Toolscaler: Scalable Generative Tool Calling via Structure-Aware Semantic Tokenization

Yunyue Su[1], **Jingshuai Zhang**[1], **Bowen Fang**[1,2,3], **Wen Ye**[1,2], **Jinghao Zhang**[1], **Bowen Song**[4*], **Weiqiang Wang**[4], **Qiang Liu**[1], **Liang Wang**[1*]

[1]New Laboratory of Pattern Recognition (NLPR),
Institute of Automation, Chinese Academy of Sciences (CASIA)
[2]School of Artificial Intelligence, University of Chinese Academy of Sciences
[3]Zhongguancun Academy [4]Ant Group
{yunyue.su, yewen2025}@ia.ac.cn, {zhangjs0711, bwn.fang}@gmail.com,
jinghao.zhang@cripac.ia.ac.cn, {bowen.sbw, weiqiang.wwg}@antgroup.com,
{qiang.liu, wangliang}@nlpr.ia.ac.cn

Abstract

Enhancing large language models (LLMs) with external tools has become a promising approach for solving complex tasks. As the number of available tools grows, context-based prompting methods increasingly rely on retrieval mechanisms. A common solution is to represent each tool with a unique token and train LLMs to generate the corresponding token during inference. However, this approach suffers from linear growth in representation space, leading to scalability challenges. It also limits generalization to novel or rare tools and underutilizes collaborative signals among tools in downstream tasks. In this paper, we propose **Toolscaler**[1], a generative tool invocation framework that introduces structure-aware semantic tokenization to encode tools as discrete code sequences. This method ensures similar tools share subtokens, enabling compression of the representation space and facilitating token sharing for new tools. We further introduce a post-guided, multistage iterative training strategy on a shared backbone model, where collaborative signals from downstream tasks guide the dynamic refinement of tool representations. Extensive experiments on the ToolBench dataset, which includes over 47,000 APIs, demonstrate the effectiveness of Toolscaler across various tasks, showcasing its potential as a scalable and generalizable generative tool-using paradigm in large-scale tool usage scenarios.

1 Introduction

Large language models (LLMs) improve their ability to interact with the real world through integration with tools, such as calculators, databases, etc.(Parisi et al., 2022; Schick et al., 2023; Thoppilan et al., 2022), and are proficient in handling external input, performing actions, and autonomously completing tasks (Wu et al., 2023b; Liu et al., 2023b). However, as the number of tools grows to tens of thousands, existing methods for tool retrieval and execution struggle to scale effectively.

While various approaches have been proposed to integrate tools into LLMs (Mialon et al., 2023; Yang et al., 2023b), including context-based prompting (Qin et al., 2024; Paranjape et al., 2023; Yao et al., 2022) and fine-tuning with tool description (Borgeaud et al., 2022; Guu et al., 2020; Puig et al., 2018; Shuster et al., 2021), they still face challenges in large-scale tool settings. Context-based prompting methods are inherently constrained by the input length limitation of LLMs, making it infeasible to include all tools within a single prompt and requiring external retrievers to select a small subset of candidate tools. On the other hand, fine-tuning-based methods that integrate tools into model parameters (Wang et al., 2024b; Hao et al., 2023) often rely on assigning each tool a unique identifier (ID) (Liu et al., 2024c; Yuan et al., 2023), which introduces several limitations in large-scale scenarios. First, the vocabulary size grows linearly with the number of tools, resulting in higher memory consumption and a larger decoding space, which increases the inference burden (Kang and McAuley, 2018; Sun et al., 2019). Second, the data sparsity and long-tail distribution of tool usage not only hinder the learning of reliable representations for infrequent tools, but also make it difficult to incorporate newly introduced tools without additional retraining or architectural changes. Third, since ID embeddings are learned independently, they fail to capture functional similarities or collaborative relationships among tools, further limiting generalization and reuse across tasks.

To address these limitations, we propose Toolscaler, a unified generative framework that provides a scalable and semantically structured rep-

*Corresponding authors
[1]https://github.com/OPilgrim/Toolscaler

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 556–578
November 4-9, 2025 ©2025 Association for Computational Linguistics

resentation of large-scale tools, enabling simultaneous tool retrieval and calling during generation. **First**, we introduce *structure-aware semantic tokenization*, which assigns each tool a compact sequence of discrete codes (Rajput et al., 2024; Singh et al., 2024; Wang et al., 2024d; Zhu et al., 2024) derived from its semantic embedding. These semantic embeddings are obtained by compressing tool knowledge into a small number of special tokens that encode functional and behavioral information. To generate the code sequences, we employ a lightweight *deep residual k-means* algorithm over the semantic embedding space for centroid assignment, and use the resulting centroids to initialize the embeddings of code tokens. The discrete codes are then dynamically refined via *post-guided training* to ensure that semantically or functionally similar tools share similar subtokens. This code-based tokenization facilitates representation compactness and encourages knowledge sharing across tools, while also enabling approximate similarity estimation (e.g., via Hamming distance) without additional model training—thus offering scalability to newly added or unseen tools. Its hierarchical structure enables a logarithmic compression of the tool vocabulary space, significantly reducing the decoding overhead compared to linear ID-based indexing. **Second**, we unify semantic tokenization, retrieval, and calling into a single generative modeling framework. This design allows for *multistage iterative training*, where the model progressively integrates tool knowledge—from basic documents to usage contexts and invocation workflows—across stages. **Finally**, in later training iterations, *downstream collaboration signals* are leveraged to refine tokenization strategies, allowing the model to dynamically adapt to tool usage patterns and improve its generative capabilities over time.

In summary, our work contributes the following key aspects:

- **Robust tool representation**: We employ semantic compress and deep residual k-means clustering to obtain the discrete structure-aware semantic code sequence, which can represent large-scale toolsets with minimal space overhead. Thanks to their structured composition, these code sequences also enable effective knowledge transfer to unseen tools, supporting robust generalization and scalability.
- **Dynamically updated strategy**: We adopt a post-guided training strategy that integrates tool knowledge from both documentation and latent logic embedded in downstream tasks—such as co-occurrence patterns and shared usage contexts—enabling dynamic refinement of code sequence generation.
- **Unified framework**: We employ a unified generative framework built upon a single LLM to jointly model tool tokenization, retrieval, and calling, thereby reducing information loss and enhancing cross-task knowledge transfer.
- **Empirical evaluation**: Extensive experiments conducted on the large-scale ToolBench dataset, collected from real-world sources, demonstrate that the Toolscaler framework achieves outstanding performance in diversity tool usage scenarios, highlighting its effectiveness and broad applicability.

2 Related Work

LLM with Tool Augmentation. Enhancing the ability of LLMs to solve complex problems by equipping them with tools for various tasks has demonstrated strong potential(Vemprala et al., 2024; Qin et al., 2023a; Wu et al., 2023a; Qian et al., 2023; Song et al., 2023; Zhuang et al., 2023; Gao et al., 2023a). By accessing external tools, LLMs can be endowed with real-time factual knowledge(Yang et al., 2023a), coding and debugging capabilities (Chen et al., 2022; Gao et al., 2023b; He-Yueya et al., 2023; Lyu et al., 2023; Xie et al., 2023; Liu et al., 2023a), multimodal functionalities (Gupta and Kembhavi, 2023; Shen et al., 2023; Lu et al., 2023), domain-specific expertise (Jin et al., 2024), and the ability to interact with the virtual or physical world (Brohan et al., 2023b; Huang et al., 2022b, 2023; Singh et al., 2023). Thanks to the powerful contextual learning ability (Brown et al., 2020), it is possible to enable LLMs to use tools simply by displaying examples within the prompt, without the need for training(Mekala et al., 2024; Khot et al., 2022). Therefore, most methods focus on guiding LLMs to mimic human task solving processes and generate plans (Zheng et al., 2024c; Liu et al., 2024d; Ahn et al., 2022; Huang et al., 2022a; Ye et al., 2023), and improving plans by incorporating execution feedback (Wang et al., 2024a; Shinn et al., 2024), thus combining reasoning with action. However, context-based learning methods are prone to hallucinations and are limited by inadequate context capacity when faced with large-scale tools. Although the tool retrieval stage is widely used, including trained additional retriever to rank top-k candidates from a large number of tools based on similarity to the query to enhance the generation process (Zheng et al., 2024b; Patil et al., 2025; Chen et al., 2024; Qin et al., 2023b). Such strategies do not improve the model's understanding of external tool knowledge, and maintaining dense retrieval databases and document indices can lead to inefficiency and difficulties in optimizing within an end-to-end agent framework.

Tool Learning. To address this problem, a promising paradigm is to integrate tool information directly into model parameters and generate tools without retrieval(Schick et al., 2023; Wang et al., 2022; Sun et al., 2023; Kishore et al., 2023; Mehta et al., 2022; Chen et al., 2023). Existing work (Brohan et al., 2023a; Asai et al., 2023; Hao et al., 2023; Wang et al., 2024b) attempts to represent tools as atomic tokens(Geng et al., 2022, 2023; Kang and McAuley, 2018; Sun et al., 2019) and trains with existing token embeddings, so that LLMs can directly output atomic tokens by means of the next token in the generation stage by conditional constraints. However, such atomic tokens are relatively independent, i.e., the semantics cannot be directly transferred to new tools without training, and the space of beam search increases linearly. Therefore, this paper employs structure-aware semantic tokenization to solve this problem, which allows tools with similar semantics to share part of the code sequences (Jin et al., 2023; Liu et al., 2024a; Zheng et al., 2024a), achieving logarithmic growth of additional tokens. On the other hand, learning tools through interactive is also prospective, especially as the traces may contain implicit logic for calling multiple tools. However, existing methods (Parisi et al., 2022; Schick et al., 2023; Nakano et al., 2021) require frequent interaction with unstable environments, resulting in high system design and tuning costs, and the tool or action space involved is small, which is not suitable for large-scale tool invocation scenarios. To this end, this paper considers direct fine-tuning of LLMs using massive trajectory data. Moreover, prior work has not sufficiently investigated the dynamic refinement of semantic code sequences during training (Qu et al., 2024; Wang et al., 2024c), leading to suboptimal performance in downstream tasks, a gap this paper aims to address.

