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 in Data science, Seoul National University
{jonggeun.lee, opusdeisong, johnhan00, haesung.pyun, yohan.jo}@snu.ac.kr

Abstract

Small language models (SLMs) offer significant computational advantages for tool-augmented AI systems, yet 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 but nonexistent tool names that reflect naming conventions internalized during pretraining but absent from the provided tool schema. Rather than forcing models to adapt to arbitrary 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 from contamination detection indicating pretraining familiarity—to automatically rename tool components. By generating multiple candidates and selecting those with highest output concentration across samples, PA-Tool identifies pretrain-aligned naming patterns. Experiments on MetaTool and RoTBench show improvements of up to 17% points, with schema misalignment errors reduced by 80%. PA-Tool enables small models to approach state-of-the-art performance while maintaining computational efficiency for adaptation to new tools without retraining. Our work demonstrates that schema-level interventions can unlock the tool-use potential of resource-efficient models by adapting schemas to models rather than models to schemas. [1]

1 Introduction

Tool-augmented language models have become essential components of modern AI systems, with tool-agent frameworks being widely adopted across various applications (Schick et al., 2023). As these systems mature, there is growing interest in deploying small language models (SLMs) due to computational constraints and the need for edge deployment (Belcak et al., 2025; Chen et al., 2025b). In these systems, SLMs must perform two critical operations: tool selection (identifying which function to call) and parameter identification (providing appropriate arguments). However, small models struggle significantly with these tasks, exhibiting severe performance degradation particularly for models below 10B parameters that are most viable for edge deployment (Patil et al., 2025; Erdogan et al., 2024).

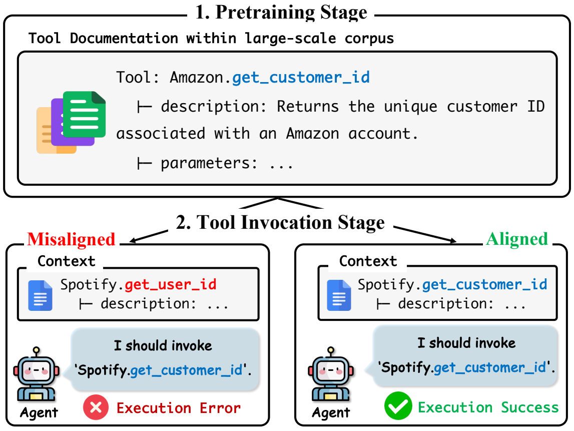
1. Pretraining Stage
Tool Documentation within large-scale corpus
Tool: Amazon.get_customer_id
⊢ description: Returns the unique customer ID associated with an Amazon account.
⊢ parameters: ...
2. Tool Invocation Stage
Misaligned
Context
Spotify.get_user_id
⊢ description: ...
I should invoke 'Spotify.get_customer_id'.
Agent
Execution Error
Aligned
Context
Spotify.get_customer_id
⊢ description: ...
I should invoke 'Spotify.get_customer_id'.
Agent
Execution Success

Figure 1: Effect of schema alignment on tool invocation. **Top**: Models learn tool documentation during pretraining. **Bottom-Left**: When schemas misalign with these internalized patterns, models generate plausible but nonexistent tools. **Bottom-Right**: Schema alignment with pretrained knowledge prevents such errors.

A common failure pattern is *schema misalignment*: models hallucinate plausible but nonexistent components within the tool schema—a hierarchical structure consisting of tools and parameters, each with associated descriptions. As illustrated in Figure 1 (Top), models encounter various tool documentation patterns during pretraining, internalizing common naming conventions and API structures. However, when deployed with a specific tool schema, misalignment can occur. For

[*] Equal contribution.
[†] Corresponding author.
[1] We will release our code upon publication of the paper.

instance (Figure 1, Bottom-Left), when the actual tool name to invoke is get_user_id, the model may instead generate get_customer_id—a similar but non-existent tool within the tool schema—leading to execution failure. Critically, these misaligned names are not random; they reflect naming patterns the model frequently encountered during pretraining but that are absent from the current tool schema. This raises a fundamental question: *rather than forcing models to adapt to arbitrary tool schemas, what if we adapt the schemas to align with the naming conventions the models have internalized deeply?* (Figure 1, Bottom-Right)

