unaligned baselines.

Model	Level	Single-turn Tool Sel.	Single-turn Param Id.	Multi-turn Tool Sel.	Multi-turn Param Id.
Qwen2.5-3B	Clean	12.4	7.6	10.0	10.0
	Slight	13.3	6.2	7.9	6.4
	Medium	11.4	6.7	7.1	5.0
	Heavy	**21.9**	5.7	9.3	5.0
	Union	8.6	3.8	10.0	4.3
	PA-Tool	18.1	**10.5**	**15.7**	**14.3**
Qwen2.5-7B	Clean	49.5	20.0	21.4	21.4
	Slight	33.3	12.9	20.0	18.6
	Medium	28.1	10.0	16.4	15.0
	Heavy	47.6	12.9	**27.9**	18.6
	Union	41.9	17.1	25.7	20.0
	PA-Tool	**55.2**	**21.9**	27.1	**22.9**
Llama3.2-3B	Clean	56.2	20.0	32.9	27.1
	Slight	34.8	10.5	20.0	13.6
	Medium	31.9	10.5	17.1	10.7
	Heavy	47.6	11.9	23.6	12.9
	Union	50.5	16.2	21.4	14.3
	PA-Tool	**62.9**	**21.9**	**34.3**	**28.6**
Llama3.1-8B	Clean	58.1	17.1	42.8	34.3
	Slight	37.1	8.1	27.9	22.1
	Medium	31.9	8.1	25.7	21.4
	Heavy	48.6	10.0	31.4	23.6
	Union	46.7	12.4	40.0	25.7
	PA-Tool	**68.6**	**18.1**	**48.6**	**35.7**

Table 19: Performance across RoTBench noise levels.

I Robustness Across Noise Levels

A central premise of PA-Tool is that SLMs rely heavily on tool *names*, not just descriptions, when making tool-use decisions. To stress-test this premise and demonstrate PA-Tool's robustness, we evaluate all four main models across the five noise environments defined by RoTBench (Ye et al., 2024): **Clean** (original schemas), **Slight** (character-level insertions, omissions, and substitutions), **Medium** (names replaced with reversed strings or random characters such as "abc"), **Heavy** (names shuffled across tools, creating anti-semantic assignments), and **Union** (a random combination of the above perturbations to both tool names and parameters). PA-Tool generates schemas solely from descriptions, so its output is identical regardless of the original schema's noise level.

Results. Table 19 reports single-turn and multi-turn results across all noise levels. Three patterns emerge, together motivating PA-Tool's role.

First, **tool names are primary selection cues**. For most models, even the mildest perturbation (Slight) causes large drops in single-turn tool selection despite descriptions remaining intact (e.g., Llama3.1-8B: 58.1% $\rightarrow$ 37.1%; Llama3.2-3B: 56.2% $\rightarrow$ 34.8%).

Second, **not all names are equally useful: plausibility matters, but alignment matters more**.

Medium noise, where names become semantically meaningless, produces the most severe drops (e.g., Llama3.1-8B: 58.1% $\rightarrow$ 31.9%), while Heavy noise shows partial recovery (e.g., 58.1% $\rightarrow$ 48.6%) because shuffled names still preserve real-world naming patterns even when attached to the wrong tools. Models thus benefit from *plausible* names, but *correctly aligned* names are needed for full performance—and human-designed Clean schemas are not guaranteed to provide them.

Third, **PA-Tool provides such alignment, surpassing the Clean baseline on nearly all settings**. Single-turn tool-selection gains reach +5.7 to +10.5 points on the three stronger models (e.g., Llama3.1-8B: 58.1% $\rightarrow$ 68.6%; Qwen2.5-7B: 49.5% $\rightarrow$ 55.2%; Llama3.2-3B: 56.2% $\rightarrow$ 62.9%), with consistent improvements on multi-turn settings and parameter identification. Since PA-Tool generates names from descriptions alone, it yields the same pretraining-aligned schema regardless of input quality, providing a stable performance floor for real-world deployments where tool schemas may be inconsistently named or poorly documented.

J AI Assistants in Research or Writing

We used AI assistants to refine writing, proofread the text, and assist with coding experiments. However, all core ideas, experimental design, analysis, and scientific contributions are entirely the work of the authors.

K Potential Risks

We acknowledge two potential risks associated with deploying PA-Tool. First, because PA-Tool explicitly aligns tool schemas with patterns internalized during pretraining, it may preserve and reinforce biased, culturally narrow, or English-centric naming conventions present in the pretraining corpus; practitioners deploying PA-Tool on multilingual or domain-specific tool ecosystems may need to audit the generated names for cultural appropriateness and inclusivity. Second, although our human evaluation (§6.7) shows that PA-Tool names are generally rated as more understandable than the originals, a minority of cases (~10%) yield names that human developers find less clear, and a human-in-the-loop review of the generated schema mapping may be warranted before deployment in safety-critical applications.