3 Preliminaries

Existing agents based on LLMs that use tools typically involve four stages (Qu et al., 2025): given a query/task $\mathcal{Q}$, (1) generating a plan p, (2) determining the tool $d \in \mathcal{D}$, (3) generating tool parameters c, (4) and collecting feedback f from tool execution. The model iteratively repeats the process (p_i, d_i, c_i, f_i) until it generates a stopping symbol or reaches the maximum number of iterations, ultimately generating the answer $\mathcal{A}$ and completing the task. The entire process forms an interaction trajectory $Traj = [\mathcal{Q}, (p_1, d_1, c_1, f_1), ..., (p_t, d_t, c_t, f_t), \mathcal{A}]$, while t is the total round, and $i \in t$. A specific example can be found in Figure 8.

We unify the four phases through a generative framework and focus on improving the second phase. During the generative tool determination phase, $y_{i+1} = logP(Idx(d)|\mathcal{Q}, y_{<i+1}, embd(\mathcal{D}))$, where $Idx(d)$ is the tool tokens. When the candidate toolset $|\mathcal{D}| = N$ is large, existing unique identifier schemes (Hao et al., 2023; Wang et al., 2024b) suffer from sparse supervision and poor generalization. Instead, if tool representations share substructures, we can reduce representation space and enhance inter-tool correlation. To this end, we adopt a codebook-based semantic tokenization (Van Den Oord et al., 2017), where a codebook with L layers and K codes per layer enables tools to share semantic components. Two similar tools will share the same code at layer $l \in L$. This yields a representation capacity of K^L, allowing compact encoding even when $N \gg K$. Compared to unique identifiers requiring $N \times \mathbf{D}$ memory, our method compresses into logarithmic space $K \times L \times \mathbf{D}$, where $\mathbf{D}$ is the embedding dimension.

4 Proposed Approach: Toolscaler

4.1 Tool Tokenization

Semantic Compression. Following previous works (Mu et al., 2024; Liu et al., 2024b), we organize the encoder input into four distinct blocks: [Content; Token; Placeholder; Task], where [;] denotes concatenation. Specifically:

$$\begin{aligned}\mathsf{Content} &= [\boldsymbol{a}_1; \boldsymbol{a}_2; \ldots; \boldsymbol{a}_r]\\ \mathsf{Token} &= [\mathrm{g}_1, \mathrm{g}_2, \ldots]\\ \mathsf{Placeholder} &= [\mathrm{p}_1, \mathrm{p}_2, \ldots]\\ \mathsf{Task} &= [\mathrm{t}_j; \boldsymbol{a}_j]\end{aligned}$$

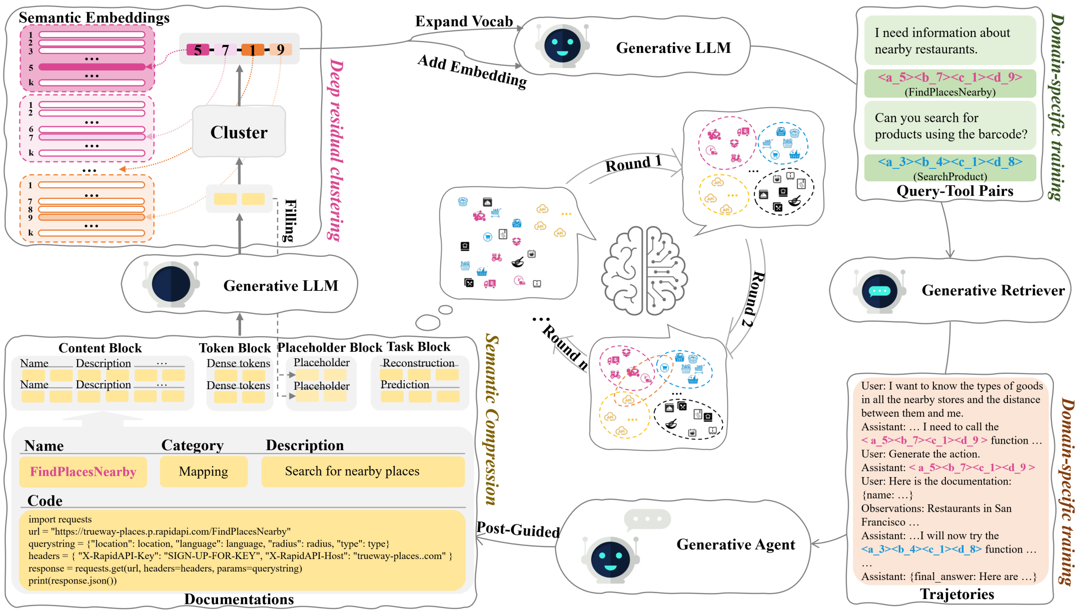


Figure 1: Overview of the Toolscaler framework. Toolscaler employs a three-stage training paradigm with multi-round iterative refinement to progressively optimize tool representations. Initially, the LLM generates clustered, structure-aware semantic code sequences that replace tool names in the corpus, forming a compact and expressive representation space. The model then learns tool usage from Query-Tool pairs, procedural logic from execution trajectories, and collaboration patterns from multi-tool interactions. Throughout iterations, tool knowledge and clustering co-evolve, refining code sequences for better alignment with downstream retrieval and invocation. Finally, a generative agent is trained to perform end-to-end tool calling.

The Content block contains textual information extracted from the tool documentation, such as functional descriptions, and is represented as $\{\boldsymbol{a}_j\}_{j=1}^{m}$, where m denotes the number of distinct pieces of information. The Token block consists of a sequence of V gist tokens (Mu et al., 2024), each with learnable embeddings designed to extract and aggregate information from the Content block. The Task block contains a special indicator token t_j and the corresponding target output $\boldsymbol{a}_j$; specifically, for $0 < j \leq r$ the task is reconstruction, whereas for $r < j \leq m$ it is generation. Finally, the Placeholder block is employed to ensure that the Task block can be effectively guided by the output of the Token block. In practice, its embedding is initialized with the output embedding of the Token block, thereby facilitating the reconstruction or generation process.

Similarly, we adopt a cascaded attention masking scheme to restrict Task output generation solely to the Token (and subsequent Placeholder) blocks. Each block applies a causal mask to capture internal sequential dependencies, while only the Content block fully attends to the Token block and the Placeholder block to the Task block; all other inter-block attention is disabled.

Deep residual clustering. After obtaining the Token block's output embeddings, we cluster these embeddings to derive semantic codes with explicit classification signals. Although we initially explored unsupervised k-means (Krishna and Murty, 1999) – in contrast to training-dependent methods such as RQ-VAE (Lee et al., 2022) – our experiments show that a single-level k-means incurs a high collision rate and yields inaccurate tool partitioning. To address this, we adopt a deep residual clustering approach.

Specifically, let the Token block's output embeddings be:

$$\mathbf{E} = \begin{bmatrix} \boldsymbol{e}_{1,1} & \boldsymbol{e}_{1,2} & \cdots & \boldsymbol{e}_{1,N} \\ \boldsymbol{e}_{2,1} & \boldsymbol{e}_{2,2} & \cdots & \boldsymbol{e}_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{e}_{V,1} & \boldsymbol{e}_{V,2} & \cdots & \boldsymbol{e}_{V,N} \end{bmatrix},$$

where $\boldsymbol{e}_{i,j} \in \mathbb{R}^{\mathbf{D}}$ denotes the i-th gist token embedding for the j-th tool (with $\mathbf{D}$ typically high, e.g., 4098). We first apply principal component analysis (PCA) (Maćkiewicz and Ratajczak, 1993) to reduce each $\boldsymbol{e}_{i,j}$ to a lower-dimensional vector $\widehat{\boldsymbol{e}}_{i,j} \in \mathbb{R}^{\widehat{\mathbf{D}}}$ (e.g., $\widehat{\mathbf{D}} = 32$).

For each gist token position i, let

$$\widehat{\mathbf{E}}[i,:] = [\widehat{\boldsymbol{e}}_{i,1}, \widehat{\boldsymbol{e}}_{i,2}, ..., \widehat{\boldsymbol{e}}_{i,N}]$$

denote the reduced embeddings across the N tools. We then adopt an L-level residual quantization framework by applying k-means clustering on the embedding space $\widehat{\mathbf{E}}$; at each level $l \in \{1, ..., L\}$, a codebook is learned as follows:

$$\mathcal{C}^l = \{\boldsymbol{z}_k^l \in \mathbb{R}^{\widehat{\mathbf{D}}} : k = 1, ..., K\}$$

where K is the number of centroids and $\boldsymbol{z}$ is the vector of centroids. For each reduced embedding $\widehat{\boldsymbol{e}}_{i,j}^l$ at level l, we assign it to its nearest centroid (measured via Euclidean distance) and compute the residual for the next level:

$$\widehat{\boldsymbol{e}}_{i,j}^{l+1} = \widehat{\boldsymbol{e}}_{i,j}^l - \boldsymbol{z}_{k^*}^l,$$

$$\text{with} \quad k^* = \underset{k \in \{1,...,K\}}{\operatorname{argmin}} \left\| \widehat{\boldsymbol{e}}_{i,j}^l - \boldsymbol{z}_k^l \right\|_2.$$

This process yields a sequence of discrete codes for each tool:

$$\boldsymbol{c}_j = [\mathbf{c}_{1,j}^1, \mathbf{c}_{1,j}^2, ..., \mathbf{c}_{1,j}^L, \mathbf{c}_{2,j}^1 ..., \mathbf{c}_{V,j}^L],$$

where each $\mathbf{c}_{i,j}^l \in \{1, ..., K\}$ corresponds to the centroid index assigned at level l for the i-th token.

We generate m augmented copies for each training sample to accommodate different tasks. In addition, we employ low-rank adaptation (LoRA) (Hu et al., 2021) and update only token and task blocks during training. The model is optimized using cross-entropy loss:

$$\mathcal{L}_{tool} = -\sum_{(i,j)} logP(\mathrm{a}_{i,j+1} | \mathrm{a}_{i,1}, \mathrm{a}_{i,2}, ..., \mathrm{a}_{i,j})$$

where $\mathrm{a}_{i,j}$ denotes the j-th token of $\boldsymbol{a}_i$. During inference, the next token is selected as

$$\mathrm{a}_{i,j+1} = \underset{\mathrm{w} \in \mathbf{W}}{\operatorname{argmin}} P(\mathrm{w} | \mathrm{a}_{i,1}, \mathrm{a}_{i,2}, ..., \mathrm{a}_{i,j})$$

, and $\mathbf{W}$ is the token vocabulary.

4.2 Generative Calling

Reframe embedding. In the subsequent stage, to allow the model to generate and invoke tools as next-tokens during interaction, we integrate the learned codes into the language model vocabulary as new tokens. For instance, consider a semantic code sequence of length four, e.g., [154, 53, 48, 1]. We represent it via unique tokens such as [<a_154>, <b_53>, <c_487>, <d_1>]. These tokens are then trained using Query-Tool examples and trajectories.