We hypothesize that by renaming tools to match patterns in the model's pretraining data—transforming get_user_id to get_customer_id if the latter better aligns with learned representations—we can reduce errors and improve tool-calling accuracy. To operationalize this idea, we propose PA-Tool (Pretraining-Aligned Tool Schema Generation), which generates a mapping between original tool names and pretrain-aligned alternatives. To identify such pretrain-aligned patterns, we leverage insights from contamination detection methods (Dong et al., 2024), which detect whether models encountered specific data during pretraining. A key finding from these studies is that contaminated data—patterns frequently seen during training—exhibits *peakedness*: models generate highly similar outputs across multiple sampling attempts, creating a concentrated distribution. This occurs because repeated exposure to specific patterns during pretraining strengthens internal representations, reducing output variance around familiar patterns (Shi et al., 2024a; Li, 2023). We adopt this *peakedness* as a signal for the pretraining familiarity of tool schemas.

PA-Tool renames each component in a schema in three stages: (1) instructing the target 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, i.e., the number of other candidates that are sufficiently similar based on 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 pretrained representation.

Experiments on MetaTool and RoTBench with SLMs demonstrate PA-Tool's effectiveness across diverse scenarios. On MetaTool, PA-Tool achieves substantial improvements across all subtasks, with the most significant gains when models must determine when no suitable tool exists (up to 17.0%) and when reasoning about multiple tools simultaneously (up to 9.6%). These improvements demonstrate PA-Tool's effectiveness in addressing schema misalignment, which becomes particularly critical in complex tool selection scenarios. On RoTBench, PA-Tool maintains robust performance in both single-turn (5-10%) and multi-turn settings (4-10%), demonstrating that alignment benefits persist across extended conversational contexts. Notably, PA-Tool enables SLMs to approach the performance of closed-source models, with configurations like Qwen2.5-7B-Instruct matching or exceeding Claude Sonnet 4.5 on tool selection tasks.

Our comprehensive analysis reveals that PA-Tool's effectiveness stems from directly addressing the predominant failure mode in SLMs. Schema misalignment errors—where models generate plausible but non-existent tool names—decrease by 80.0% with PA-Tool, while other error types show more modest reductions (18.8-24.0%). This targeted impact demonstrates that aligning schemas with pretrained knowledge resolves a bottleneck that disproportionately affects small models. Furthermore, PA-Tool's training-free nature makes it highly practical for real-world deployment. This method requires only a simple one-time schema mapping without any model modification, enabling a model to use diverse toolsets without costly and complex retraining, and without having to worry about catastrophic forgetting or overfitting. This schema-level intervention represents an alternative approach: rather than forcing models to conform to arbitrary documentation standards, we adapt the interface to match the models' internalized knowledge, enabling efficient tool use while preserving the computational advantages of small models.

Our contributions are summarized as follows:

- We propose PA-Tool, a training-free schema optimization method that repurposes peakedness—a contamination detection signal—to identify pretrain-aligned tool names.

- We demonstrate improvements of up to 17% points 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, small models can approach closed-source model performance in specific aspects while maintaining computational efficiency.

2 Related Works

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; Madaan et al., 2023; Wu et al., 2025; 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 (Belcak et al., 2025; Cheng et al., 2024). 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 don't exist in the actual schema—making them difficult to detect and correct through traditional validation mechanisms. Our work addresses this vulnerability by generating pretraining-aligned schemas that improve SLM reliability in these critical operations.

2.2 Improving Tool Utilization in LLMs

As LLMs have demonstrated 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 (Seo et al., 2025; Yao et al., 2025; Trivedi et al., 2024; Qin et al., 2024; Shim et al., 2025).

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 (Qian et al., 2025; Zhou et al., 2025; Shi et al., 2024b; Feng et al., 2025; Chen et al., 2025a), but they require substantial data or computational resources. Training-free methods refine tool documentation (Yuan et al., 2025; Qu et al., 2025) or leverage interaction histories (Wang et al., 2024; Cui et al., 2025; Zhao et al., 2024; Fu et al., 2024). However, existing training-free approaches primarily focus on improving descriptions rather than aligning the schema itself with model-preferred representations. Our work directly addresses this gap by generating schemas aligned with pretraining distributions, targeting the root cause of schema misalignments.

