TOOLSPEC: Accelerating Tool Calling via Schema-Aware and Retrieval-Augmented Speculative Decoding

Heming Xia[1], **Yongqi Li**[1*], **Cunxiao Du**[1], **Mingbo Song**[2], **Wenjie Li**[1]
[1] Department of Computing, The Hong Kong Polytechnic University [2] Peking University
{he-ming.xia}@connect.polyu.hk; {harukioj}@outlook.com

Abstract

Tool calling has greatly expanded the practical utility of large language models (LLMs) by enabling them to interact with external applications. As LLM capabilities advance, effective tool use increasingly involves multi-step, multi-turn interactions to solve complex tasks. However, the resulting growth in tool interactions incurs substantial latency, posing a key challenge for real-time LLM serving. Through empirical analysis, we find that tool-calling traces are highly structured, conform to constrained schemas, and often exhibit recurring invocation patterns. Motivated by these observations, we propose TOOLSPEC, a schema-aware, retrieval-augmented speculative decoding method for accelerating tool calling. TOOLSPEC exploits predefined tool schemas to generate accurate drafts, using a finite-state machine to alternate between deterministic schema token filling and speculative generation for variable fields. In addition, TOOLSPEC retrieves similar historical tool invocations and reuses them as drafts to further improve efficiency. TOOLSPEC presents a training-free, plug-and-play solution that can be seamlessly integrated into existing LLM workflows. Experimental results across multiple tool-calling benchmarks demonstrate that TOOLSPEC achieves up to a $4.2\times$ speedup, substantially outperforming existing training-free speculative decoding methods. We release our code at https://github.com/hemingkx/ToolSpec.

1 Introduction

Recent advances in large language models (LLMs) have substantially broadened their capabilities, including open-domain dialogue, mathematical reasoning, and program synthesis (OpenAI, 2024; Anthropic, 2024; Yang et al., 2025). In parallel, tool calling has emerged as a pivotal capability that enables LLMs to interact with external tools such as search engines (Jin et al., 2025; Zheng et al., 2025), calculators (Schick et al., 2023; Qin et al., 2024b), and code interpreters (Gao et al., 2023; Gou et al., 2024). By incorporating external tools, LLMs can alleviate fundamental limitations of standalone models, including outdated knowledge and arithmetic errors (Qin et al., 2024a). As LLM capabilities continue to improve, tool use increasingly adopts multi-step, multi-turn, feedback-driven interaction loops to solve complex tasks (Patil et al., 2024; Yao et al., 2025).

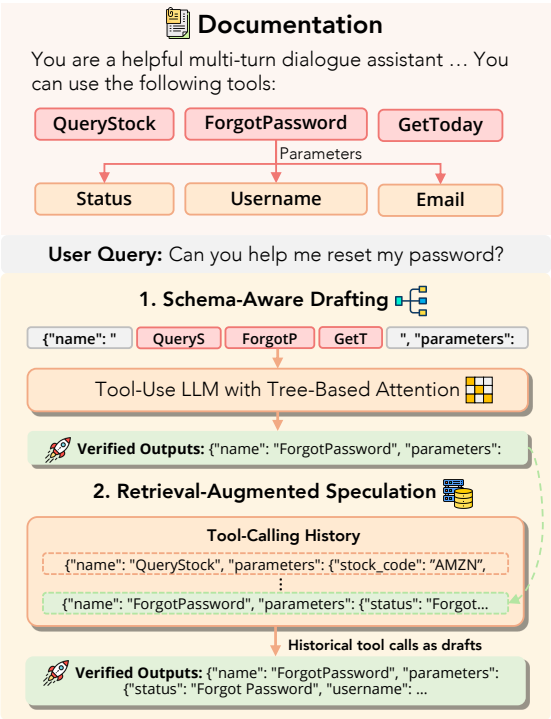

Figure 1: Illustration of TOOLSPEC, which accelerates tool-calling generation via two innovations: **1) schema-aware drafting**, where predefined tool schemas serve as faithful drafts and enable parallel verification of constrained variables (e.g., tool names); and **2) retrieval-augmented speculation**, retrieving and reusing similar historical tool invocations as high-quality drafts.

However, the growing frequency of tool invo-

[*] Corresponding author.

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 9401–9414
October 24-29, 2026 ©2026 Association for Computational Linguistics

cations substantially increases inference latency, posing a major challenge for real-time LLM serving. Recent work has proposed several strategies to accelerate tool calling (Kim et al., 2024; Zhu et al., 2025; Xu et al., 2024; Nichols et al., 2025). One line of research models tool-calling trajectories as directed acyclic graphs (DAGs) that capture inter-tool dependencies and then execute independent tools in parallel to reduce overall latency (Kim et al., 2024; Zhu et al., 2025). Another line improves parallelism by overlapping tool execution with text generation, that is, launching tool calls while the model is still decoding, thereby further accelerating end-to-end tool-calling (Xu et al., 2024; Nichols et al., 2025).

Despite these advances in accelerating *tool execution*, this work shifts focus to improving the efficiency of *tool-calling generation*. Our preliminary analysis indicates that generation latency can be a dominant bottleneck: on the widely used benchmark ToolBench (Qin et al., 2024b), generating tool calls with Qwen2.5-14B-Instruct accounts for approximately 80% of the end-to-end inference latency and is roughly $4\times$ larger than the time spent executing the tools. Moreover, whereas tool execution time is often relatively stable for a fixed toolset and environment, tool-calling generation latency grows with model size and with the length of the generated tool-call sequence, making it the primary contributor to latency in many tool-using scenarios.

In this work, we present two key observations regarding tool calling: **1) tool-calling outputs are highly structured.** These outputs are typically expressed in JSON and adhere to strictly constrained schemas. As a result, among the generated tokens, only a small subset corresponds to variable fields, while the rest are schema-determined; **2) LLMs exhibit recurring tool-call patterns across different requests,** indicating the presence of reusable structural and semantic regularities. Motivated by these insights, we propose TOOLSPEC, a schema-aware and retrieval-augmented speculative decoding (SD) approach for accelerating tool calling. TOOLSPEC leverages the task-specific tool schemas to construct high-quality draft sequences, and employs a finite-state machine to alternate between deterministic schema token filling and speculative generation for variable fields. Furthermore, TOOLSPEC retrieves similar historical tool calls and reuses them as draft candidates, thereby further improving efficiency. Experiments on widely used tool-calling benchmarks, including API-Bank, ToolAlpaca, BFCLv2, and ToolBench, demonstrate that TOOLSPEC achieves up to a $4.2\times$ speedup and outperforms existing SD methods.

We summarize our contributions as follows:

- We propose TOOLSPEC, a schema-aware and retrieval-augmented SD approach for accelerating tool-calling generation. It presents a *training-free* solution that can be seamlessly integrated into existing LLM workflows.
- Extensive experiments across diverse models and tasks demonstrate that TOOLSPEC consistently achieves a $3.5\times\sim4.2\times$ speedup, with up to a 71% relative improvement over the previous state-of-the-art SD method.
- Further analysis validates the effectiveness of our proposed components and highlights the impact of format adherence on TOOLSPEC 's speedup, indicating its efficiency potential as LLM capabilities continue to advance.