While explicit tool knowledge is transferred through these codes—multiple retrieval-capable tools may share the code <a_154> —further tool information remains embedded in the Token block's output embeddings. This aspect is often overlooked by previous works, which either fine-tune directly on downstream tasks (Wang et al., 2024b) or apply alignment objectives for refine-tuning (Liu et al., 2024b). We posit that this embedded knowledge can be implicitly transferred through shared network parameters. Hence, we reassign the output embeddings as the initial tool memory for the newly introduced tokens, allowing them to be updated in subsequent training.

However, we cannot directly assign $\boldsymbol{z}_k^l$ to $\mathbf{c}^l$ due to dimensional mismatch (e.g., $32 \neq 4028$). To address this, we aggregate the embeddings for each residual level l of the gist tokens as follows:

$$\mathbf{E}^l = \begin{cases} \mathbf{E}^l, & l = 0, \\ PCA^{-1}(\widehat{\mathbf{E}}^l), & l \geq 1 \end{cases}$$

For $l \geq 1$, we restore the low-dimensional residual vectors via the inverse PCA transform; for $l = 0$, we simply use the original output embeddings. Next, for each code $\mathbf{c}^l$ with centroid index k, its embedding is defined as the average of all tool embeddings assigned to that code:

$$\boldsymbol{e}_{\mathbf{c}^l} = \frac{1}{|\Delta|}(\sum_{\delta \in \Delta} \boldsymbol{e}_\delta^l), \ \Delta = \{\delta | \mathbf{c}^l \in \boldsymbol{c}_j\}, \boldsymbol{e}_\delta^l \in \mathbf{E}^l$$

Ultimately, combining this reframed embedding with the compression process equips the LLM with fundamental tool knowledge and their associated operations.

Domain-specific training. Following Wang et al. (2024b), we implement generative tool calls using two data-organization strategies derived from ToolBench. First, using Query-Tool examples, we train the model to generate the correct code sequences $\boldsymbol{c}_j$ conditioned on a user query q. We fine-tune the LLM's parameters $\boldsymbol{\theta}$ using a next-token prediction loss:

$$\mathcal{L}_{ret} = \sum_{q \in Q} \sum_{i=1}^{V*L} -\log P_{\boldsymbol{\theta}}(\mathbf{c}_j^i | q)$$

Second, we fine-tune the model on trajectories (described in Section A) to enable it to function as an intelligent agent. In this phase, the model learns to determine a solution schema, select appropriate tools, generate input parameters based on tool documentation, and produce a final answer from the tools' execution results. We employ cross entropy based next-token prediction over the assistant's response within each dialogue:

$$\mathcal{L}_{traj} = \sum_{u \in Traj} \sum_{v=1}^{\mathcal{T}_{\mathrm{a}}^u} -logP_{\boldsymbol{\theta}}(\mathrm{a}_v^u | q^u, \mathrm{a}_1^u, ..., \mathrm{a}_{v-1}^{(u)})$$

where q^u is the user query for dialogue u, a_v^u is the v-th token in the assistant's response, and $\mathcal{T}_a^u$ is the total number of tokens in that response. Only the assistant tokens contribute to the loss, enabling the model to jointly learn tool calling and final answer generation.

Post-guided Training. Pre-generated code sequences may be suboptimal for downstream tasks, as trajectory data contain collaboration signals suggesting that functionally similar tools should share similar code sequences, yet these signals remain underexploited. To address this limitation, we propose a post-guided iterative training strategy. In the first round, the standard pipeline produces initial parameters $\boldsymbol{\theta}_0$ for the final LLM. In subsequent rounds $t \in T$, we update the Token embeddings while keeping $\boldsymbol{\theta}_{t-1}$ fixed. At the end of each epoch, a new codebook $\mathcal{C}_t$ is generated to replace the previous one. The trajectory loss $\mathcal{L}_{traj}^t$ is computed using the frozen $\boldsymbol{\theta}_{t-1}$, and the overall fine-tuning loss in round t is given by the sum $\mathcal{L}_{tool}^t + \mathcal{L}_{traj}^t$. After that, the updated code sequences serve as the foundation for the remaining training stages.

Appendix E.2 shows that this multi-round strategy dynamically refines the code sequences and embeddings, yielding code sequences that better support downstream tasks and improve LLM performance.

4.3 Inference

During inference, we employ constrained beam search to ensure that generated tokens correspond to valid code sequences. To this end, we construct a code tree that encompasses all possible code combinations, where each node's children represent the feasible codes that can follow the current code. This tree restricts the search space by effectively blocking infeasible token combinations.

Since the trajectory is divided into several steps (p_i, d_i, c_i, f_i) and the model outputs the tool's code sequence directly in the second step, we apply constrained search only at that step, while standard beam search is used for the other steps.

5 Experiments

5.1 Experimental Setups

Datasets. We evaluate our method on ToolBench (Qin et al., 2023b), a state-of-the-art, large-scale benchmark designed for instruction tuning in tool-use scenarios. ToolBench contains 16,464 real-world RESTful APIs sourced from the RapidAPI Hub[2], each associated with a name, domain category, and a set of API functions. In this work, we treat each API function as a distinct tool, resulting in 46,985 unique and usable tools. For evaluation, we consider three scenarios: **I1** (single-tool queries), **I2** (multi-tool queries within the same category), and **I3** (multi-tool queries within the same collection). Detailed dataset statistics and illustrative examples are provided in Appendix B.

Baselines. We adopt several classical retrieval methods as baselines, including BM25 (Robertson et al., 2009), Embedding Similarity (EmbSim) (Kohane and Zitnik), and ToolRetriever (Qin et al., 2023b), to evaluate the effectiveness of our method in retrieving tools relevant to a given query. In addition, we compare our approach with ToolGen (Wang et al., 2024b), a state-of-the-art generative tool usage model. For tool calling tasks, we benchmark against GPT-4o-mini, ToolGen, and ToolLlama-2 (Qin et al., 2023b). A comprehensive description of all baselines is provided in Appendix C.

Metrics. To evaluate the effectiveness of each retrieval scheme in selecting the appropriate tool for a given query, we employ Normalized Discounted Cumulative Gain (NDCG), a standard metric in information retrieval. We report NDCG@1, NDCG@3, and NDCG@5 to assess ranking quality at varying depths. For tool calling evaluation, we adopt the StableToolBench framework (Guo et al., 2024), which provides two key metrics: Solvable Pass Rate (SoPR), indicating the proportion of successfully completed queries, and Solvable Win Rate (SoWR), measuring the percentage of cases where the candidate model's answer surpasses that of the reference one (GPT-4o-mini based on ground truth).

5.2 Experimental Results

As shown in Table 1, Toolscaler consistently achieves the best performance across all settings, demonstrating strong retrieval accuracy in both simple and complex queries. Compared to ToolGen, Toolscaler demonstrates notable gains(e.g., +4.5 NDCG@1 on **I1**, +6.5 on **I2** and +8 on **I3**), validating the benefit of its tokenization and training strategies. Moreover, on subsets involving unseen tools (**Tool.** and **Cat.**), Toolscaler still maintains top performance, surpassing ToolGen by up to 7 NDCG@1 on **I1-Tool.** and 8.54 on **I2-Cat.**, high-

[2]https://rapidapi.com/hub

Table 1: Multi-domain tool retrieval and evaluation. We train all models on the full ToolBench dataset (**I123**) and evaluate retrieval performance across all tools. BM25 and EmbSim serve as unsupervised baselines, while ToolRetriever and ToolGen are supervised. ToolGen, like our method, is trained via next-token prediction. All results are re-evaluated using publicly released checkpoints. In addition to unseen instruction subsets for **I1**, **I2**, and **I3**, we also assess generalization to unseen tools in **I1** and **I2** (denoted as **Tool.** and **Cat.**).

Model	NDCG@1			NDCG@3			NDCG@5		
	I1	**I2**	**I3**	**I1**	**I2**	**I3**	**I1**	**I2**	**I3**
BM25	26.92	20.00	10.00	26.13	21.92	10.08	29.00	23.46	12.33
EmbSim	50.50	46.00	18.00	48.15	39.58	17.77	53.41	43.05	20.94
ToolRetriever	75.92	63.00	28.00	76.96	66.38	39.28	82.31	72.72	44.54
ToolGen	88.50	84.00	81.00	88.83	85.65	80.83	91.65	89.02	85.83
Toolscaler	**93.00**	**90.50**	**89.00**	**93.87**	**92.26**	**88.16**	**94.85**	**93.68**	**91.98**
	I1-Tool.	**I1-Cat.**	**I2-Cat.**	**I1-Tool.**	**I1-Cat.**	**I2-Cat.**	**I1-Tool.**	**I1-Cat.**	**I2-Cat.**
BM25	20.75	20.63	16.58	21.12	20.67	19.55	23.64	24.18	20.89
EmbSim	53.00	58.00	35.68	49.82	54.38	33.92	54.93	59.24	36.22
ToolRetriever	75.25	73.50	60.30	78.26	73.56	64.11	83.08	79.10	73.01
ToolGen	84.00	89.50	83.42	86.40	89.95	86.06	89.52	92.01	88.47
Toolscaler	**91.00**	**93.00**	**91.96**	**92.20**	**93.56**	**91.06**	**93.89**	**94.92**	**92.97**

Table 2: Task completion evaluation with ground-truth and retrieved tools. We evaluate model performance under two settings: (1) using ground-truth candidate tools, and (2) retrieving candidates from the full toolset. Both GPT and ToolLlama rely on external retrievers. All results are reported as the average of three runs using SoPR and SoWR metrics. Bold indicates the best result under each setting.

Model	Setting	SoPR						SoWR					
		I1	**I2**	**I3**	**I1-Tool.**	**I1-Cat.**	**I2-Cat.**	**I1**	**I2**	**I3**	**I1-Tool.**	**I1-Cat.**	**I2-Cat.**
GPT-4o-mini	GT.	52.66	43.40	**33.06**	**50.11**	49.46	**52.82**	-	-	-	-	-	-
ToolLlama-2	GT.	36.30	17.30	7.92	31.86	39.54	21.24	25.77	20.75	21.31	25.94	35.95	15.32
ToolGen	GT.	47.85	34.91	29.23	35.76	41.29	25.27	38.65	35.85	37.70	25.31	33.33	22.58
Toolscaler	GT.	**60.22**	**44.03**	27.87	44.20	**51.09**	39.65	**39.88**	**43.40**	**40.98**	**37.97**	**47.06**	**31.45**
GPT-4o-mini	Retrieval	52.25	40.41	24.86	**53.16**	50.11	39.38	-	-	-	-	-	-
ToolLlama-2	Retrieval	28.94	24.69	10.93	28.48	36.93	19.09	25.15	30.19	24.59	26.58	27.45	20.16
ToolGen		52.97	45.13	36.34	45.36	55.56	45.56	36.20	42.45	**49.18**	32.91	42.48	37.90
Toolscaler		**62.78**	**52.04**	**41.26**	52.53	**57.19**	**56.99**	**42.94**	**46.23**	45.90	**42.41**	**47.71**	**37.90**

lighting its strong compositional generalization and scalability to previously unseen tools.