2.3 Data Contamination Detection

Data contamination, defined as overlap between pretraining data and evaluation benchmarks, can inflate performance through memorization. Early detection methods used n-gram overlap (Brown et al., 2020), but this approach fails 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 pretrain-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 pretrain-aligned tool schemas. A tool schema is hierarchical, structured documentation that defines tools and parameters along with detailed descriptions for each component. Our objective is to transform these descriptions into component names that the model is familiar with and that align with patterns the model has frequently encountered during pretraining. Specifically, we construct a dictionary mapping that associates each original component with a pretrain-aligned name—one that the language model would naturally generate based on its pretraining experience.

The core principle is straightforward: by analyzing the peakedness of a model's output distribution when generating names from component

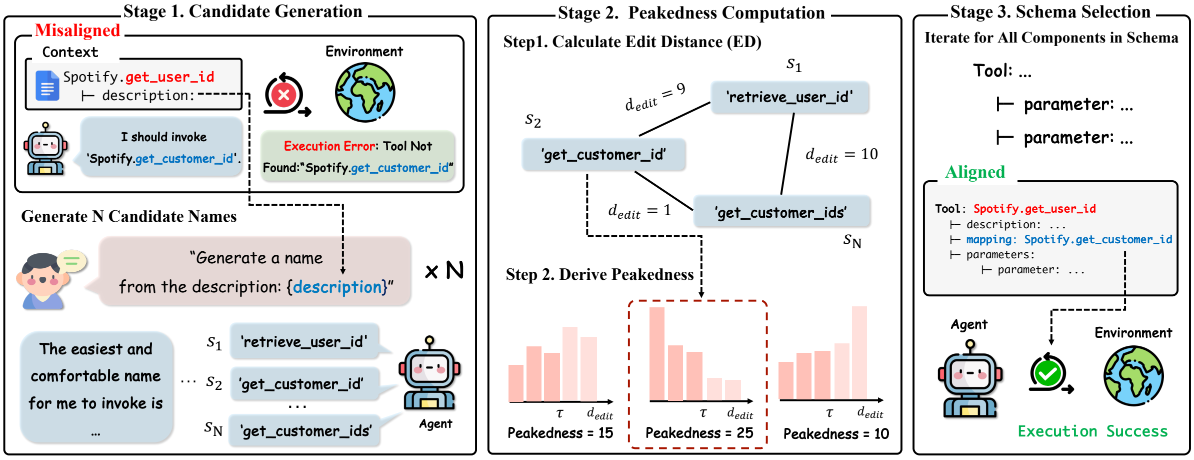
Stage 1. Candidate Generation
Misaligned
Context
Spotify.get_user_id
⊢ description:
Environment
I should invoke 'Spotify.get_customer_id'.
Execution Error: Tool Not Found:"Spotify.get_customer_id"
Generate N Candidate Names
"Generate a name from the description: {description}"
× N
The easiest and comfortable name for me to invoke is ...
s_1 'retrieve_user_id'
s_2 'get_customer_id'
s_N 'get_customer_ids'
Agent
Stage 2. Peakedness Computation
Step1. Calculate Edit Distance (ED)
s_1 'retrieve_user_id'
s_2 'get_customer_id'
s_N 'get_customer_ids'
$d_{edit} = 9$
$d_{edit} = 10$
$d_{edit} = 1$
Step 2. Derive Peakedness
τ d_{edit}
Peakedness = 15
Peakedness = 25
Peakedness = 10
Stage 3. Schema Selection
Iterate for All Components in Schema
Tool: ...
⊢ parameter: ...
⊢ parameter: ...
Aligned
Tool: Spotify.get_user_id
⊢ description: ...
⊢ mapping: Spotify.get_customer_id
⊢ parameters:
⊢ parameter: ...
Agent
Environment
Execution Success

Figure 2: Overview of our PA-Tool framework.

descriptions, we can identify which naming patterns the model has most frequently encountered during pretraining. These pretrain-aligned names are hypothesized to improve the model's ability to accurately invoke tools with correct parameters, reducing schema-related errors such as incorrect tool selections and parameter mismatches.