L Prompt Templates

Candidate Name Generation Prompt

Generate a `{{ component }}` name from the description below.
The `{{ component }}` will be used in a tool agent scenario.

Description:
`{{ description }}`

If component == "tool":

Example:
Description: A tool that manages files and directories on the system.
Output: `file_manager`

Generate only the name without additional explanation.

Elif component == "parameter":

Example:
Context:
Tool: `file_manager` - A tool for managing files and directories
Output: `file_path`

Context:
Tool: `{{ tool_name }}` - {{ tool_description }}
Generate only the name without additional explanation.

Figure 6: Candidate name generation prompt.

RoTBench Inference Prompt

System: You are an expert in using tools. At each step, analyze the state and decide the next action with a function call.

Available tools:
```
[{"name": "get_translation_nllb",
  "description": "Translate text using NLLB model.",
  "parameters": {"input_text": {...},
"tgt_lang": {...},
   "src_lang": {...}, ...}},
 {"name": "get_translation_baidu",
  "description": "Translate using BAIDU API.",
  "parameters": {"text": {...},
"tgt_lang": {...}, ...}},...]
```

User: "What is the translation of 'See you later' in Japanese?"

Expected output:
Thought: Need to translate English to Japanese
Action: get_translation_nllb
Action Input: {"input_text": "See you later", "tgt_lang": "jpn_Jpan", "src_lang": "eng_Latn"}

Figure 7: RoTBench inference prompt with system message, tool schemas, and structured output format.

MetaTool Inference Prompt

You are a helpful AI assistant. Your current task is to choose the appropriate tool to solve the user's query based on their question. I will provide you with the user's question and information about the tools. If there is a tool in the list that is applicable to this query, please return the name of the tool (you can only choose one tool). If there isn't, please return 'None.' Additionally, you will need to support your answer with a brief explanation.

User's Query: [User's Query Start] Planning a beach day next week, while considering the ideal weather conditions, availability of beach accessories, preferred beach location, potential activities to engage in like swimming, sunbathing, or playing beach games, and coordinating with friends or family members to join in the fun. [User's Query End]

List of Tools with Names and Descriptions: [List of Tools with Names and Descriptions Start]

1. tool name: airqualityforeast, tool description: ['Planning something outdoors? Get the 2-day air quality forecast for any US zip code.']

2. tool name: WeatherTool, tool description: Provide you with the latest weather information.

3. tool name: AusSurfReport, tool description: ['Get today's surf report for any break throughout Australia!']

   ...

10. tool name: EarthquakeTool, tool description: Provides real-time earthquake notifications and news.

[List of Tools with Names and Descriptions End]

Here are some examples: [Examples Start]
query: "I'm planning a hiking trip next week. What will the weather be like in the Grand Canyon?" tool: WeatherTool
query: "I'm a beginner surfer. Can you suggest a beach with mild waves for me to practice in Australia?" tool: AusSurfReport
...
[Examples End]

User query: "Planning a beach day next week, while considering the ideal weather conditions..."
tool:

Figure 8: MetaTool inference prompt with task instructions, candidate tools, and few-shot examples.

API-Bank Inference Prompt

Instruction: Based on the given API description and the existing conversation history 1..t, please generate the API request that the AI should call in step t+1 and output it in the format of [ApiName(key1='value1', key2='value2', ...)], replace the ApiName with the actual API name, and replace the key and value with the actual parameters.

Your output should start with a square bracket "[" and end with a square bracket "]". Do not output any other explanation or prompt or the result of the API call in your output.

This year is 2023.

API descriptions:
```
[{"name": "GetWeather",
  "description": "Get current weather
information",
  "parameters": {"location": {"type":
"string",
   "description": "City name"},
  "unit": {"type": "string",
   "enum": ["celsius", "fahrenheit"]}}},
 ...]
```

Input:
User: What's the weather like in San Francisco?
AI: I'll check the current weather in San Francisco for you.

Expected output:
```
[GetWeather(location='San Francisco',
unit='fahrenheit')]
```

Figure 9: API-Bank inference prompt with API call generation instructions, schemas, and input dialogue.

τ-bench Inference Prompt

Instruction: You need to act as an agent that uses the above tools to help the user according to the above policy. At each step, your generation should have exactly the following format:

```
Thought: <A single line of reasoning to
process the context
  and inform the decision making. Do not
include extra lines.>
Action: {"name": <The name of the action>,
  "arguments": <The arguments to the action
in json format>}
```

The Action will be parsed, so it must be valid JSON. You should not use made-up or placeholder arguments.