Table 2 presents the task execution success rates under two settings: (1) **GT.**: where the ground-truth tool is provided in the query prompt, and (2) **Retrieval**: where tools are retrieved from the entire toolset without prior hints. Toolscaler and ToolGen, which both directly integrate tool retrieval into the generation process, consistently outperform retriever-dependent models (GPT-4o-mini, ToolLlama-2) in SoPR, demonstrating superior tool selection and end-to-end reasoning capabilities. Of course, Toolscaler clearly demonstrates superior performance. Interestingly, generative models such as Toolscaler and ToolGen perform better in the **Retrieval** setting than in the **GT.** setting. We hypothesize this counterintuitive result stems from potential interaction mismatch introduced by supervised fine-tuning (SFT) when ground-truth tools are forcefully injected, which may reduce robustness. We leave a detailed investigation of this phenomenon for future work. Regarding SoWR, Toolscaler also outperforms most generative baselines, confirming its ability to produce high-quality outputs. Despite Toolscaler achieving notably higher SoPR than the reference model (GPT-4o-mini **GT.**), its SoWR remains below 50%. This suggests a systemic gap between model predictions and GPT-4o-mini's internal satisfaction criteria, raising broader questions about evaluation alignment.

5.3 Further Analysis

Ablaiton Study. To evaluate the contribution of key components in our method, we conduct ablation experiments and present the results in Figure 2. The results show that removing either component leads to consistent performance degradation across all settings (**I1**, **I2**, **I3**). Notably, eliminating post-guided training causes significant drops in NDCG@1, especially in **I2** and **I3**, where the

Table 3: Retrieval performance of different tokenization methods in the Multi-domain setting. All models are trained on Query-Tool pairs and Trajectories. The results of ToolGen are directly adopted as the baseline for the Atomic.

Tokenization	NDCG@1			NDCG@3			NDCG@5		
	I1	I2	I3	I1	I2	I3	I1	I2	I3
Numerical	82.00	77.50	81.91	84.18	77.53	76.51	70.00	88.07	84.30
Hierarchical	87.50	77.50	79.00	86.11	78.82	81.44	89.91	83.81	87.47
Semantic	90.00	84.50	84.00	91.56	84.33	79.41	92.96	88.44	87.40
Atomic	88.50	84.00	81.00	88.83	85.65	80.83	91.65	89.02	85.83
Toolscaler	**93.00**	**90.50**	**89.00**	**93.87**	**92.26**	**88.16**	**94.85**	**93.68**	**91.98**

Table 4: Tool calling evaluation for different tokenization methods. Bold values denote the highest performance.

Tokenization	SoPR						SoWR					
	I1	I2	I3	I1-Tool.	I1-Cat.	I2-Cat.	I1	I2	I3	I1-Tool.	I1-Cat.	I2-Cat.
Numerical	21.98	9.12	11.20	20.68	26.14	17.20	16.56	16.04	16.39	20.89	23.53	14.52
Hierarchical	39.16	20.28	17.49	36.29	31.81	14.92	29.45	28.30	26.23	29.11	24.83	14.52
Semantic	50.20	29.72	16.39	33.02	51.42	27.02	39.26	29.24	32.79	29.11	43.79	22.58
Atomic	52.97	45.13	36.34	45.36	55.56	45.56	36.20	42.45	**49.18**	32.91	42.48	37.90
Toolscaler	**62.78**	**52.04**	**41.26**	**52.53**	**57.19**	**56.99**	**42.94**	**46.23**	45.90	**42.41**	**47.71**	**37.90**

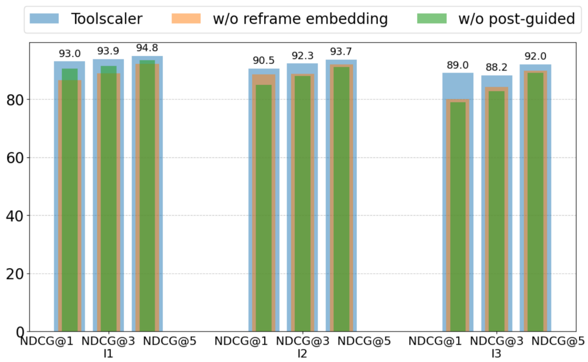


Figure 2: Ablation study for tool retrieval in the Multi-domain setting. We evaluate the impact of removing embedding initialization for code tokens and omitting the second training iteration on Toolscaler's performance.

performance drops by 5 and 10 points, respectively. These results highlight the importance of semantic transfer from language modeling and the effectiveness of iterative guidance in enhancing tool discrimination and retrieval accuracy in complex compositions.

Tokenization Strategy Evaluation. We compare several tokenization strategies for tool retrieval. Our structure-aware semantic tokenization, while conceptually related to Hierarchical and Semantic tokenizations, goes further by constructing code sequences across both feature dimensions and residual depth, and dynamically refining them using richer semantic and interaction signals. As reported in Table 3 and Table 4, our method consistently outperforms existing strategies, achieving stronger tool ranking (NDCG) and downstream invocation accuracy (SoPR/SoWR), especially on more ambiguous cases like **I2** and **I3**.

This performance gain stems from Toolscaler's ability to preserve interaction-aware structure during tokenization. While other strategies often rely on fixed hierarchies or shallow semantics, Toolscaler dynamically groups semantically related tool actions and constructs trajectory-aligned code sequences, reducing information loss across modalities. This leads not only to higher relevance ranking, but also to clearer contextual grounding for accurate tool calling. For instance, improvements in NDCG@3/5 translate into SoPR/SoWR gains across both **I1/I2/I3** and categorically split settings, reflecting the method's generalizability and real-world robustness.

More results and implementation details can be found in Appendix E and Appendix C.

6 Conclusions

We propose Toolscaler, a model-agnostic framework that leverages a single LLM to perform generative tool retrieval and calling, thereby eliminating the need for external retrievers. Toolscaler introduces structure-aware semantic code sequences to concisely and effectively represent large-scale toolsets, while maintaining adaptability to the continual expansion of new tools. Our method integrates basic tool knowledge and inter-tool coordination signals, and dynamically refines code sequences through multistage iterative training. Extensive experiments demonstrate the effectiveness of Toolscaler, particularly in multi-tool scenarios. It consistently outperforms strong baselines in accuracy, pass rate, and win rate, and further shows clear advantages over other tokenization strategies through structure-aware semantic modeling. Our study provides a promising direction for large-scale generative tool execution and lays the groundwork for future extensions, such as combining generative agents with reinforcement learning to further enhance tool-use autonomy in LLMs.

Limitations

While our structure-aware semantic tokenization method demonstrates strong scalability and generalization in large-scale tool scenarios, its performance still relies on the initial quality of tool documentation and the stability of the clustering process. Specifically, when tool descriptions are sparse, ambiguous, or inconsistent across domains, the generated semantic identifiers may not fully capture functional nuances, potentially affecting downstream retrieval or planning accuracy. Similarly, due to the token-augmentation framework, Toolscaler requires a certain degree of training or fine-tuning to match the potential space of the underlying language model with the newly introduced semantic labels. This dependence weakens its adaptability in cold-start or zero-sample scenarios. Moreover, the current framework lacks the ability to seamlessly remove invalid or outdated tools, which may hinder the long-term maintenance of the dynamic tool ecosystem. Furthermore, although our current iterative training pipeline, though effective, involves non-negligible computational overhead, which may limit applicability in rapid deployment scenarios.

Ethical Considerations

We recognize the ethical considerations in developing large language models and have carefully used publicly available pretrained LLMs (e.g., Llama-2-7B, Llama-3-8B) and the ToolBench dataset. The ToolBench dataset is licensed under Apache 2.0, which permits free use and modification. Our use fully complies with its license terms and intended purposes. The data contain no sensitive personal information, and all ethical guidelines are observed in processing these resources.

Acknowledgements

This work is jointly sponsored by National Natural Science Foundation of China (62141608, 62236010, 62576339), Beijing Natural Science Foundation (L252033), and CAAI-Ant Group Research Fund.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. 2023a. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*.

Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. 2023b. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on robot learning*, pages 287–318. PMLR.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Jiangui Chen, Ruqing Zhang, Jiafeng Guo, Maarten de Rijke, Wei Chen, Yixing Fan, and Xueqi Cheng. 2023. Continual learning for generative retrieval over dynamic corpora. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 306–315.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammadhossein Bateni, Chen-Yu Lee, and Tomas Pfister. 2024. Re-invoke: Tool invocation rewriting for zero-shot tool retrieval. *arXiv preprint arXiv:2408.01875*.

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359.

Difei Gao, Lei Ji, Luowei Zhou, Kevin Qinghong Lin, Joya Chen, Zihan Fan, and Mike Zheng Shou. 2023a. Assistgpt: A general multi-modal assistant that can plan, execute, inspect, and learn. *arXiv preprint arXiv:2306.08640*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023b. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. 2022. Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5). In *Proceedings of the 16th ACM Conference on Recommender Systems*, pages 299–315.

Shijie Geng, Juntao Tan, Shuchang Liu, Zuohui Fu, and Yongfeng Zhang. 2023. Vip5: Towards multimodal foundation models for recommendation. *arXiv preprint arXiv:2305.14302*.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. *arXiv preprint arXiv:2403.07714*.

Tanmay Gupta and Aniruddha Kembhavi. 2023. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36:45870–45894.

Joy He-Yueya, Gabriel Poesia, Rose E Wang, and Noah D Goodman. 2023. Solving math word problems by combining language models with symbolic solvers. *arXiv preprint arXiv:2304.09102*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022a. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR.