3.1 Framework Overview

Let $\mathcal{M}$ denote a pre-trained 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 pretrain-aligned name (Stage 3). By applying this process to each component in the schema hierarchy, we construct a dictionary mapping from original names to pretrain-aligned names. The complete algorithm is provided in Appendix B.

3.2 Stage 1: Candidate Generation

In this stage, we collect diverse candidate names of a component that the model may have encountered during the 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 the Appendix 5. Additionally, we generate a reference name s_{ref} using greedy decoding ($t = 0$) for tie-breaking purposes in the selection stage.

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 around it. The intuition is that candidate names with many similar variants indicate pretrain-frequent patterns 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:

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

where $\ell_{\max} = \max_{s_i \in \mathcal{C}} |s_i|$ 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:

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

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: Representative Name Selection

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

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

This criterion identifies the naming pattern that the model generates most consistently across multiple sampling attempts, indicating it represents the most deeply internalized pattern from the training data. In case of ties where multiple candidates achieve the same maximum peakedness score, we break ties by selecting the candidate with the minimum edit distance to the reference name:

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

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. When a naming pattern appears multiple times during training, the model develops stronger internal representations, leading to higher probability mass and reduced variance around these patterns. 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 schema components (Figure 2-3), we obtain the final pretrain-aligned schema.

4 Experimental Setup

To evaluate whether PA-Tool improves models' core tool-use capabilities, we conduct comprehensive experiments assessing two fundamental aspects: (1) **Tool selection**, whether models correctly identify the appropriate tool for a given query from available options, and (2) **Parameter identification**, whether models accurately extract necessary arguments to construct valid tool calls.

4.1 Benchmarks

MetaTool. MetaTool (Huang et al., 2024) evaluates tool selection capabilities across 4,287 test cases in four subtasks: (1) **Similar** tests semantic comprehension by distinguishing tools with overlapping functionalities (e.g., Sudoku vs. Tic-Tac-Toe); (2) **Scenario** select 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 whether models can robustly use tools when descriptions contain various noise perturbations (e.g. reversed names). However, we focus exclusively on the Clean environment to isolate the effects of PA-Tool on core tool-use capabilities rather than noise robustness. We evaluate on 105 test cases across 568 tools in both single-turn and multi-turn settings. RoTBench measures two fundamental tool use capabilities: (1) **Tool Selection** evaluates whether models correctly identify the appropriate tool; and (2) **Parameter Identification** assesses whether models accurately extract the required parameter set, conditioned on correct tool selection.

4.2 Models and Baselines

We focus on models with 8B parameters or fewer (SLMs), evaluating four open-source models: Qwen 2.5 3B/7B (Qwen et al., 2025), Llama-3.1 8B (Meta AI, 2024a), and Llama 3.2 3B (Meta AI, 2024b). For each model, we evaluate three configurations: **Base** without any modifications, **Greedy** using greedy decoding for schema generation, and **PA-Tool** with our proposed 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, 2024), Gemini 2.5 Flash (Basu Mallick et al., 2025), and Claude Sonnet 4.5 (Anthropic, 2025).

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	**28.6**	12.9
Gemini-2.5-Flash	Base	70.0	79.8	**89.2**	77.3	82.9	56.2	22.9	12.9
Claude-Sonnet-4.5	Base	75.6	83.0	84.3	**85.1**	**83.3**	**67.6**	25.7	**17.1**
Open-Source Models									
Qwen2.5-3B	Base	48.7	55.3	83.6	**75.1**	1.9	0.0	1.4	0.0
	Greedy	49.1	57.0	82.9	63.4	**6.7**	**3.8**	5.7	1.4
	PA-Tool	**50.0**	**58.8**	**86.2**	72.6	**6.7**	1.9	**7.1**	**4.3**
Qwen2.5-7B	Base	59.6	74.4	78.3	78.3	49.5	20.0	8.6	5.7
	Greedy	60.6	75.4	85.4	82.7	50.5	21.0	14.3	8.6
	PA-Tool	**64.1**	**78.4**	**88.2**	**84.9**	**55.2**	**21.9**	**18.6**	**11.4**
Llama3.2-3B	Base	55.0	58.6	43.6	79.1	56.2	20.0	27.1	**5.7**
	Greedy	57.7	58.9	39.8	70.4	59.1	20.0	28.6	4.3
	PA-Tool	**65.7**	**67.7**	**60.6**	**80.5**	**62.9**	**21.9**	**31.4**	**5.7**
Llama3.1-8B	Base	61.5	73.9	53.5	78.7	58.1	17.1	17.1	**4.3**
	Greedy	64.6	72.9	51.5	78.9	63.8	**18.1**	17.1	**4.3**
	PA-Tool	**70.4**	**79.9**	**66.0**	**88.3**	**68.6**	**18.1**	**22.9**	**4.3**