2 Related Work

Efficient Tool Calling As large language models (LLMs) increasingly rely on multi-step, multi-turn tool-calling workflows, inference latency in LLM serving has grown substantially. To address this, recent studies leverage directed acyclic graphs (DAGs) to model inter-tool dependencies, enabling parallel execution of independent API calls and thereby reducing latency (Kim et al., 2024; Zhu et al., 2025). Another line of work executes tool calls proactively during generation to minimize system idle time (Xu et al., 2024; Nichols et al., 2025). In contrast to prior methods that focus on tool execution, this work shifts attention to accelerating tool-calling generation, further improving the inference efficiency of tool-use LLMs.

Speculative Decoding In recent years, speculative decoding (SD) has emerged as a widely adopted paradigm for accelerating LLM inference (Leviathan et al., 2023; Xia et al., 2023). At each decoding step, SD first generates a sequence of draft tokens, which are subsequently verified in parallel by the target LLM (Xia et al., 2024). Recent advances in SD, such as the Eagle series (Li et al., 2024b,a, 2025), introduce lightweight draft models layered on top of the target LLM, achieving promising speedups. While these approaches require additional training or parameters, an alternative line of research focuses on training-free SD methods (Luo et al., 2025; Saxena, 2023; Xia

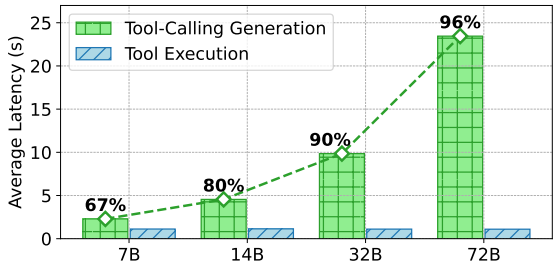

Figure 2: Latency breakdown of the Qwen2.5-Instruct model series on ToolBench (Qin et al., 2024b). Tool execution latency remains essentially constant, whereas tool-calling generation latency increases with model scale, reaching up to **96**% of the end-to-end latency.

et al., 2025). Notably, Token Recycling (Luo et al., 2025) stores the top-k candidate tokens of previous steps in an adjacency matrix and uses breadth-first search to construct a draft tree. Similarly, SAM-Decoding (Hu et al., 2025) leverages a suffix automaton to efficiently retrieve high-quality drafts from both the input context and a static text corpus.

3 Preliminary Study

In this section, we conduct an empirical analysis of tool-calling generation, covering (i) the latency breakdown between generation and tool execution, and (ii) analysis of tool-call patterns.

3.1 Latency Breakdown

We begin by analyzing the latency bottlenecks in tool calling. Experiments are conducted using the Qwen2.5-Instruct series on ToolBench (Qin et al., 2024b). As shown in Figure 2, tool execution latency remains nearly constant across model scales, whereas tool-calling generation latency increases significantly, accounting for up to **96**% of the end-to-end latency for Qwen2.5-72B-Instruct. These results indicate that tool-calling generation is the dominant bottleneck in tool-calling pipelines, particularly in multi-tool, multi-turn scenarios, underscoring the importance of accelerating generation.

3.2 Analysis of Tool-Call Patterns

Figure 3 shows an example tool-calling trace from API-Bank (Li et al., 2023). As illustrated, these traces are highly structured JSON objects that conform to a constrained schema, consisting of:

- **schema tokens**, e.g., "`name`", "`parameters`", and punctuation, which form the fixed scaffolding required by the tool-call schema;

A Case of Tool Calls

```
{
    "name": "RecordHealthData",
    "parameters": {
        "user_id": "12345",
        "time": "2023-10-19 10:30:00",
        "health_data": [
            {"name": "blood_pressure", "value": "120/80"},
            {"name": "heart_rate", "value": "80"}
        ]
    }
}
```

Figure 3: A tool-calling trace from API-Bank (Li et al., 2023), with different fields highlighted in color.

Models	API-Bank	ToolAlpaca
Llama-3.1-8B-Instruct	100.0%	100.0%
Llama-3.2-3B-Instruct	99.5%	93.4%
Qwen2.5-14B-Instruct	99.7%	100.0%
Qwen2.5-7B-Instruct	100.0%	100.0%

Table 1: Format adherence of different models.

- **tool and parameter names**, highlighted in red and blue, respectively, which specify the target tool and its arguments;
- **parameter values**, in green, which instantiate the arguments with query-specific content.

These structured patterns distinguish tool-calling generation from general natural-language generation, suggesting that it can be formulated as a *constrained decoding* problem. In practice, valid tool calls must adhere to strict JSON schemas, implying that a large portion of the output tokens, i.e., schema tokens, is effectively predetermined. Once the target tool is selected, its parameter names are drawn from a finite, schema-defined set (e.g., "`user_id`", "`time`", and "`health_data`"). Consequently, the model only needs to generate a small subset of tokens, i.e., the parameter values, conditioned on the user query and the dialogue context.

In the following, we further analyze practical tool-calling behaviors in LLMs, including schema adherence and recurring tool-invocation patterns.

Schema Adherence As shown in Table 1, current tool-use LLMs exhibit strong adherence to tool-calling formats, i.e., their outputs can be reliably parsed as valid JSON. Both the LLaMA-Instruct and Qwen2.5-Instruct model families achieve over 99% accuracy in following the required tool-call schema. This observation further supports our earlier claim that tool-calling generation can be formulated as constrained decoding under predefined schemas. In light of this, acceleration methods

Datasets	#Tools	#Instances	Avg. Tool Rep.
API-Bank	2138	5221	10.95
ToolAlpaca	426	3938	4.82
BFCLv2	739	2251	9.01

Table 2: Statistics of tool-calling benchmarks.

designed for general text generation may be suboptimal for tool calling. In contrast, explicitly exploiting the constrained structure of tool-calling generation can provide additional efficiency gains.

Repetitive Tool Calls Beyond schema adherence, we observe that LLMs frequently invoke the same tools to satisfy recurring user intents and preference patterns. This behavior is consistently observed across widely used tool-calling benchmarks. As shown in Table 2, repeated invocations of identical tools occur frequently in API-Bank (Li et al., 2023), with the average number of calls per tool exceeding 10.95. This observation motivates exploiting historical tool-calling traces. Rather than discarding prior calls, a system can reuse previously issued tool calls when they remain valid, thereby further improving tool-calling efficiency.

4 TOOLSPEC

In this section, we introduce TOOLSPEC, a plug-and-play SD approach for accelerating tool-calling generation. The method leverages the highly structured nature of tool calls and recurring historical patterns to construct high-quality drafts with negligible overhead. As shown in Figure 5, TOOLSPEC integrates two complementary drafting strategies: *schema-aware drafting* and *retrieval-augmented speculation*, which we detail below.

4.1 Task Formulation

Given a user query and an input context containing system prompts and tool documentation, TOOLSPEC extracts tool specifications from the documentation and converts them into a structured schema. The schema defines (i) the set of schema tokens Σ_{schema}, (ii) a set of callable tools $\mathcal{C}_{\text{tools}}$ (e.g., `ForgotPassword`), and (iii) the parameter fields associated with each tool (e.g., `Status`, `Username`).