Wenlong Huang, Fei Xia, Dhruv Shah, Danny Driess, Andy Zeng, Yao Lu, Pete Florence, Igor Mordatch, Sergey Levine, Karol Hausman, et al. 2023. Grounded decoding: Guiding text generation with grounded models for robot control. *arXiv preprint arXiv:2303.00855*.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2022b. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Bowen Jin, Hansi Zeng, Guoyin Wang, Xiusi Chen, Tianxin Wei, Ruirui Li, Zhengyang Wang, Zheng Li, Yang Li, Hanqing Lu, et al. 2023. Language models as semantic indexers. *arXiv preprint arXiv:2310.07815*.

Qiao Jin, Yifan Yang, Qingyu Chen, and Zhiyong Lu. 2024. Genegpt: Augmenting large language models with domain tools for improved access to biomedical information. *Bioinformatics*, 40(2):btae075.

Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*, pages 197–206. IEEE.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Varsha Kishore, Chao Wan, Justin Lovelace, Yoav Artzi, and Kilian Q Weinberger. 2023. Incdsi: incrementally updatable document retrieval. In *International Conference on Machine Learning*, pages 17122–17134. PMLR.

Isaac S Kohane and Marinka Zitnik. Deep learning for diagnosing patients with rare genetic diseases.

K Krishna and M Narasimha Murty. 1999. Genetic k-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 29(3):433–439.

Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. 2022. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11523–11532.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023a. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Qijiong Liu, Hengchang Hu, Jiahao Wu, Jieming Zhu, Min-Yen Kan, and Xiao-Ming Wu. 2024a. Discrete semantic tokenization for deep ctr prediction. In *Companion Proceedings of the ACM on Web Conference 2024*, pages 919–922.

Qijiong Liu, Jieming Zhu, Lu Fan, Zhou Zhao, and Xiao-Ming Wu. 2024b. Store: Streamlining semantic tokenization and generative recommendation with a single llm. *arXiv preprint arXiv:2409.07276*.

Qijiong Liu, Jieming Zhu, Yanting Yang, Quanyu Dai, Zhaocheng Du, Xiao-Ming Wu, Zhou Zhao, Rui Zhang, and Zhenhua Dong. 2024c. Multimodal pretraining, adaptation, and generation for recommendation: A survey. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6566–6576.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023b. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Yulong Liu, Yunlong Yuan, Chunwei Wang, Jianhua Han, Yongqiang Ma, Li Zhang, Nanning Zheng, and Hang Xu. 2024d. From summary to action: Enhancing large language models for complex tasks with open world apis. *arXiv preprint arXiv:2402.18157*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems*, 36:43447–43478.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning. *arXiv preprint arXiv:2301.13379*.

Andrzej Maćkiewicz and Waldemar Ratajczak. 1993. Principal components analysis (pca). *Computers & Geosciences*, 19(3):303–342.

Sanket Vaibhav Mehta, Jai Gupta, Yi Tay, Mostafa Dehghani, Vinh Q Tran, Jinfeng Rao, Marc Najork, Emma Strubell, and Donald Metzler. 2022. Dsi++: Updating transformer memory with new documents. *arXiv preprint arXiv:2212.09744*.

Dheeraj Mekala, Jason Weston, Jack Lanchantin, Roberta Raileanu, Maria Lomeli, Jingbo Shang, and Jane Dwivedi-Yu. 2024. Toolverifier: Generalization to new tools via self-verification. *arXiv preprint arXiv:2402.14158*.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*.

Jesse Mu, Xiang Li, and Noah Goodman. 2024. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. 2023. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*.

Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2025. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8494–8502.

Cheng Qian, Chi Han, Yi R Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. 2023. Creator: Tool creation for disentangling abstract and concrete reasoning of large language models. *arXiv preprint arXiv:2305.14318*.

Yujia Qin, Zihan Cai, Dian Jin, Lan Yan, Shihao Liang, Kunlun Zhu, Yankai Lin, Xu Han, Ning Ding, Huadong Wang, et al. 2023a. Webcpm: Interactive web search for chinese long-form question answering. *arXiv preprint arXiv:2305.06849*.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. 2024. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023b. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

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

Haohao Qu, Wenqi Fan, Zihuai Zhao, and Qing Li. 2024. Tokenrec: Learning to tokenize id for llm-based generative recommendation. *arXiv preprint arXiv:2406.10450*.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE.

Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Tran, Jonah Samost, et al. 2024. Recommender systems with generative retrieval. *Advances in Neural Information Processing Systems*, 36.

Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

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.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. 2021. Retrieval augmentation reduces hallucination in conversation. *arXiv preprint arXiv:2104.07567*.

Anima Singh, Trung Vu, Nikhil Mehta, Raghunandan Keshavan, Maheswaran Sathiamoorthy, Yilin Zheng, Lichan Hong, Lukasz Heldt, Li Wei, Devansh Tandon, et al. 2024. Better generalization with semantic ids: A case study in ranking for recommendations. In *Proceedings of the 18th ACM Conference on Recommender Systems*, pages 1039–1044.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE.

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

Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1441–1450.

Weiwei Sun, Lingyong Yan, Zheng Chen, Shuaiqiang Wang, Haichao Zhu, Pengjie Ren, Zhumin Chen, Dawei Yin, Maarten Rijke, and Zhaochun Ren. 2023. Learning to tokenize for generative retrieval. *Advances in Neural Information Processing Systems*, 36:46345–46361.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*.

Aaron Van Den Oord, Oriol Vinyals, et al. 2017. Neural discrete representation learning. *Advances in neural information processing systems*, 30.

Sai H Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. 2024. Chatgpt for robotics: Design principles and model abilities. *IEEE Access*.

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024a. Llms in the imaginarium: tool learning through simulated trial and error. *arXiv preprint arXiv:2403.04746*.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. 2024b. Toolgen: Unified tool retrieval and calling via generation. *arXiv preprint arXiv:2410.03439*.

Wenjie Wang, Honghui Bao, Xinyu Lin, Jizhi Zhang, Yongqi Li, Fuli Feng, See-Kiong Ng, and Tat-Seng Chua. 2024c. Learnable tokenizer for llm-based generative recommendation. *arXiv preprint arXiv:2405.07314*.

Ye Wang, Jiahao Xun, Minjie Hong, Jieming Zhu, Tao Jin, Wang Lin, Haoyuan Li, Linjun Li, Yan Xia, Zhou Zhao, et al. 2024d. Eager: Two-stream generative recommender with behavior-semantic collaboration. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3245–3254.

Yujing Wang, Yingyan Hou, Haonan Wang, Ziming Miao, Shibin Wu, Qi Chen, Yuqing Xia, Chengmin Chi, Guoshuai Zhao, Zheng Liu, et al. 2022. A neural corpus indexer for document retrieval. *Advances in*

Neural Information Processing Systems, 35:25600–25614.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023a. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023b. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. 2023. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*.

Qiancheng Xu, Yongqi Li, Heming Xia, and Wenjie Li. 2024. Enhancing tool retrieval with iterative feedback from large language models. *arXiv preprint arXiv:2406.17465*.

Linyao Yang, Hongyang Chen, Zhao Li, Xiao Ding, and Xindong Wu. 2023a. Chatgpt is not enough: Enhancing large language models with knowledge graphs for fact-aware language modeling. *arXiv preprint arXiv:2306.11489*.

Sherry Yang, Ofir Nachum, Yilun Du, Jason Wei, Pieter Abbeel, and Dale Schuurmans. 2023b. Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Yining Ye, Xin Cong, Yujia Qin, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2023. Large language model as autonomous decision maker. *arXiv preprint arXiv:2308.12519*.

Zheng Yuan, Fajie Yuan, Yu Song, Youhua Li, Junchen Fu, Fei Yang, Yunzhu Pan, and Yongxin Ni. 2023. Where to go next for recommender systems? id-vs. modality-based recommender models revisited. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2639–2649.

Bowen Zheng, Yupeng Hou, Hongyu Lu, Yu Chen, Wayne Xin Zhao, Ming Chen, and Ji-Rong Wen. 2024a. Adapting large language models by integrating collaborative semantics for recommendation. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 1435–1448. IEEE.

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

Yuanhang Zheng, Peng Li, Ming Yan, Ji Zhang, Fei Huang, and Yang Liu. 2024c. Budget-constrained tool learning with planning. *arXiv preprint arXiv:2402.15960*.

Jieming Zhu, Mengqun Jin, Qijiong Liu, Zexuan Qiu, Zhenhua Dong, and Xiu Li. 2024. Cost: Contrastive quantization based semantic tokenization for generative recommendation. In *Proceedings of the 18th ACM Conference on Recommender Systems*, pages 969–974.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36:50117–50143.

A Introduce New Tools After Training

After completing the model training in Section 4, one can follow the process outlined in the Algorithm 1 to invoke the newly added tools without the need for re-training or rebuilding the codebook. Specifically, during semantic compression in Section 4.1, the model learns to encode tool documentation into Token Block embeddings, producing a fixed checkpoint for inference. Importantly, we save the multi-layer residual centroids and PCA from *Deep Residual Clustering* step, which form the "codebook" mapping embeddings to code sequences. For new tools, we:

1. Encode their documentation with the frozen model to get Token Block embeddings;
2. Project embeddings using the saved PCA;
3. Quantize sequentially via saved centroids to generate code sequences compatible with the existing space.

This procedure allows us to embed new tools into the existing semantic code space based on the original codebook.

In order to verify that the codebook has effectively aggregated the information of the tool, we conducted a small-scale experiment. Specifically, we used GPT-4o mini to generate several new tools in the same domain and with similar functionality to an existing tool, "PragmavantApi_web_snapshot" (whose code sequence is [282, 123, 144, 204]). One such generated tool is "PragmavantApi_web_render". Using the saved PCA transformation and clustering centroids, we encoded these new tools via the same residual clustering process and obtained the code sequence [282,

Algorithm 1: Introducing New Tools

Input: Pretrained model M, codebook $C = \{PCA_i, Centers_i^d\}$, new tools T_{new}
Output: Codes Z_{new} for T_{new}
Step 1: Export Gist Embeddings
Replace toolset T in training data with T_{new};
Run M in inference mode to export gist embeddings E_{new} for each $t \in T_{new}$;
Step 2: Load Codebook Components
Load PCA projections $\{PCA_i\}_{i=1}^{N}$ and residual cluster centers $\{Centers_i^d\}_{d=1}^{D}$ from C;
Step 3: Residual Clustering Inference
for $i \leftarrow 1$ **to** N **do**
    foreach $t \in T_{new}$ **do**
        $v_0 \leftarrow PCA_i$.transform$(E_{new}[t][i])$;
        for $d \leftarrow 1$ **to** D **do**
            $label \leftarrow \arg\min_k \|v_{d-1} - Centers_i^d[k]\|$;
            $Z_{new}[t].append(label)$;
            $v_d \leftarrow v_{d-1} - Centers_i^d[label]$;
return Z_{new};

184, 30, 262].The result shows that our framework can capture semantic and functional continuity, enabling structural sharing between existing and newly introduced tools.