Table 1: Performance comparison on MetaTool and RoTBench. All metrics are reported as accuracy (%). Tool Sel. and Param Iden. denote Tool Selection and Parameter Identification, respectively. **Bold** indicates the best performance within each closed-source model and among all open-source models.

4.3 Implementation Details

When generating tool schemas with PA-Tool, we generate 32 candidates with temperature 0.4 and set α to 0.2. For all benchmark evaluations, we employ a one-shot prompting strategy where the prompt includes: (1) the task instruction, (2) the target tool description (not the complete schema), and (3) one demonstration example. We use greedy decoding 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 prompt templates are provided in Appendix C.

5 Main Results

We present our main experimental results demonstrating that PA-Tool consistently achieves the best performance across all models and tasks on both MetaTool and RoTBench benchmarks.

5.1 MetaTool: Consistent Improvements Across All Subtasks

Table 1 shows that PA-Tool consistently improves tool selection performance across all four MetaTool subtasks. In the Similar and Scenario tasks, PA-Tool achieves improvements of up to 10.7 % points, demonstrating that even simple alignment of tool names with the model's internal knowledge effectively enhances tool selection. This phenomenon is more pronounced in the Multi-tool task, where gains reach up to 9.6 % points, with Llama3.1-8B improving from 78.7% to 88.3%. When tasks require identifying multiple tools simultaneously, the impact of schema misalignment compounds across each tool selection, making alignment particularly critical.

The Reliability task shows substantial improvements of up to 17.0%. For example, Llama3.2-3B improves from 43.6% to 60.6%. This task requires models to recognize when no suitable tool exists in the candidate list. Success depends on accurately understanding what each available tool does. PA-Tool aligns tool names with pretrained knowledge, enabling models to more clearly distinguish between available options and confidently determine when none match the query requirements.

Notably, the Greedy Schema baseline occasionally underperforms the Base model, with Llama3.2-3B showing 39.8% versus 43.6% in Reliability. This can be attributed to greedy decoding generating only a single schema candidate, which limits exploration of the schema space and increases susceptibility to suboptimal or misaligned schemas.

5.2 RoTBench: Persistent Benefits in Multi-Turn Interactions

RoTBench evaluates models in both single-turn and multi-turn conversational settings across two sequential stages: tool selection and parameter identification. In tool selection, PA-Tool demonstrates consistent improvements of 5-10% points across all models in single-turn scenarios. Importantly, these benefits remain robust in multi-turn settings with gains of 4-10% points in multi-turn settings, demonstrating that schema alignment remains effective across extended conversational contexts.

Parameter identification also shows improvements, though more modest than tool selection. The most notable gain appears in multi-turn settings, where Qwen2.5-7B nearly doubles from 5.7% to 11.4%. The smaller overall improvement stems from the nature of parameter naming. Parameters typically have explicit, descriptive names that already align well with common conventions (e.g., `user_id` with description "a unique user identifier"). Unlike tool names, which vary significantly across different tools, parameter names follow more standardized patterns. This leaves less room for alignment-based improvements.

5.3 Comparison with Closed-source Models

While our open-source models with PA-Tool show substantial improvements, a performance gap remains compared to state-of-the-art closed-source models. Claude Sonnet 4.5 achieves 83.3% tool selection and 67.6% parameter identification on RoTBench single-turn, significantly outperforming even our best configurations. However, PA-Tool narrows this gap, elevating SLMs closer to closed-source baselines while maintaining the computational efficiency advantages of SLMs.