Available tools:

```
[{"type": "function",
  "function": {"name":
"get_current_weather",
   "description": "Get the current
weather",
   "parameters": {"type": "object",
    "properties": {"location": {"type":
"string",
     "description": "The city and state,
e.g. San Francisco, CA"},
    "format": {"type": "string", "enum":
["celsius", "fahrenheit"],
     "description": "The temperature unit
to use. Infer this
      from the users location."}},
    "required": ["location", "format"]}}},
 ...]
```

User: "I want to know the current weather of San Francisco"

Expected output:
Thought: Since the user asks for the weather of San Francisco in USA, the unit should be in fahrenheit. I can query get_current_weather to get the weather.
Action: {"name": "get_current_weather", "arguments": {"location": "San Francisco, CA", "format": "fahrenheit"}}

[If tool returns "70F"]
Thought: I can answer the user now.
Action: {"name": "respond", "arguments": {"response": "The current weather of San Francisco is 70F."}}

Figure 10: τ-Bench inference prompt with structured agent instructions, tool schemas, and multi-step reasoning examples.

Error Classification Prompt

You are an expert in analyzing tool selection errors in language models.

Given a query, the ground truth tool(s), the model's selected tool, and the available tool list, classify the error into ONE of these three categories:

1. Schema Misalignment Error: The model selected a tool name that seems plausible but doesn't exist in the tool list, OR the model selected a tool with a similar name pattern but different from the correct tool. This indicates the model is following its pretrained naming patterns rather than the actual schema.

2. Functional Confusion Error: The model selected an existing tool from the list that has similar functionality to the correct tool, but is not the right choice. The model understands the query but confuses functionally similar tools.

3. Context Understanding Error: The model failed to understand the query's intent or context, selecting a tool that is functionally unrelated to what the query requires.

Query: `{query}`
Ground Truth Tool: `{ground_truth}`
Model Selected: `{model_output}`
Available Tools: `{tool_list}`

IMPORTANT: Respond with ONLY ONE of these exact phrases:
Schema Misalignment Error
Functional Confusion Error
Context Understanding Error

Figure 11: Error classification prompt for analyzing tool selection errors.

Training Data Generation Prompt

You are an expert at explaining tool selection reasoning for AI systems. Your task is to generate high-quality, flowing reasoning that explains WHY specific tools were chosen for a given query.

Your Task:

Given:

- A user's query
- A list of available tools with descriptions
- The tools that were selected (answer)

Generate: A clear, logical reasoning in natural paragraph form that explains why these specific tools are the best choices.

Quality Criteria:

Your reasoning MUST:

1. Analyze the query deeply: Identify key requirements, implicit needs, and user intent
2. Connect query to tools: Explicitly link query elements to tool capabilities
3. Justify each selection: Explain why each tool is necessary
4. Show tool synergy: Explain how the tools work together
5. Consider alternatives: Briefly mention why other tools were NOT chosen
6. Be specific: Use concrete details from the query and tool descriptions
7. Write in flowing prose: Use natural paragraphs, not bullet points or lists

Example:

...

Now Generate:

User's Query: `{{ query }}`
Selected Tools: `{{ selected_tools }}`

Write your reasoning as flowing paragraphs that naturally cover query analysis, tool justification for each selected tool, their combined strategy, and why alternatives were not chosen.

Figure 12: Training data generation prompt for supervised fine-tuning in Metatool.

Description Enhancement Prompt (EasyTool-style)

You are tasked with improving a tool description following the EasyTool methodology. The goal is to create clear, structured, and unified instructions that help language models better understand the tool's purpose and usage.

Original Description: `{{ description }}`

Instructions:
1. Create a CONCISE description that clearly states the tool's core functionality
2. Focus on WHAT the tool does and WHEN to use it
3. Remove redundant, verbose, or marketing-style language
4. Use action-oriented verbs at the start (e.g., "Retrieves", "Calculates", "Searches")
5. Keep the description to 1-2 sentences maximum

Critical Rules:
- Do NOT include ANY tool name, API name, or brand name
- Do NOT start with "This tool...", "The tool...", or any name reference
- Start DIRECTLY with an action verb describing the functionality

Example:
Original: "WeatherAPI is a comprehensive weather tool that can be used to get current weather data and forecasts."
Improved: "Retrieves current weather conditions and forecasts including temperature, humidity, and precipitation for specified locations."

Generate ONLY the improved description without additional explanation.

Figure 13: Prompt template for description enhancement following EasyTool (Yuan et al., 2025).