According to Section , theoretically, the codebook can represent up to K^L tools. When $K = 512$ and $L = 4$, mathematically, it allows for over 68 billion unique code sequences. This exponential encoding capacity means that even thousands or tens of thousands of tools can be efficiently represented with a relatively small token vocabulary, far more compact than traditional text-based descriptions. Of course, if the size of the toolset is further expanded beyond the range that the current codebook can represent, Toolscaler can expand the codebook by adding more centroids K (i.e., code token vocabulary size) or increasing the length of the code sequences L (e.g., from 4 to 6 tokens), thus further enlarging the representational space. However, this would also require additional re-training.

B Dataset

The ToolBench dataset, introduced by Qin et al. (2023b), was automatically constructed using ChatGPT and supports both single-tool and multi-tool usage scenarios. It involves the generation of instructions and tool call sequences for RESTful APIs. The dataset comprises 16,464 real-world RESTful APIs spanning 49 categories, such as social media, e-commerce, and weather services, totaling 46,985 unique API functions. In our paper, **each API function is treated as an individual tool** and represented using a semantic code sequence. Figure 5 illustrates a real RESTful API example, where each entry in the api_list corresponds to a single API function. In our experiments, we utilize the following fields: "$tool\_name$" and "$name$", "$description$", "$categories$", and "$code$".

Following the setup of Qin et al. (2023b), we construct our training and evaluation sets based on three subsets: **I1** for single-tool queries, **I2** with multi-tool queries from the same category, and **I3** with multi-tool queries from the same collection. **I2** and **I3** are created by randomly selecting 2–5 RESTful APIs from the same category or collection in RapidAPI and sampling up to 3 API functions per RESTful API to form each instruction sample. It should be noted that these independent API services have different interfaces and purposes. For example, for a user query: "*I need to gather some information about Ethereum for my company's financial report. Please fetch the last mined Ethereum block number and provide me with the ticker data for ETHUSD. It would be great if you could also give me the historical rates of currency conversion from USD to ETH.*" Its trajectory involves at least three API services:

- *getLastBlockNumber* from tool named *Chaingateway.io*: Returns the block number of the last mined ethereum block;
- *TickerPerSymbol* from tool named *Global*

Table 5: Dataset statistics for the three-stage training.

Dataset	Tool Tokenization	Retrieval				Tool Calling
		I1	I2	I3	All	
Train	49,936	194,086	222,783	72,833	489,702	183,336

Ethereum Price Index (GEX): Returns ticker data for specified symbol;

- *Timeframe* from tool named *CurrencyAPI.net*: Displays a currency conversion's historical rates within a given timeframe.

These API services fall under the same top-level "Finance" category in RapidAPI, but they come from entirely different tool providers, exhibit distinct semantics and calling structures, and collectively address a cross-functional financial analysis task. Such examples are not isolated—we identified 80,847 trajectories in ToolBench with similarly diverse tool compositions, which indicates that ToolBench can provide meaningful coverage of generalization challenges.

Anyway, the resulting subsets contain 87,413 (**I1**), 84,815 (**I2**), and 25,251 (**I3**) Query-Tool pairs, respectively.

In the semantic compression stage, tool documentation serves as input, and the objective is to reconstruct individual fields such as category, tool name, code, and description. Consequently, each tool document generates $m = 4$ training instances, one for each reconstruction target.

For the domain-specific training stage, we extract Query-Tool pairs from ToolBench, using the query as input and the semantic code sequence of the relevant tools as output. Figure 6 shows an example training instance. Further, for the tool calling stage, we follow the procedure from Wang et al. (2024b), removing the system prompt tool descriptions and adopting a three-stage output format (Thought, Action, Action Input). We replace tool names in the trajectories with our code sequences and construct a mapping dictionary between tool names and their corresponding codes to enable document lookup during execution. Figure 7 presents an illustrative training example.

Finally, Table 5 summarizes the data scale across each training phase.

C Baselines and Tokenization Methods

Baseline Models. In the tool retrieval comparison experiment, we adopted the following representative retrieval models as the baseline for comparison with Toolscaler:

- BM25: An unsupervised retrieval model that ranks documents by query relevance, using normalized term frequency and document length.
- Embedding Similarity (EmbSim): utilizes sentence embeddings generated by OpenAI's text-embedding-3-large model to compute semantic similarity between queries and tool documents.
- ToolRetriever (Qin et al., 2023b): A BERT-based retriever trained using contrastive learning to distinguish between relevant and irrelevant tools by maximizing the similarity between queries and corresponding tools.
- ToolGen (Wang et al., 2024b): A unified framework that integrates tool retrieval and calling within large language models by representing each tool as an atomic token, enabling the model to generate tool calls and arguments directly.

In Appendix E, under the In-domain setting, we also made a comparison with Re-Invoke and Iter-Feedback:

- Re-Invoke (Chen et al., 2024): An unsupervised retrieval method that generates synthetic queries to enrich tool documents and employs large language models to extract user intent during inference, using a multi-view similarity ranking strategy to identify relevant tools.
- IterFeedback (Xu et al., 2024): A retrieval method that incorporates iterative feedback from large language models, using a BERT-based retriever and prompting a language model like gpt-3.5-turbo-0125 to refine retrieval over multiple rounds.

In the tool calling comparison experiment, apart from the comparison with ToolGen, we further evaluate Toolscaler against the following baselines:

- GPT-4o-mini: We employ gpt-4o-mini-2024-07-18 as a baseline, a cost-effective model introduced by OpenAI, utilizing its tool-calling capabilities to form a tool agent.

- ToolLlama-2 (Qin et al., 2023b): Developed by fine-tuning the Llama-2 model on the ToolBench dataset, enhancing its ability to interact with external tools. In this article, we use checkpoint which was open-sourced by Wang et al. (2024b).

- ToolLlama (Wang et al., 2024b): Fine-tuned the Llama-3 model on the ToolBench dataset by Wang et al. (2024b). However, since they did not open source checkpoints, we directly used the data in their paper.

Tokenization Methods. In 5.3, we compared structure-aware semantic with four tokenization methods:

- Numerical: Use a unique numeric string to represent a tool. For example, if the toolkit contains 47,000 tools, then use a five-digit string to represent them, and the 3rd tool is represented as 0 0 0 0 3.

- Hierarchical: Use a unique number to represent a tool and at the same time use clustering to integrate all the numbers in the toolkit into a hierarchical tree. We continue to use the hierarchical coding of Wang et al. (2024b), like 1 0 1 4 0.

- Semantic: Represent a tool using one or more semantic tokens, for example, directly using the names of the API functions, for instance, $compress\_for\_imagon$.

- Atomic: Each tool is represented by a single unique token. ToolGen encodes this as the combined string $<< tool\_name\&\&api\_name >>$ as a token. For instance, the API function *compress* from the RESTful API *IMAGON* is tokenized as $<< IMAGON\&\&compress >>$.

D Experimental Setups

Settings. As proposed by ToolGen and others, we adopt two evaluation settings: In-domain and Multi-domain. In Appendix E, we provide a comprehensive evaluation under both settings, while in the main paper, we report only the results under the Multi-domain setting. In the In-domain scenario, models are restricted to retrieving and reasoning over tools within the same domain (**I1**, **I2**, and **I3**), whereas the Multi-domain setup requires operating over the full toolset, making it considerably more challenging.

For the tool calling experiments, we evaluate two configurations: with Ground Truth Tools (**GT.**) and with Retriever. These two settings are motivated by the fact that methods like ChatGPT and ToolLlama require an explicit list of candidate tools to be included in the prompt. Therefore, the choice between ground truth tools and tools selected by a retriever significantly impacts performance. Following ToolGen, we treat the tools provided by ChatGPT as the Ground Truth Tools for a given query, and we employ a unified retriever (ToolRetriever) for the Retriever-based setting. For ToolLlama, candidate tools are directly included in the prompt. For ToolGen and our proposed Toolscaler, in the **GT.** setting, we constrain the candidate tool space during the planning phase via a prefix prompt. In the **Retriever** setting, we rely entirely on generation without using any external retriever module.

Implementation Details. **i)** In the first training iteration, we start from a pre-trained Llama-3-8B model to learn tool knowledge representations. We optimize using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of $1e^{-3}$, weight decay of $1e^{-4}$, batch size of 12, and LoRA configurations set to rank 32, alpha 128, dropout 0.1. The token block size is set to 2. After obtaining the output embeddings from the token block, we apply PCA (Maćkiewicz and Ratajczak, 1993) to reduce the dimensionality to 32. We then cluster each position into 512 clusters using a two-level residual quantization scheme. The resulting coding sequence length is 4. **ii)** Next, we replace all tool mentions in the ToolBench training text with their semantic code sequences, and we expand the vocabulary of Llama-3-8B by adding 2,048 new tokens (512×4). These new tokens are initialized following the method described in Section 4.1. **iii)** Based on this extended model, we train it on two tasks: Query-Tool pairs and Trajectories. We employ a cosine learning rate scheduler with a 3% warm-up ratio and a maximum learning rate of 4×10^{-5}. For trajectory inputs, the context length is truncated to 6144 tokens. The total batch size is set to 1×64, where 64 denotes the number of gradient accumulation steps. **iv)** After completing the above steps,

Table 6: Tool retrieval evaluation across two settings: In-domain and Multi-domain. * represents the results disclosed in Wang et al. (2024b), while the others are the results we re-implemented based on the open-source checkpoints.