Notably, in certain aspects of tool selection, PA-Tool enables small models to achieve competitive or superior performance. In MetaTool's Multi-tool task, Llama3.1-8B with PA-Tool achieves 88.3%, surpassing all closed-source models including Claude Sonnet 4.5. Similarly, in the Reliability task, Qwen2.5-7B with PA-Tool reaches 88.2%, closely approaching Gemini 2.5 Flash's 89.2%. These results demonstrate that schema alignment can enable resource-efficient models to match or exceed state-of-the-art systems in specific aspects of tool selection, particularly in tasks where schema misalignment is a primary failure mode.

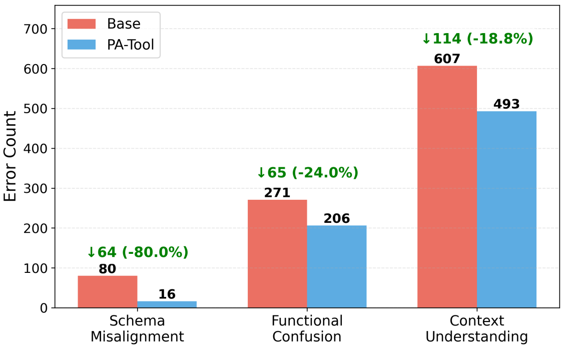

Figure 3: Error type distribution for Llama-3.1-8B on MetaTool tool selection tasks.

6 Analysis

6.1 Error Analysis

To understand how PA-Tool addresses different failure modes, we analyze error type distributions in Llama3.1-8B before and after applying PA-Tool. We categorize all incorrect predictions from both Base and PA-Tool configurations into three error types: *Schema Misalignment Error*, where the model generates non-existent or incorrectly named tools; *Functional Confusion Error*, where the model selects wrong tools with similar functionality; and *Context Understanding Error*, where the model selects functionally unrelated tools. Detailed error type definitions and experimental settings are provided in Appendix E.

Results. Figure 3 shows how PA-Tool affects different error types. Schema Misalignment Errors decrease dramatically by 80.0% (from 80 to 16 cases), confirming that aligning tool names with pretrained knowledge directly addresses this primary failure mode. In contrast, Functional Confusion Errors decrease by 24.0% (from 271 to 206 cases) and Context Understanding Errors decrease by 18.8% (from 607 to 493 cases). This stark difference—80.0% reduction for schema-related errors versus 18.8-24.0% for other types—demonstrates PA-Tool's targeted impact. Schema alignment effectively eliminates naming-related failures, but distinguishing functionally similar tools and understanding complex query contexts require capabilities beyond schema optimization.

6.2 Integration with Supervised Fine-tuning

While PA-Tool is training-free, we demonstrate it remains effective when combined with supervised fine-tuning (SFT), addressing a key question: do models retain pretraining-aligned naming prefer-

Configuration	Similar	Scenario	Reliability	Multi-tool
Llama3.1-8B	65.2	71.1	55.1	79.8
+ SFT	71.2	77.6	57.1	82.8
+ PA-Tool	72.2	75.9	**67.2**	**89.9**
+ SFT + PA-Tool	**72.7**	**80.8**	57.1	**89.9**

Table 2: Results of fine-tuning method and PA-Tool on MetaTool 10% random sampled test set.

ences after task-specific adaptation? Detailed information about SFT is presented in Appendix D. We compare four configurations: (1) Vanilla: base model with original schemas; (2) SFT only: fine-tuned model with original schemas; (3) PA-Tool only: base model with aligned schemas; (4) SFT + PA-Tool: SFT model with schemas aligned using the SFT model's own PA-Tool generation.

Results. Table 2 shows PA-Tool provides consistent gains whether applied alone or with SFT. Notably, PA-Tool alone (third row) outperforms SFT alone (second row) across Similar, Reliability and Multi-tool by up to 10.1%, demonstrating superior efficiency without training. When combined, SFT + PA-Tool (fourth row) achieves the best performance on Similar (72.7%), Scenario (80.8%) and Multi-tool (89.9%) with consistent improvements over SFT alone across all subtasks.

Implications. The consistent gains demonstrate that pretraining-aligned naming preferences persist through fine-tuning. Schemas generated by PA-Tool using either the base or fine-tuned model are highly similar and both improve the model's performance, indicating that task-specific adaptation does not override deeply internalized naming conventions from pretraining. This shows PA-Tool can deliver further performance gains on top of fine-tuning: schema-level alignment remains necessary to bridge the gap between arbitrary documentation standards and the model's pretrained knowledge.