Let $\mathcal{M}$ denote the target model, $\boldsymbol{x}$ the input context, and $\boldsymbol{y} = (y_1, y_2, \ldots, y_L)$ the tool-call sequence to be generated. At decoding step t, following the standard SD paradigm, TOOLSPEC first proposes a sequence of draft tokens $\tilde{\boldsymbol{y}}_{t:t+\gamma-1} = (\tilde{y}_t, \tilde{y}_{t+1}, \ldots, \tilde{y}_{t+\gamma-1})$, as predictions of the subsequent decoding steps. The target model $\mathcal{M}$ then verifies these draft candidates in parallel and accepts the longest prefix that is consistent with its probability distribution. When the drafting process incurs significantly lower latency than verification, maximizing the acceptance length of $\tilde{\boldsymbol{y}}$ directly improves the effective decoding parallelism and reduces the end-to-end latency.

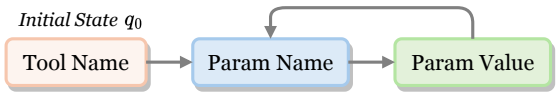

Figure 4: Illustration of state transitions in TOOLSPEC. Once tool calling is triggered by `<tool_call>`, the FSM enters the tool-name state q_t, and then alternates between the parameter-name state q_p and parameter-value state q_v until the tool call is completed.

4.2 Schema-aware Drafting

As discussed in Section 3.2, tool invocations from advanced LLMs adhere to predefined schema formats. This structural regularity enables deterministic drafting for large portions of the output. To exploit this property, TOOLSPEC constructs a finite-state machine (FSM) derived from the parsed tool schema. Formally, the FSM is defined as

$$\mathcal{F} = (\mathcal{Q}, \Sigma_{\text{schema}}, \delta, q_0), \tag{1}$$

where $\mathcal{Q}$ is the set of states, Σ_{schema} is the schema token alphabet, $\delta : \mathcal{Q} \times \Sigma_{\text{schema}} \to \mathcal{Q}$ is the transition function, and $q_0 \in \mathcal{Q}$ is the initial state.

The state space $\mathcal{Q}$ decomposes the tool-calling generation process into structural components:

- q_t: **tool-name state**, where the model selects a tool from the predefined tool set $\mathcal{C}_{\text{tools}}$.
- q_p: **parameter-name state**, where the model selects one parameter field of the chosen tool.
- q_v: **parameter-value state**, where the model generates the value of the selected parameter.
- q_o: **"others" state**, corresponding to general natural language generation outside tool calls.

As illustrated in Figure 4, once the special token `<tool_call>` is generated, the FSM transitions to the tool-name state q_t. In this state, drafting is executed via constrained decoding to generate candidate draft sequences. We formulate the proposed draft sequence $\tilde{\boldsymbol{y}}$ as the concatenation (denoted by $\oplus$) of the necessary structural elements:

$$\tilde{\boldsymbol{y}} = \tilde{\boldsymbol{y}}_{\text{prefix}} \oplus \tilde{\boldsymbol{y}}_{\text{tool_name}} \oplus \tilde{\boldsymbol{y}}_{\text{suffix}} \tag{2}$$

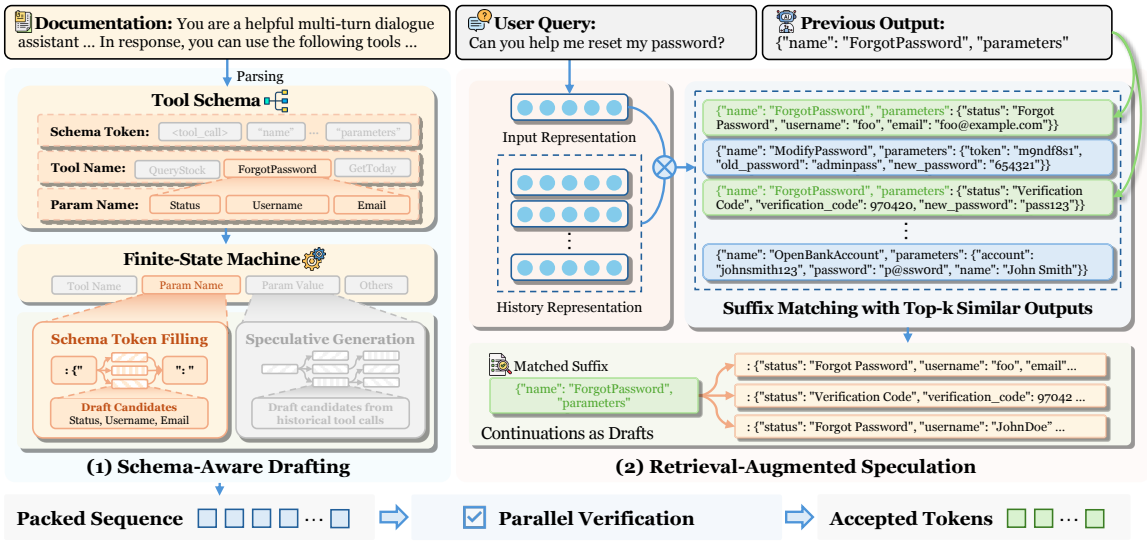

Figure 5: Illustration of tool-calling generation with TOOLSPEC. To accelerate structural tool calling, TOOLSPEC integrates two complementary drafting strategies: **(a) schema-aware drafting**, where a finite-state machine guides highly accurate drafting within the predefined tool schema; and **(b) retrieval-augmented speculation**, which reuses historical generation patterns as high-quality drafts to speed up recurrent tool calls.

where $\tilde{\boldsymbol{y}}_{\text{prefix}}$ and $\tilde{\boldsymbol{y}}_{\text{suffix}}$ denote the corresponding schema prefix and suffix associated with q_t, and $\tilde{\boldsymbol{y}}_{\text{tool_name}} \in \mathcal{C}_{\text{tools}}$ represents the candidate tool names. All potential drafts are subsequently verified in parallel by the target model $\mathcal{M}$.

Upon successful verification, the FSM transitions to the parameter-name state q_p, where $\mathcal{M}$ verifies the candidate parameter names associated with the selected tool in parallel. The FSM then transitions to the parameter-value state q_v. Because parameter values are typically unconstrained, TOOLSPEC eschews schema constraints at this stage and instead relies on a general SD algorithm to predict these variable fields.

Then, the FSM transitions between states according to schema-specific delimiters. For example, the delimiter `",` terminates the parameter-value state q_v and returns to q_p for the next parameter; while the delimiter `"}}"` marks completion of the tool call. Consequently, TOOLSPEC seamlessly alternates between deterministic schema token filling for structural states (e.g., q_t and q_p) and speculative generation for open-ended states (e.g., q_v and q_o).

4.3 Retrieval-augmented Speculation

While the schema defines the structural skeleton of a tool call and supports local drafting for constrained fields such as tool names, variable fields, particularly parameter values, depend heavily on the input context and cannot be inferred from the schema alone. Motivated by the recurring patterns in tool invocations, TOOLSPEC introduces a retrieval-augmented drafting strategy to accelerate tool-calling generation at the sequence level.