Model	I1			I2			I3		
	NDCG@1	NDCG@3	NDCG@5	NDCG@1	NDCG@3	NDCG@5	NDCG@1	NDCG@3	NDCG@5
In-domain									
BM25*	29.46	31.12	33.27	24.13	25.29	27.65	32.00	25.88	29.78
EmbSim*	63.67	61.03	65.37	49.11	42.27	46.56	53.00	46.40	52.73
Re-Invoke*	69.47	-	61.10	54.56	-	53.79	59.65	-	59.55
IterFeedback*	90.70	90.95	92.47	89.01	85.46	87.10	91.74	87.94	90.20
ToolRetriever*	80.50	79.55	84.39	71.18	64.81	70.35	70.00	60.44	64.70
ToolGen*	89.17	90.85	92.67	91.45	88.79	91.13	87.00	85.59	90.16
BM25	29.25	31.04	33.49	26.50	25.97	27.96	32.00	25.88	29.78
EmbSim	61.00	57.78	62.31	54.00	45.31	49.54	54.00	46.56	52.91
ToolRetriever	83.50	83.67	88.66	72.00	73.27	80.40	70.00	70.01	77.21
ToolGen	91.00	92.15	94.11	87.50	88.52	90.81	87.00	85.35	90.08
Toolscaler	**94.50**	**95.13**	**96.44**	**93.50**	**93.20**	**94.88**	**89.00**	**88.98**	**92.46**
Multi-domain									
BM25*	22.77	22.64	25.61	18.29	20.74	22.18	10.00	10.08	12.33
EmbSim*	54.00	50.82	55.86	40.84	36.67	39.55	18.00	17.77	20.70
ToolRetriever*	72.31	70.30	74.99	64.54	57.91	63.61	52.00	39.89	42.92
ToolGen*	87.67	88.84	91.54	83.46	86.24	88.84	79.00	79.80	84.79
BM25	26.92	26.13	29.00	20.00	21.92	23.46	10.00	10.08	12.33
EmbSim	50.50	48.15	53.41	46.00	39.58	43.05	18.00	17.77	20.94
ToolRetriever	75.92	76.96	82.31	63.00	66.38	72.72	28.00	39.28	44.54
ToolGen	88.50	88.83	91.65	84.00	85.65	89.02	81.00	80.83	85.83
Toolscaler	**93.00**	**93.87**	**94.85**	**90.50**	**92.26**	**93.68**	**89.00**	**88.16**	**91.98**

Table 7: Tool retrieval evaluation under In-domain and Multi-domain settings, including results on **I1-Tool.**, **I1-Cat.**, and **I2-Cat.** subsets.

Model	I1-Tool.			I1-Cat.			I2-Cat.		
	NDCG@1	NDCG@3	NDCG@5	NDCG@1	NDCG@3	NDCG@5	NDCG@1	NDCG@3	NDCG@5
In-domain									
BM25	28.00	31.37	33.06	31.12	30.87	33.13	21.75	24.75	27.44
EmbSim	61.50	58.74	62.99	69.00	66.43	71.00	44.22	39.18	43.50
ToolRetriever	79.50	81.54	86.78	80.50	81.68	87.15	70.35	74.09	81.45
ToolGen	**89.50**	**91.61**	**93.34**	87.50	88.79	91.21	88.44	88.85	91.34
Toolscaler	88.50	91.60	93.24	**95.00**	**95.78**	**96.43**	**92.96**	**92.98**	**93.99**
Multi-domain									
BM25	20.75	21.12	23.64	20.63	20.67	24.18	16.58	19.55	20.89
EmbSim	53.00	49.82	54.93	58.00	54.38	59.24	35.68	33.92	36.22
ToolRetriever	75.25	78.26	83.08	73.50	73.56	79.10	60.30	64.11	73.01
ToolGen	84.00	86.40	89.52	89.50	89.95	92.01	83.42	86.06	88.47
Toolscaler	**91.00**	**92.20**	**93.89**	**93.00**	**93.56**	**94.92**	**91.96**	**91.06**	**92.97**

we treat the resulting model as the base model for the second iteration and repeat steps i, ii, and iii. In our early-stage experiments, we observed that two rounds of iterative training are sufficient to achieve strong performance on ToolBench. Adding more rounds (e.g., 3 or 4) yielded diminishing returns, with little or no measurable improvement in downstream retrieval or execution metrics.

In terms of computation resources, step i is trained on a single A100 GPU, while steps ii and iii require 4×A100 GPUs. We leverage Deepspeed ZeRO-3 (Rajbhandari et al., 2020) and FlashAttention (Dao et al., 2022; Dao, 2023) to optimize training efficiency. We conduct two full training iterations. Each iteration includes 5 epochs of tool retrieval training and 2 epochs of tool calling training. For the tool representation learning phase, we employ an early stopping mechanism, with an average of 6 epochs per run.

E Comprehensive Results

E.1 Main experiments

Tables 6 and 7 provide a more comprehensive evaluation of the tool retrieval stage. Beyond the results presented in the main text, we include experiments under both In-domain and Multi-domain settings, and compare our reproduced results with those reported by Wang et al. (2024b). The close match between our results and theirs indicates that our data preparation and experimental configurations

Table 8: Tool calling evaluation performance on unseen instructions and unseen tools under two settings. Bold values denote the highest performance, considering only the results reproduced in our experimental setting.

Model	Setting	SoPR						SoWR					
		I1	I2	I3	I1-Tool.	I1-Cat.	I2-Cat.	I1	I2	I3	I1-Tool.	I1-Cat.	I2-Cat.
GPT-3.5*	GT.	56.60	47.80	54.64	58.90	60.70	54.60	-	-	-	-	-	-
ToolLlama-2*	GT.	53.37	41.98	46.45	-	-	-	47.27	59.43	27.87	-	-	-
ToolLlama*	GT.	55.93	48.27	52.19	57.38	58.61	56.85	50.31	53.77	31.15	43.04	50.31	54.84
ToolGen*	GT.	61.35	49.53	43.17	52.32	40.46	39.65	51.53	57.55	31.15	39.24	38.56	40.32
GPT-4o-mini	GT.	52.66	43.40	**33.06**	**50.11**	49.46	**52.82**	-	-	-	-	-	-
ToolLlama-2	GT.	36.30	17.30	7.92	31.86	39.54	21.24	25.77	20.75	21.31	25.94	35.95	15.32
ToolGen	GT.	47.85	34.91	29.23	35.76	41.29	25.27	38.65	35.85	37.70	25.31	33.33	22.58
Toolscaler	GT.	**60.22**	**44.03**	27.87	44.20	**51.09**	39.65	**39.88**	**43.40**	**40.98**	**37.97**	**47.06**	**31.45**
GPT-3.5*	Retrieval	51.43	41.19	34.43	57.59	53.05	46.51	53.37	53.77	37.70	46.20	54.25	54.81
ToolLlama-2*	Retrieval	56.13	49.21	34.70	-	-	-	50.92	53.77	21.31	-	-	-
ToolLlama*	Retrieval	54.60	49.96	51.37	57.70	61.76	45.43	49.08	61.32	31.15	48.73	50.98	44.35
ToolGen*		56.13	52.20	47.54	56.54	49.46	51.96	50.92	62.26	34.42	40.51	39.87	37.90
GPT-4o-mini	Retrieval	52.25	40.41	24.86	**53.16**	50.11	39.38	**47.24**	**52.83**	44.26	**49.37**	**50.33**	**42.74**
ToolLlama-2	Retrieval	28.94	24.69	10.93	28.48	36.93	19.09	25.15	30.19	24.59	26.58	27.45	20.16
ToolGen		52.97	45.13	36.34	45.36	55.56	45.56	36.20	42.45	**49.18**	32.91	42.48	37.90
Toolscaler		**62.78**	**52.04**	**41.26**	52.53	**57.19**	**56.99**	42.94	46.23	45.90	42.41	47.71	37.90

Table 9: Evaluating tool retrieval via ablation studies in Multi-domain settings.

Model	NDCG@1			NDCG@3			NDCG@5		
	I1	I2	I3	I1	I2	I3	I1	I2	I3
Toolscaler	93.00	90.50	89.00	93.87	92.26	88.16	94.85	93.68	91.98
w/o reframe embedding	86.50	88.50	80.00	88.89	88.76	84.11	92.10	92.05	89.79
w/o post-guided	90.50	85.00	79.00	91.47	87.92	82.68	93.41	91.11	89.17
	I1-Tool.	**I1-Cat.**	**I2-Cat.**	**I1-Tool.**	**I1-Cat.**	**I2-Cat.**	**I1-Tool.**	**I1-Cat.**	**I2-Cat.**
Toolscaler	91.00	93.00	91.96	92.20	93.56	91.06	93.89	94.92	92.97
w/o reframe embedding	86.00	88.00	86.43	89.92	91.04	87.26	92.26	92.88	90.27
w/o post-guided	89.00	92.00	88.44	89.85	93.65	88.87	92.65	94.60	91.52

are well aligned.

Notably, Toolscaler significantly outperforms IterFeedback, which is a more complex retrieval system involving multiple models and a feedback mechanism, across both settings, despite being a single-model solution. This highlights the strength and efficiency of our approach in addressing challenging real-world retrieval tasks. Additionally, since Wang et al. (2024b) did not report results on the **Tool.** and **Cat.** datasets, we include them in Table 7. Toolscaler demonstrates robust generalization to unseen tools, maintaining strong performance even in open-set conditions.

In Table 8, we include experimental results from Wang et al. (2024b). Their reported SoPR scores are generally higher than those we reproduced, likely due to their use of GPT-3.5 as both the dialog agent and evaluator—potentially enhanced through additional tool-use-specific tuning. However, considering the significantly higher cost of GPT-3.5 and the fact that it is no longer state-of-the-art, we adopt GPT-4o-mini for evaluation in our experiments. For consistency in SoWR evaluation, we also use GPT-4o-mini (**GT.**) as the reference model.

While the effectiveness of this evaluation is partially influenced by the choice of evaluator (GPT-3.5 vs. GPT-4o-mini), our method, Toolscaler, still demonstrates competitive performance without additional intervention from the ground truth model (**GT.**). Notably, on the **I1** and **I2** subsets, Toolscaler surpasses GPT-3.5 (**GT.**) with task completion rates of 62.78% and 52.04%, respectively. Even against the retrieval-augmented GPT-3.5, Toolscaler achieves comparable results, falling behind only on **I1-Tool**. These findings highlight the robustness of our approach in real-world scenarios involving large-scale tool utilization.

Note that we do not report the SoWR results of GPT-4o-mini Retrieval in the main text, as we observed a strong preference for its own answers, which introduces evaluation bias. To ensure a fair comparison with other methods, we exclude these results from the main discussion but provide the complete results in the appendix.

E.2 Ablation experiment

Table 9 presents the complete ablation results, corresponding to the visualization shown in Figure 2.

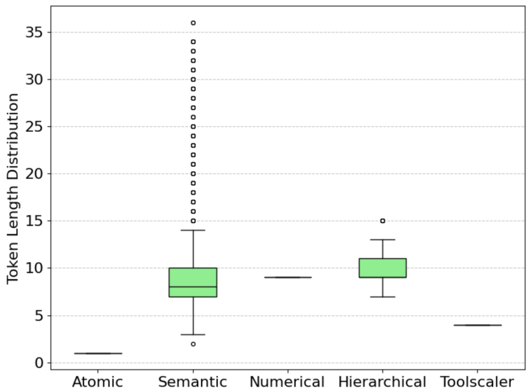


Figure 3: The distribution of the number of subtokens per tool.