6.3 Hyperparameter Analysis

We investigate the impact 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, and we report the average accuracy across all four subtasks. Detailed results are provided in Appendix F.

Number of candidates (N) (Figure 4, Top). Smaller models (3B) achieve stable performance with 16-32 candidates, while larger models (7-8B) require 32-64 candidates before performance plateaus. Beyond these ranges, additional candidates provide minimal gains.

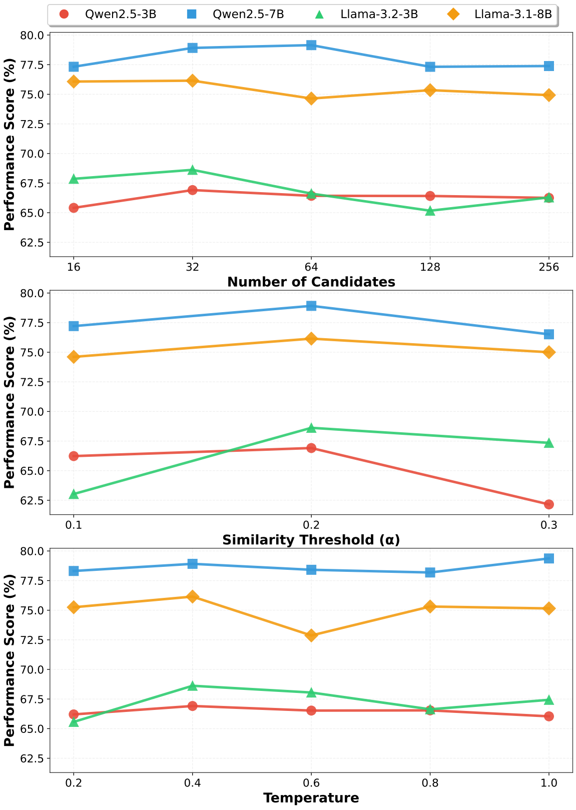

Figure 4: Impact of hyperparameters on PA-Tool across different models. All results are averaged across four MetaTool subtasks. **Top:** Effect of the number of candidates (N). **Middle:** Effect of similarity threshold (α). **Bottom:** Effect of sampling temperature (t).

Similarity threshold (α) (Figure 4, Middle). Performance peaks at $\alpha = 0.2$ consistently across all models. At $\alpha = 0.1$, performance drops by 2-3 % points, while $\alpha = 0.3$ shows similar degradation.

Sampling temperature (t) (Figure 4, Bottom). Performance remains stable across temperatures $t \in [0.2, 1.0]$, varying within 1-2 % points. Moderate temperatures ($t = 0.4$-0.6) show slightly better results, though the differences are marginal.

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% points 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

While PA-Tool effectively reduces schema misalignment errors by 80%, it shows more modest improvements for functional confusion and context understanding errors, indicating that schema optimization alone cannot address all failure modes. Our reliance on peakedness as an alignment signal assumes this metric reliably indicates pretraining 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 similarity metrics may differ for non-Latin scripts or morphologically complex languages. PA-Tool incurs one-time computational overhead from candidate generation and evaluation. While negligible compared to fine-tuning, this cost could become significant for tool libraries with thousands of components.

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 2025-10-04.

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 2025-10-04.

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 B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher 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. *Preprint*, arXiv:2005.14165.

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 Ponde 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, Josh 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. *Preprint*, arXiv: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. *Preprint*, arXiv:2402.15938.

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.

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

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, et al. 2024. Metatool benchmark for large language models: Deciding whether to use tools and which to use. In *ICLR*.

V. I. Levenshtein. 1965. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848. Original Russian publication.

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*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Meta AI. 2024a. Introducing llama 3.1: Our most capable models to date. `https://ai.meta.com/blog/meta-llama-3-1/`. Accessed 2025-10-04.

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 2025-10-04.

OpenAI. 2024. Gpt-4.1 mini. `https://openai.com/index/gpt-4-1/`. Accessed 2025-10-04.

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. *Preprint*, arXiv:2504.13958.

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. 2025. From exploration to mastery: Enabling LLMs to master tools via self-driven interactions. In

The Thirteenth International Conference on Learning Representations.

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.