We maintain a datastore $\mathcal{H} = \{(\boldsymbol{h}_j, \boldsymbol{y}_j)\}_{j=1}^{|\mathcal{H}|}$ consisting of prior successful tool calls, where $\boldsymbol{h}_j \in \mathbb{R}^d$ denotes the hidden representation of the input question $\boldsymbol{x}_j$ at its final token, d is the hidden dimension, and $\boldsymbol{y}_j$ is the corresponding tool-call output. Given a query $\boldsymbol{x}_i$, we retrieve the top-k most similar historical tool calls:

$$\mathcal{I}_k = \text{Top-K}_{j \in \{1 \ldots |\mathcal{H}|\}} \, \text{sim}(\boldsymbol{h}_i, \boldsymbol{h}_j) \tag{3}$$

where $\text{sim}(\cdot)$ denotes cosine similarity and $\mathcal{I}_k$ the indices of the retrieved entries. Given the partially generated output $\boldsymbol{y}_{<t}$, we align it with each retrieved tool call $\boldsymbol{y}'$ by matching the suffix:

$$\boldsymbol{y}_{t-L:t-1} = \boldsymbol{y}'_{m-L:m-1} \tag{4}$$

where L denotes the suffix length and m indicates the corresponding match position in $\boldsymbol{y}'$. When a suffix match is found, TOOLSPEC extracts the next n tokens $\boldsymbol{y}'_{m:m+n-1}$, using these continuations as drafts. As shown in Figure 5, if the current suffix matches a historical invocation containing `..."parameters"`, the subsequent tokens (e.g., `:{"status": "Forgot Password"...`) are proposed as draft candidates. Intuitively, once the tool name and key–value pairs are aligned, the remaining structure of the tool call often follows recurring patterns, making $\boldsymbol{y}'_{m:m+n-1}$ high-quality drafts for subsequent value spans.

4.4 Parallel Verification

Following prior work (Li et al., 2024b; Cai et al., 2024), TOOLSPEC adopts a tree-based attention mechanism to verify multiple candidate draft sequences in parallel. Specifically, the candidate drafts are compacted into a single flattened sequence and fed into the target model $\mathcal{M}$. The model then performs a *single forward pass* to verify all draft sequences simultaneously using a tree-structured attention mask, as illustrated in Figure 1.

To support different drafting strategies, TOOLSPEC employs two types of draft-tree architectures: (1) **Static trees.** For open-ended states (e.g., q_v and q_o), TOOLSPEC adopts a general SD strategy (e.g., TR) for text generation, where a pre-defined static tree architecture is used for drafting; (2) **Dynamic trees.** For structural states (e.g., q_t and q_p) and retrieval-augmented speculation, TOOLSPEC constructs dynamic tree masks that adapt to the length of each draft sequence, thereby reducing unnecessary verification computation.

4.5 Algorithm

We present the pseudo code of TOOLSPEC in Algorithm 1. Concretely, at each decoding step, TOOLSPEC checks whether the current suffix matches any sequence in historical calls. If a match is found, it extracts the continuations as high-quality drafts. Otherwise, TOOLSPEC employs a finite-state machine (FSM) to guide schema-aware drafting, thereby accelerating tool-calling generation.

Algorithm 1 Tool-calling with TOOLSPEC

Require: target model $\mathcal{M}$; input query $\boldsymbol{x}$; tool documentation $\mathcal{D}$; historical datastore $\mathcal{H} = \{(\boldsymbol{h}_j, \boldsymbol{y}_j)\}_{j=1}^{|\mathcal{H}|}$; parameter k.
Ensure: Generated tool-call sequence $\boldsymbol{y}$.

```
Parse documentation D into schema S
Construct the FSM F = (Q, Σ_schema, δ, q_0) from S
y ← ∅, q ← q_0
Compute query representation h_x
Retrieve historical calls I_k ← Top-K_j sim(h_x, h_j)
while NotFinished(q, y) do
    // Retrieval-augmented speculation
    B_retrieval ← SuffixMatch(y, {y_j}_{j∈I_k})
    if B_retrieval ≠ ∅ then
        B ← B_retrieval
    else
        // Schema-aware drafting
        B ← SchemaAwareDraft(q, S, y)
    end if
    // Packed sequence with tree attention
    (z, A) ← PackDraftsAsTree(B)
    B̂ ← Verification(M, x, y, z, A)
    (ỹ*, ℓ) ← SelectLongestAcceptedPrefix(B̂)
    y ← y ⊕ ỹ*_{1:ℓ}
    // Update FSM state
    q ← δ(q, ỹ*_{1:ℓ})
end while
return y
```

5 Experiments

5.1 Experimental Setup

Models and Datasets We primarily evaluate our method using the Qwen2.5-Instruct (Yang et al., 2024) series, ToolLLaMA (Qin et al., 2024b) series, LLaMA-3.1-8B-Instruct, and LLaMA-3.2-3B-Instruct (Grattafiori et al., 2024). The evaluation covers four widely used tool-use benchmarks: API-Bank (Li et al., 2023), ToolAlpaca (Tang et al., 2023), BFCLv2 (Patil et al., 2025), and ToolBench (Qin et al., 2024b). Additional details of these datasets are provided in the Appendix A.1.

Implementation Details The retrieval parameter k is set to 10. For suffix matching, we follow PLD (Saxena, 2023) and consider suffix lengths $L \in \{5, 6, 7\}$. The continuations are extracted using predefined lengths $n \in \{32, 16, 8, 8\}$. We adopt Token Recycling (TR) (Luo et al., 2025) as the general speculative strategy in TOOLSPEC. Following prior work, we adopt speculative sampling (Leviathan et al., 2023) as our acceptance strategy, with a batch size of 1. Additional implementation details are provided in the Appendix A.2.

Baselines We primarily compare TOOLSPEC to three existing *plug-and-play* methods: PLD (Saxena, 2023), TR (Luo et al., 2025), and SAM-Decoding (SAMD) (Hu et al., 2025). For SAMD, we construct the static SAM from the input context and adopt TR as its decoding strategy. For comparison, we also include the advanced training-required methods Eagle-2 (Li et al., 2024a) and Eagle-3 (Li et al., 2025) in our main results.

Evaluation Metrics We evaluate TOOLSPEC using three widely adopted metrics: the mean number of accepted tokens (#MAT), throughput (tokens/s), and wall-clock speedup relative to vanilla autoregressive decoding. From a theoretical perspective, TOOLSPEC preserves the output distribution of the target LLM, thereby obviating the need for extensive evaluation of generation quality. Nevertheless, we report performance on API-Bank and ToolAlpaca in Appendix A.3 for reference.

5.2 Main Results

Table 3 compares the inference efficiency of TOOLSPEC with prior plug-and-play methods on widely

Models	Methods	API-Bank			ToolAlpaca			BFCLv2			Overall Speedup
		#MAT	Tokens/s	Speedup	#MAT	Tokens/s	Speedup	#MAT	Tokens/s	Speedup	
Qwen2.5-7B -Instruct	Vanilla	1.00	32.76	1.00×	1.00	32.72	1.00×	1.00	32.73	1.00×	1.00×
	PLD	2.37	64.56	1.97×	2.10	60.75	1.86×	2.24	62.52	1.91×	1.91×
	TR	3.09	78.95	2.41×	2.87	71.28	2.18×	2.89	73.64	2.25×	2.28×
	SAMD	3.60	83.96	2.56×	2.62	63.57	1.94×	2.92	70.70	2.16×	2.22×
	TOOLSPEC	**4.90**	**129.40**	**3.95×**	**4.55**	**113.54**	**3.47×**	**4.73**	**117.17**	**3.58×**	**3.67×**
Qwen2.5-14B -Instruct	Vanilla	1.00	19.50	1.00×	1.00	19.02	1.00×	1.00	19.31	1.00×	1.00×
	PLD	1.98	35.30	1.81×	2.00	34.85	1.83×	2.02	35.34	1.83×	1.82×
	TR	3.01	46.94	2.41×	2.85	41.97	2.21×	2.99	46.34	2.40×	2.34×
	SAMD	2.93	45.86	2.35×	2.62	38.48	2.02×	2.78	42.29	2.19×	2.19×
	Eagle-2*	3.87	46.80	2.40×	3.58	36.24	1.91×	3.67	39.39	2.04×	2.12×
	TOOLSPEC	**4.43**	**75.27**	**3.86×**	**4.02**	**60.86**	**3.20×**	**4.15**	**65.07**	**3.37×**	**3.48×**
LLaMA-3.1-8B -Instruct	Vanilla	1.00	28.21	1.00×	1.00	27.69	1.00×	1.00	27.83	1.00×	1.00×
	PLD	2.35	60.72	2.15×	2.27	58.28	2.10×	2.30	58.72	2.11×	2.12×
	TR	3.34	71.10	2.52×	3.32	67.69	2.44×	3.29	66.79	2.40×	2.45×
	SAMD	3.20	69.68	2.47×	3.12	65.82	2.38×	3.14	66.24	2.38×	2.41×
	Eagle-3*	3.34	47.24	1.67×	2.93	40.37	1.46×	3.17	42.86	1.54×	1.56×
	TOOLSPEC	**5.80**	**125.59**	**4.45×**	**5.26**	**112.26**	**4.05×**	**5.31**	**113.55**	**4.08×**	**4.19×**
LLaMA-3.2-3B -Instruct	Vanilla	1.00	32.60	1.00×	1.00	33.20	1.00×	1.00	32.91	1.00×	1.00×
	PLD	2.19	69.81	2.14×	2.06	61.41	1.85×	2.11	64.83	1.97×	1.99×
	TR	3.11	75.50	2.32×	3.03	73.45	2.31×	3.06	76.35	2.32×	2.32×
	SAMD	2.99	72.79	2.18×	2.68	71.84	2.16×	2.75	71.09	2.16×	2.17×
	TOOLSPEC	**5.20**	**133.30**	**4.09×**	**4.52**	**112.22**	**3.38×**	**4.78**	**115.84**	**3.52×**	**3.66×**

Table 3: Comparison of inference efficiency between TOOLSPEC and prior plug-and-play methods. We report the mean number of accepted tokens (#MAT), average decoding speed (tokens/s), and wall-clock speedup relative to vanilla autoregressive decoding. The best results are shown in **boldface**, and the second-best results are underlined. *The Eagle series, a training-required SD method, is included for reference only.

used tool-use tasks. The results reveal the following key findings: (1) TOOLSPEC consistently outperforms existing approaches, achieving speedups of 3.5×∼4.2× over standard autoregressive decoding across a range of models and tasks. Compared with the previous best method, it yields up to a **71**% relative improvement in speedup (from 2.45× to 4.19×). (2) The efficiency gains of TOOLSPEC are primarily attributed to a high number of accepted tokens (denoted as #MAT), ranging from 4.02 to 5.80 across different settings. Unlike model-based SD methods such as Eagle-3, TOOLSPEC does not require an additional drafting module and introduces only minimal drafting overhead (approximately 6%, as shown in Figure 11). This low overhead enables the increased #MAT to translate more effectively into end-to-end speedup.

Discussion on Eagle Performance We observe that the #MAT achieved by the Eagle series is lower than that of prior plug-and-play methods, such as TR, as shown in Table 3. We hypothesize that this gap arises because Eagle models are not trained on tool-calling formats, making such inputs out-of-distribution for these models. In contrast, as discussed in Section 3.2, tool-calling traces exhibit structural regularities and recurring invocation patterns. These properties favor SD methods such as TR, which leverage historical traces to construct draft sequences. Building on this insight, TOOLSPEC exploits the schema-constrained nature of tool-calling tasks, enabling more accurate draft generation and achieving higher overall efficiency.

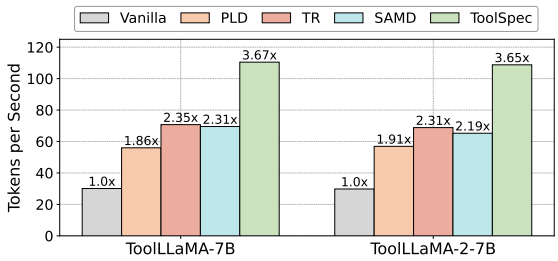

Figure 6: Speedup comparison between TOOLSPEC and prior plug-and-play methods on ToolBench.

Results on ToolLLaMA In addition to general-purpose LLMs, we further evaluate TOOLSPEC on two variants of ToolLLaMA, a model specifically trained for tool-use tasks. Figure 6 presents a comparison of inference efficiency between TOOLSPEC and prior plug-and-play SD methods on ToolBench. The results show that TOOLSPEC achieves a speedup of approximately 3.7× over standard autoregressive decoding, demonstrating its effec-

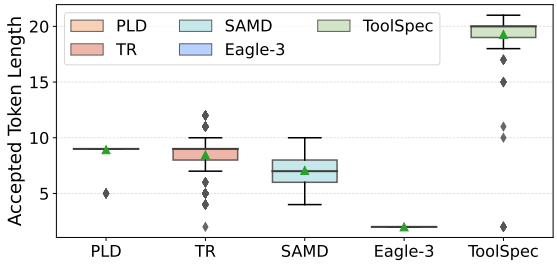

Figure 7: Distribution of accepted token lengths in the first two decoding steps using LLaMA-3.1-8B-Instruct on API-Bank. Green triangles represent mean values, and black diamonds denote outliers.

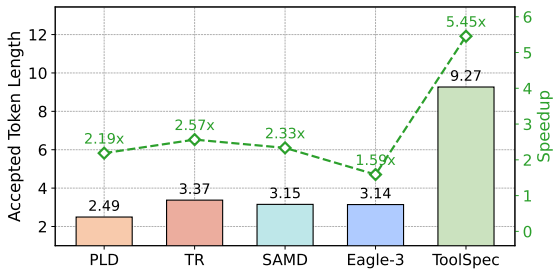

Figure 8: Mean number of accepted tokens (#MAT) and speedups for repeated tool calls. Results are obtained using LLaMA-3.1-8B-Instruct on API-Bank.

tiveness on specialized tool-use LLMs.

6 Analysis

This section presents a comprehensive analysis of TOOLSPEC. Specifically, we evaluate its effectiveness (§6.1), examine the impact of adherence to the tool-calling format (§6.2), and analyze the time allocation across different modules (§6.3).