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. 2025. Simuhome: A temporal- and environment-aware benchmark for smart home llm agents. *Preprint*, arXiv:2509.24282.

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. *Preprint*, arXiv:2310.16789.

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. 2025. Non-collaborative user simulators for tool agents. *Preprint*, arXiv:2509.23124.

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.

Qinzhuo Wu, Pengzhi Gao, Wei Liu, and Jian Luan. 2025. Backtrackagent: Enhancing gui agent with error detection and backtracking mechanism. *Preprint*, arXiv:2505.20660.

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.

Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. 2025. Sweet-rl: Training multi-turn llm agents on collaborative reasoning tasks. *Preprint*, arXiv:2503.15478.

A Example of Schema Generation Process

We provide a concrete example of how PA-Tool generates pretrain-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.

B Detailed Algorithm for PA-Tool

Algorithm 1 Pseudocode for PA-Tool

```
Require: Language model $\mathcal{M}$, Component description $d$, samples $s$, temperature $t$, hyperparameter $\alpha$
Ensure: Representative schema $s^*$
// Stage 1: Candidate Generation
$s_{\text{ref}} \leftarrow$ Generate schema from $\mathcal{M}$ with temperature 0
$\mathcal{C} \leftarrow \{\}$
for $i = 1$ to $s$ do
    $s_i \leftarrow$ Generate schema 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 schema 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 schemas in $\mathcal{C}$ with edit distance $\leq \tau$ from $s_i$   ▷ Eq. (2)
end for
// Stage 3: Schema Selection
$\mathcal{C}^* \leftarrow$ schemas in $\mathcal{C}$ with maximum peakedness   ▷ Eq. (3)
if $|\mathcal{C}^*| == 1$ then
    $s^* \leftarrow$ the unique schema in $\mathcal{C}^*$
else
    $s^* \leftarrow$ schema in $\mathcal{C}^*$ closest to $s_{\text{ref}}$     ▷ Eq. (4)
end if
return $s^*$
```

C 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 5: Component name generation prompt for Stage(1) in Figure 2.

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 6: Error classification prompt for analyzing tool selection errors.

D Experimental Setup of Supervised Fine-tuning

Setup. We fine-tune `Llama-3.1-8B` on MetaTool using LoRA (rank=32, $\alpha = 64$, dropout=0.05) with a 60/20/20 train/val/test split. We compare four configurations: (1) Vanilla (base model, original schemas); (2) PA-Tool only; (3) SFT only; (4) SFT + PA-Tool.

E Error Analysis Details

Error taxonomy. We categorize tool selection errors into three types based on their underlying failure modes: (1) *Schema Misalignment Error* occurs when the model generates a plausible but non-existent tool name, or selects a tool with similar naming patterns but different from the correct one. This indicates the model follows its pretrained conventions rather than the provided schema. For example, when the correct tool is `get_user_id`, the model might generate `get_customer_id`—a naming pattern it encountered frequently during pretraining but absent from the current schema. (2) *Functional Confusion Error* occurs when the model selects an existing tool with similar functionality to the correct tool but is not the right choice. This suggests the model understands the query but confuses functionally similar tools. For instance, when asked to send an email notification, the model might select `send_sms` instead of the correct `send_email`, both of which are communication tools but use different channels. (3) *Context Understanding Error* occurs when the model fails to comprehend the query's intent, selecting a functionally unrelated tool. For example, when asked to delete a user account, the model might incorrectly select `create_user` or `list_products`, indicating a fundamental misunderstanding of the task.

Experimental setup. We implement an automated error analyzer using GPT-4.1-mini (OpenAI, 2024) to classify errors from all open-sourced models in both Base and PA-Tool configurations. The analysis is conducted on MetaTool's three tool selection subtasks: Similar, Scenario, and Multi-tool. We exclude the Reliability subtask as it specifically tests the "no suitable tool" scenario rather than tool selection errors. The classification prompt is provided in Figure 6.

F Hyperparameter Detailed Results

Table 3: Effect of Number of Candidates on Model Performance

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	**70.3**	**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 4: Effect of Similarity Threshold on Model Performance

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	**70.3**	**79.9**	**66.0**	88.3
	0.3	68.1	78.7	64.4	**88.7**

Table 5: Effect of Sampling Temperature on Model Performance

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	**70.3**	**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