6.1 Effectiveness of TOOLSPEC

Statistical Analysis We first conduct a statistical evaluation to assess the effectiveness of the proposed modules, including: **1) Schema-aware Drafting.** Figure 7 reports accepted token lengths in the first two decoding steps, corresponding to the tool-name state q_t and the first parameter-name state q_p in TOOLSPEC. As shown, schema-aware drafting enables TOOLSPEC to generate an average of 19.3 tokens across these steps. In contrast, prior approaches such as TR produce an average of 8.5 tokens, while Eagle-3 yields only 2 tokens on average, equivalent to standard autoregressive decoding. These results demonstrate the clear advantage of schema-aware drafting for tool-calling generation. **2) Retrieval-augmented Speculation.** Figure 8 presents #MAT for repeated tool calls. The results indicate that TOOLSPEC consistently

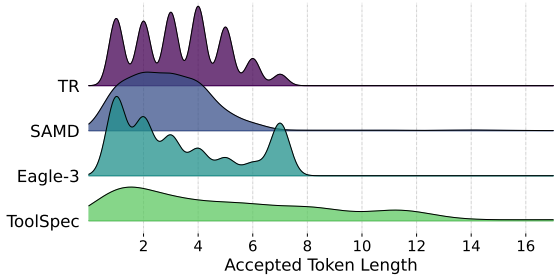

Figure 9: Distribution of accepted token lengths using LLaMA-3.1-8B-Instruct on API-Bank. Values exceeding 17 are truncated for clearer visualization.

Methods	#MAT	Tokens/s	Speedup
Vanilla	1.00	28.21	1.00×
TOOLSPEC	**5.80**	**125.59**	**4.45**×
w/o SAD	5.58	118.99	4.22×
w/o RAS	4.29	94.53	3.35×
w/o both	3.62	76.00	2.69×

Table 4: Ablation study of TOOLSPEC modules. "*SAD*" and "*RAS*" denote schema-aware drafting and retrieval-augmented speculation, respectively. Results are obtained with LLaMA-3.1-8B-Instruct on API-Bank.

outperforms prior methods, achieving an average of 9.3 accepted tokens per decoding step and approximately 5.5× speedup. This improvement highlights the effectiveness of the proposed retrieval strategy in handling repetitive tool invocations.

Acceptance Distribution Figure 9 presents the distribution of accepted token lengths for TOOLSPEC in comparison with prior methods. While existing approaches typically yield accepted token lengths of fewer than 8 tokens, TOOLSPEC shifts the distribution toward longer accepted lengths, with 22% of decoding steps exceeding 8 tokens. Furthermore, TOOLSPEC exhibits a pronounced long-tail distribution: nearly 4% of decoding steps attain accepted token lengths between 17 and 33. This long-tail behavior stems from retrieval-augmented speculation, which reuses historical tool-call trajectories to produce longer and more accurate drafts. These consistently higher accepted token lengths contribute to the overall speedup achieved by TOOLSPEC.

Ablation Study Table 4 illustrates an ablation study of TOOLSPEC and its proposed components. The results indicate that both schema-aware drafting and retrieval-augmented speculation are critical to the efficiency of TOOLSPEC. When these components are removed, similar to PLD, TOOLSPEC

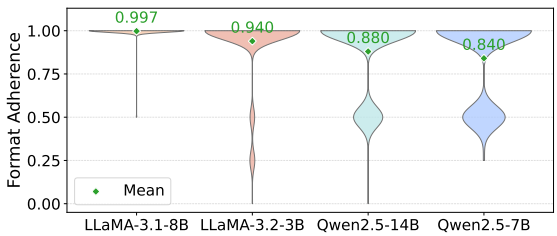

Figure 10: Format adherence of different models in tool-calling generation. Results are obtained on API-Bank.

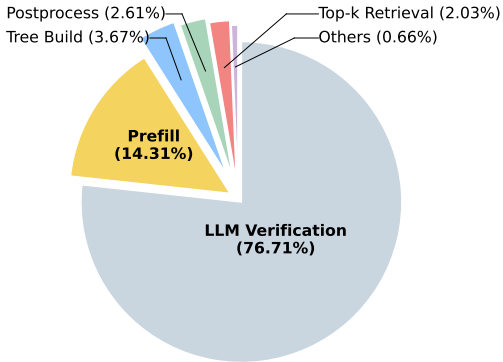

Figure 11: Time allocation for each operation when LLMs respond to a query. Results are obtained using LLaMA-3.1-8B-Instruct on API-Bank.

relies on suffix matching within the input context, resulting in slightly higher speedup than TR.

6.2 Impact of Format Adherence

Beyond the analysis in Section 3.2, we further evaluate how strictly tool-use LLMs follow the required schema during tool-calling generation. We define strict format adherence using two criteria: 1) the outputs must be correctly parsed as valid JSON; and 2) any extraneous natural language, Markdown fences, XML-like tags, or malformed JSON structures are treated as violations. The results reveal a strong positive correlation between format adherence and the speedup achieved by TOOLSPEC, with a Pearson correlation coefficient of approximately 0.90. Notably, LLaMA-3.1-8B-Instruct achieves the highest adherence score of 0.997, corresponding to a $4.45\times$ speedup on API-Bank.

6.3 Time Allocation

Figure 11 presents the time allocation across the primary latency-intensive components of TOOLSPEC. As shown, the decoding stage of the target model dominates overall latency, accounting for 91.0% of the total runtime. In contrast, the additional operations introduced by TOOLSPEC, including top-k retrieval and draft tree construction, incur only modest overhead, contributing just 5.7% of the total latency. Despite this minimal overhead, TOOLSPEC achieves an approximately $4\times$ speedup in tool-calling generation, demonstrating its effectiveness in reducing end-to-end latency.

7 Conclusion

This work presents TOOLSPEC, a schema-aware, retrieval-augmented speculative decoding approach for accelerating tool-calling generation. It requires no additional training or auxiliary models, enabling seamless integration into existing LLM workflows. Extensive experiments across various LLMs and tool-use benchmarks demonstrate that TOOLSPEC achieves a $3.5\times\sim4.2\times$ speedup while preserving output distributions. Further analysis highlights the effectiveness of the proposed components and the impact of format adherence on the efficiency of TOOLSPEC. We hope this work advances efficient tool use in LLMs and provides practical insights into scalable tool-calling generation.

Limitations

Due to computational constraints, our experiments do not include larger-scale language models such as Qwen2.5-72B-Instruct (Yang et al., 2024) and LLaMA-3.1-70B-Instruct (Grattafiori et al., 2024). Nevertheless, we expect TOOLSPEC to generalize well to these models and achieve further improvements in tool-calling generation efficiency. We include experimental results of Qwen2.5-32B-Instruct in Appendix A.6. Additionally, our approach assumes strong adherence to predefined tool schemas and the existence of recurring invocation patterns, which are consistent with real-world application scenarios characterized by frequent tool usage. As discussed in Sections 3 and 6.2, we anticipate that the effectiveness of TOOLSPEC will continue to improve as LLM capabilities advance.

Ethics Statement

The datasets used in our experiments are publicly released and labeled through interaction with humans in English. In this process, user privacy is protected, and no personal information is contained in the dataset. The scientific artifacts that we used are available for research with permissive licenses. The use of these artifacts in this paper is consistent with their intended use. Therefore, we believe that our research work meets ACL's ethical standards.

References

Anthropic. 2024. Claude 3.5 sonnet.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In Proceedings of the 41st International Conference on Machine Learning, volume 235 of Proceedings of Machine Learning Research, pages 5209–5235. PMLR.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided language models. In Proceedings of the 40th International Conference on Machine Learning, volume 202 of Proceedings of Machine Learning Research, pages 10764–10799. PMLR.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. Tora: A tool-integrated reasoning agent for mathematical problem solving. In The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, and 1 others. 2024. The llama 3 herd of models. Preprint, arXiv:2407.21783.

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. 2025. SAM decoding: Speculative decoding via suffix automaton. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 12187–12204, Vienna, Austria. Association for Computational Linguistics.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. Preprint, arXiv:2503.09516.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. 2024. An LLM compiler for parallel function calling. In Proceedings of the 41st International Conference on Machine Learning, volume 235 of Proceedings of Machine Learning Research, pages 24370–24391. PMLR.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In Proceedings of the 40th International Conference on Machine Learning, volume 202 of Proceedings of Machine Learning Research, pages 19274–19286. PMLR.

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.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024a. EAGLE-2: Faster inference of language models with dynamic draft trees. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 7421–7432, Miami, Florida, USA. Association for Computational Linguistics.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. EAGLE: speculative sampling requires rethinking feature uncertainty. In Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024. OpenReview.net.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In The Thirty-ninth Annual Conference on Neural Information Processing Systems.

Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, and Dongliang Xu. 2025. Turning trash into treasure: Accelerating inference of large language models with token recycling. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 6816–6831, Vienna, Austria. Association for Computational Linguistics.

Daniel Nichols, Prajwal Singhania, Charles Jekel, Abhinav Bhatele, and Harshitha Menon. 2025. Optimizing agentic language model inference via speculative tool calls. Preprint, arXiv:2512.15834.

OpenAI. 2024. Openai o1 system card. CoRR, abs/2412.16720.

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, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025, Proceedings of Machine Learning Research. PMLR / OpenReview.net.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive apis. In Advances in Neural Information Processing Systems, volume 37, pages 126544–126565. Curran Associates, Inc.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, and 24 others. 2024a. Tool learning with foundation models. ACM Comput. Surv., 57(4).

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. 2024b. Toollm: Facilitating large language models to master 16000+ real-world apis. In The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net.

Apoorv Saxena. 2023. Prompt lookup decoding.

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. In Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. CoRR, abs/2306.05301.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. 2023. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In Findings of the Association for Computational Linguistics: EMNLP 2023, pages 3909–3925, Singapore. Association for Computational Linguistics.

Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. 2025. SWIFT: on-the-fly self-speculative decoding for LLM inference acceleration. In The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025. OpenReview.net.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In Findings of the Association for Computational Linguistics: ACL 2024, pages 7655–7671, Bangkok, Thailand. Association for Computational Linguistics.

Yechen Xu, Xinhao Kong, Tingjun Chen, and Danyang Zhuo. 2024. Conveyor: Efficient tool-aware llm serving with tool partial execution. Preprint, arXiv:2406.00059.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. CoRR, abs/2505.09388.

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, and 22 others. 2024. Qwen2.5 technical report. CoRR, abs/2412.15115.

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.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. 2025. DeepResearcher: Scaling deep research via reinforcement learning in real-world environments. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 414–431, Suzhou, China. Association for Computational Linguistics.

Dongsheng Zhu, Weixian Shi, Zhengliang Shi, Zhaochun Ren, Shuaiqiang Wang, Lingyong Yan, and Dawei Yin. 2025. Divide-then-aggregate: An efficient tool learning method via parallel tool invocation. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025, pages 28859–28875. Association for Computational Linguistics.

A Experimental Details

In this section, we present our experimental details, including dataset descriptions, implementation specifics, extended results, and additional analyses that complement our main findings.

A.1 Datasets

Our evaluation encompasses four widely used tool-use benchmarks. The details of the evaluation data are illustrated as follows:

- **API-Bank** (Li et al., 2023): a three-level evaluation framework comprising 73 diverse APIs. It evaluates an LLM's ability to select and invoke tools through natural multi-turn dialogues across varying levels of difficulty.
- **ToolAlpaca** (Tang et al., 2023): a benchmark for assessing general-purpose tool-use capabilities of LLMs. The evaluation dataset consists of two subsets: (i) a simulated subset with 10 synthetic tools, and (ii) a real-world subset with 11 APIs from diverse domains.
- **BFCLv2** (Patil et al., 2025): a comprehensive benchmark covering a wide range of scenarios, including single-step reasoning, multi-step tool use, real-time execution, etc. We evaluate TOOLSPEC under both single-function and multi-function settings.
- **ToolBench** (Qin et al., 2024b): a large-scale, high-quality benchmark featuring multi-tool and multi-turn interactions over 16,464 real-world APIs, automatically constructed using ChatGPT. We evaluate TOOLSPEC on its single-tool subset.

A.2 Implementation Details

The retrieval parameter k is set to 10, such that the top-10 candidates are retained at each retrieval step. For suffix matching, we follow PLD (Saxena, 2023) and consider suffix lengths $L \in \{5, 6, 7\}$, applying a longest-first strategy (from 7 to 5). Once a match is found, we extract continuations following the matched suffix in sequential order, using predefined lengths $n \in \{32, 16, 8, 8\}$ to construct draft candidates. We adopt Token Recycling (TR) (Luo et al., 2025) as the general speculative generation strategy in TOOLSPEC. All experiments were conducted using PyTorch 2.5.1 on 2×NVIDIA A100-PCIE GPUs (40GB) with 40 CPU cores under CUDA 12.4. Inference for our method and all baselines was performed using the Hugging Face transformers package. Following prior work, we adopt speculative sampling (Leviathan et al., 2023) as our acceptance strategy, with a batch size of 1.

Models	Methods	API-Bank	ToolAlpaca
Qwen2.5-7B	Vanilla	54.94	58.94
	TOOLSPEC	54.94	58.94
Qwen2.5-14B	Vanilla	54.61	58.94
	TOOLSPEC	54.61	59.60
LLaMA-3.1-8B	Vanilla	59.30	48.34
	TOOLSPEC	59.30	48.34
LLaMA-3.2-3B	Vanilla	50.08	41.72
	TOOLSPEC	50.08	41.72

Table 5: Performance comparison between TOOLSPEC and vanilla autoregressive decoding across different models on two widely used tool-use benchmarks.

A.3 Performance Comparison

Table 5 illustrates the accuracy comparison between TOOLSPEC and vanilla autoregressive decoding across different models on API-Bank and ToolAlpaca. The results demonstrate that TOOLSPEC preserves the original performance of autoregressive decoding, achieving consistently comparable accuracy across various models and datasets. The slight discrepancy observed for Qwen2.5-14B on ToolAlpaca can be attributed to fp16 precision.

A.4 Hyperparameters in TOOLSPEC

Table 6 and Table 7 show the speedup of TOOLSPEC under different hyperparameter settings, including continuation length and the retrieval parameter k. The evaluation is conducted on 100 instances sampled from the training set of API-Bank (Li et al., 2023). Table 6 shows that shorter continuation lengths (e.g., $\{16, 8, 4, 4\}$) lead to suboptimal #MAT and speedup. Although longer continuation lengths (e.g., $\{64, 32, 16, 8\}$) slightly improve #MAT, they also introduce additional computational overhead and reduce the overall speedup. Therefore, we adopt continuation lengths of $\{32, 16, 8, 8\}$ in our experiments. We further compare this setting with uniform continuation lengths of $\{16, 16, 16, 16\}$, and the results show that allocating longer continuations to top-ranked candidates brings greater benefits. Similarly, Table 7 shows that setting the retrieval parameter k to 10 achieves the best speedup in our experiments.

A.5 Memory Cost

The additional memory overhead introduced by TOOLSPEC is modest. The primary persistent cost

Continuations	#MAT	Tokens/s	Speedup
$\{16, 8, 4, 4\}$	5.52	115.07	4.08×
$\{32, 16, 8, 8\}$	5.75	**124.77**	**4.42**×
$\{64, 32, 16, 8\}$	**5.77**	122.60	4.35×
$\{16, 16, 16, 16\}$	5.60	117.95	4.18×

Table 6: Speedup of TOOLSPEC under different continuation lengths. Results are obtained with LLaMA-3.1-8B-Instruct on API-Bank.

k	#MAT	Tokens/s	Speedup
1	5.13	113.18	4.01×
5	5.56	119.70	4.24×
10	**5.75**	**124.77**	**4.42**×
15	5.69	123.22	4.37×

Table 7: Speedup of TOOLSPEC with different retrieval parameter values k. Results are obtained with LLaMA-3.1-8B-Instruct on API-Bank.

comes from the *output memory*, which stores the hidden state of each input question, along with its corresponding output tokens for each memorized sample. On API-Bank, this memory requires only 4.9 MB for 597 entries, corresponding to approximately 0.0082 MB per entry. To support larger datasets, TOOLSPEC can maintain a fixed-size memory bank, such as 1,000 memorized samples, thereby bounding the output memory while preserving the benefits of retrieval-augmented speculation. We leave the design of such memory management strategies for future investigation.

A.6 Results on Qwen2.5-Instruct-32B

Table 8 compares the speedup of TOOLSPEC with prior plug-and-play methods on API-Bank using Qwen2.5-32B-Instruct. The results demonstrate that TOOLSPEC outperforms these baselines and achieves up to a 3× speedup over standard autoregressive decoding. Figure 12 shows that the relative speedup gain of TOOLSPEC over the strongest reproduced baseline becomes larger as the model size increases. These results suggest that TOOLSPEC scales favorably with model size and provides increasingly pronounced efficiency benefits over existing plug-and-play approaches.

A.7 Case Study

Figure 13 presents a case study of TOOLSPEC on API-Bank using LLaMA-3.1-8B-Instruct, where the current user query is similar to previously observed queries. As shown, TOOLSPEC first applies schema token filling to generate the required

Models	#MAT	Tokens/s	Speedup
Vanilla	1.00	17.23	1.00×
PLD	1.91	28.24	1.64×
TR	2.90	30.30	1.76×
SAMD	<u>2.99</u>	<u>30.73</u>	<u>1.78×</u>
TOOLSPEC	**4.72**	**52.42**	**3.04**×

Table 8: Speedup of TOOLSPEC and prior plug-and-play methods using Qwen2.5-32B-Instruct on API-Bank. The best results are shown in **boldface**, and the second-best results are <u>underlined</u>.

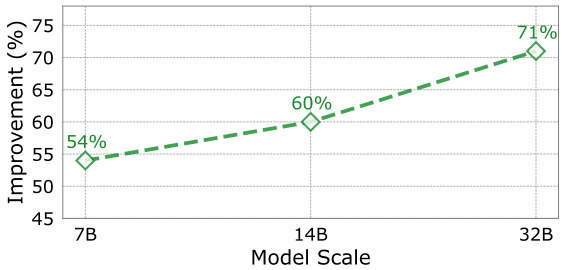

Figure 12: Relative speedup improvement of TOOLSPEC over the previous best method. Results are obtained using the Qwen series on API-Bank.

tool name and schema-constrained tokens. It then retrieves a highly similar historical tool call and reuses its continuation as draft tokens, yielding up to 33 accepted tokens in subsequent decoding steps. This case demonstrates the effectiveness of schema token filling and retrieval-augmented speculation.

Similarly, Figure 14 presents a case with similar user queries but different tool-argument values. TOOLSPEC reuses the continuation of a previous output as draft tokens, yielding 16 accepted tokens in the third decoding step. It then corrects the user-specific travel information and completes the tool call through general speculative generation. This case highlights the strength of TOOLSPEC in repeated tool-calling scenarios, where most tokens can be drafted from historical invocations while the target LLM focuses on key argument values.

Previous User Query

Can you help me modify a reminder with the description "Submit report" at 9 am on March 17th, 2023? My username is testuser, password is testpass, and email is testuser@example.com. The description is "Submit report" and it should be triggered at 9 am on March 17th, 2023.

Historical Tool Call

```
{
    "name": "ModifyReminder",
    "parameters": {
        "token": "p9o8i7u6y5t4k3e2w1q",
        "content": "Submit report",
        "time": "2023-03-17 09:00:00"
    }
}
```

User Query

Can you help me modify a reminder? I want to change the description to "Submit report" and the time to "2023-03-17 09:00:00". My username is testuser and password is testpass.

Tool Call

Step 1 (Schema Token Filling, accepted token=11):
<tool_call>{"name": "ModifyReminder

Step 2 (Schema Token Filling, accepted token=8):
", "parameters": {"token": "

Step 3 (Retrieval-Augmented Speculation, accepted token=33):
p9o8i7u6y5t4k3e2w1q", "content": "Submit report", "time": "2023

Step 4 (Retrieval-Augmented Speculation, accepted token=16):
-03-17 09:00:00"}}</tool_call>

Figure 13: Case study of TOOLSPEC with similar user queries on API-Bank using LLaMA-3.1-8B-Instruct.

Previous User Query

Retrieve flight options and hotel availability for a given destination and travel dates.

Historical Tool Call

```
{
    "name": "FlightSearch",
    "parameters": {
        "source": "New York",
        "destination": "Los Angeles",
        "travel_dates": "2024-08-01-2024-08-15"
    }
}
```

User Query

Retrieve flight options and hotel availability for New York to Los Angeles on January 5th, 2022.

Tool Call

Step 1 (Schema Token Filling, accepted token=11):
<tool_call>{"name": "FlightSearch

Step 2 (Schema Token Filling, accepted token=8):
", "parameters": {"source": "

Step 3 (Retrieval-Augmented Speculation, accepted token=16):
New York", "destination": "Los Angeles", "travel_dates": "202

Step 4 (Retrieval-Augmented Speculation, accepted token=4):
2-01-

Step 5 (Speculative Generation, accepted token=7):
05"}}</tool_call>

Figure 14: Case study of TOOLSPEC under different tool-argument values using LLaMA-3.1-8B-Instruct.