In addition to quantitative analysis, we also conducted a detailed comparison of the codebooks generated during the two iterative training processes, and observed that the tool clustering evolved meaningfully across iterations. For example, consider two tools from the same functional category: "*DownloadAuditLog*" and "*BehalfDocuments*". In the first iteration, their code sequences were [90, 464, 192, 356] and [90, 213, 192, 356], respectively, showing a high degree of overlap. In the second iteration, the model refined their representations to [244, 188, 63, 292] and [295, 218, 63, 292], providing clearer differentiation. However, since these tools were co-used within the same training trajectory, they still retained shared substructures (e.g., the last two tokens), reflecting semantic proximity. On the other hand, for tools such as "*Instagram_Data_Service_accountInfo*" and "*Instagram_Data_Service_posts*", their initial code sequences were quite distinct: [249, 148, 124, 45] and [148, 349, 44, 426]. Yet, because they were also jointly used within a trajectory sample, their second-round encodings evolved to [264, 413, 173, 287] and [380, 35, 173, 177], with the third token now shared across both. This indicates that the clustering process can effectively benefit from the procedural and contextual signals embedded in the training data.

E.3 Tokenization comparisons

we perform a statistical comparison of how many subtokens are required to represent each tool across different tokenization methods (see Figure 3). The results show that structure-aware semantic tokenization achieves compact and efficient representations, with an average subtoken count second only to Atomic (which uses exactly one token per tool). In contrast, Semantic and Hierarchical strategies exhibit highly variable subtoken lengths across tools—some being very short, others excessively long—resulting in a scattered distribution that may hinder effective model learning. Notably, both our method and Numerical/Atomic use fixed-length sequences, which contribute to greater stability and learnability in representation.

Furthermore, We augment the experimental results related to various tokenization strategies in Table 10 and Table 11, incorporating both reproduced outcomes and reported results from Wang et al. (2024b). The more comprehensive comparisons reveal that Toolscaler consistently outperforms all competing methods across all datasets, establishing itself as the state-of-the-art in both tool retrieval (NDCG) and tool calling (SoPR and SoPW) tasks. Notably, Toolscaler demonstrates clear advantages in long-tail retrieval scenarios, achieving improvements of 2.91 to 6.15 NDCG@5 points over the Atomic baseline. This gain can be attributed to its fine-grained semantic modeling. In contrast, approaches like Semantic and Atomic, while competitive in isolated scenarios (e.g., Semantic achieves 92.96 NDCG@5 on **I1**), lack dynamic optimization mechanisms, which hinders their ability to generalize in multi-tool interaction settings.

Interestingly, we observe that further training on Trajectories after pretraining with Query-Tool pairs tends to degrade NDCG performance. As shown in Table 10, the most significant drops are seen in the Numerical and Hierarchical methods, followed by Atomic. In contrast, Semantic and Toolscaler experience only marginal degradation, with Toolscaler exhibiting the most stable performance across almost all datasets. This degradation may stem from the distributional mismatch between the Query-Tool supervision and the sequential supervision in Trajectories. While Query-Tool pairs provide explicit relevance signals, Trajectories often introduce noise or indirect supervision, which may mislead models lacking strong semantic grounding. The robustness of Toolscaler can likely be attributed to its structure-aware and semantics-preserving tokenization, which helps maintain consistency across different training paradigms.

E.4 Different model sizes

In order to evaluate both cross-architecture generalizability (Qwen2.5(Hui et al., 2024) vs. LLaMA3) and scaling behavior (from 0.5B to 14B), we conducted experiments across multiple parameter

Table 10: Retrieval performance of different tokenization methods in the Multi-domain setting. The results of ToolGen are directly adopted as the baseline for the Atomic. Results marked with ∗ are directly taken from the original paper (Wang et al., 2024b). All other results are re-evaluated using open-source checkpoints. † indicates models trained with Trajectories, while others are trained with Query-Tool pairs only.

Tokenization	NDCG@1			NDCG@3			NDCG@5		
	I1	I2	I3	I1	I2	I3	I1	I2	I3
Numerical*	83.17	79.20	71.00	84.99	79.23	74.81	88.73	83.88	82.95
Hierarchical*	85.67	82.22	78.50	87.38	82.70	79.47	90.26	86.63	84.15
Semantic*	89.17	83.71	82.00	91.29	84.51	78.86	93.29	88.22	85.43
Atomic*	87.67	83.46	79.00	88.84	86.24	79.80	91.54	88.84	84.79
Numerical	82.00	77.50	81.91	84.18	77.53	76.51	70.00	88.07	84.30
Numerical†	58.50↓**23.5**	49.50↓**28.0**	45.00↓**36.91**	65.78↓**18.4**	56.86↓**20.67**	55.88↓**20.63**	73.62↑**3.62**	63.41↓**24.66**	65.96↓**18.34**
Hierarchical	87.50	77.50	79.00	86.11	78.82	81.44	89.91	83.81	87.47
Hierarchical†	66.00↓**21.5**	61.50↓**16.0**	62.00↓**17.0**	70.33↓**15.78**	64.50↓**14.32**	71.07↓**10.37**	77.89↓**12.02**	71.81↓**12.0**	80.01↓**7.46**
Semantic	90.00	84.50	84.00	91.56	84.33	79.41	92.96	88.44	87.40
Semantic†	86.50↓**3.5**	80.00↓**4.5**	72.00↓**12.0**	86.92↓**4.64**	78.21↓**6.12**	73.45↓**5.96**	90.51↓**2.45**	83.73↓**4.71**	83.34↓**4.06**
Atomic	88.50	84.00	81.00	88.83	85.65	80.83	91.65	89.02	85.83
Atomic†	86.5↓**2.0**	76.00↓**8.0**	73.00↓**8.0**	85.76↓**3.07**	75.68↓**9.97**	74.65↓**6.18**	89.99↓**1.66**	81.92↓**7.1**	83.15↓**2.68**
Toolscaler	**93.00**	**90.50**	**89.00**	**93.87**	**92.26**	**88.16**	**94.85**	**93.68**	**91.98**
Toolscaler†	89.00↓**4.0**	90.00↓**0.5**	84.00↓**5.0**	89.91↓**3.96**	86.21↓**6.05**	79.87↓**8.29**	92.44↓**2.41**	91.15↓**2.53**	87.11↓**4.87**
	I1-Tool.	**I1-Cat.**	**I2-Cat.**	**I1-Tool.**	**I1-Cat.**	**I2-Cat.**	**I1-Tool.**	**I1-Cat.**	**I2-Cat.**
Numerical	83.50	81.50	79.39	85.04	85.57	80.90	88.06	88.88	85.13
Numerical†	68.50↓**15.0**	57.50↓**24.0**	50.75↓**28.64**	73.09↓**11.95**	63.43↓**22.14**	58.68↓**22.22**	77.19↓**10.87**	70.72↓**18.16**	65.25↓**19.88**
Hierarchical	80.50	87.50	86.43	85.48	88.59	86.08	88.19	91.19	89.04
Hierarchical†	72.00↓**8.5**	52.50↓**35.0**	62.81↓**23.62**	71.94↓**13.54**	62.76↓**25.83**	67.07↓**19.01**	79.44↓**8.75**	70.58↓**20.61**	74.33↓**14.71**
Semantic	87.50	89.50	82.91	89.98	90.45	84.44	92.12	93.26	88.03
Semantic†	85.50↓**2.0**	86.00↓**3.5**	72.36↓**10.55**	85.88↓**4.1**	85.96↓**4.49**	77.07↓**7.33**	89.67↓**2.45**	89.39↓**3.87**	82.33↓**5.7**
Atomic	84.00	89.50	83.42	86.40	89.95	86.06	89.52	92.01	88.47
Atomic†	78.00↓**6.0**	80.50↓**9.0**	70.85↓**12.57**	79.16↓**7.24**	82.42↓**7.53**	73.09↓**12.97**	84.08↓**5.44**	86.63↓**5.38**	78.44↓**10.03**
Toolscaler	**91.00**	**93.00**	**91.96**	**92.20**	**93.56**	**91.06**	**93.89**	**94.92**	**92.97**
Toolscaler†	88.00↓**3.0**	89.50↓**3.5**	88.44↓**3.52**	87.62↓**4.58**	89.57↓**3.99**	86.54↓**4.52**	91.75↓**2.14**	92.34↓**2.58**	90.69↓**2.28**

scales in Qwen2.5. The results are summarized in Table 12. Key observations include:

- **Cross-architecture generalization**: The Qwen2.5-7B achieves results very close to LLaMA3-8B across all evaluation subsets, indicating that Toolscaler generalizes well across different model architectures as long as the model has sufficient reasoning capacity.
- **Effect of model size**: As expected, performance improves significantly as model size increases. However, the improvement plateaus after 7B parameters—Qwen2.5-14B shows only marginal gains and even slight declines on certain subsets.
- **Efficiency of small models**: Remarkably, even Qwen2.5-0.5B achieves NDCG@1 > 0.8 on some subsets. Qwen2.5-1.5B performs consistently well across all test sets, suggesting that our framework remains robust and effective even on relatively small models.

These results confirm that Toolscaler is both model-agnostic and size-resilient, capable of transferring across architectures and scaling gracefully with model capacity.

Table 12: Performance and generalization of Toolscaler with different sizes of LLMs as the base model. We reported the NDCG@1 for the tool retrieval stage across all sub-test sets.

Model	I1	I2	I3	I1-Tool.	I1-Cat.	I2-Cat.
Qwen2.5-0.5B	84.00	75.00	68.00	84.50	86.50	80.40
Qwen2.5-1.5B	86.00	82.00	83.00	83.50	87.00	85.93
Qwen2.5-3B	88.00	85.00	83.00	84.00	89.00	86.43
Qwen2.5-7B	92.00	89.50	91.00	89.50	93.50	89.50
Llama3-8B	93.00	90.50	89.00	91.00	93.00	91.96
Qwen2.5-14B	92.50	91.50	91.50	89.00	92.50	90.95

E.5 Adaptability

Table 13: Dataset statistics for the **I1** In-domain experiment after re-partitioning based on toolset size.

Dataset	Scale			
	1%	10%	50%	100%
Tool	469	4,698	23,492	46,985
Train	4,800	44,829	245,499	489,702
Test	7	89	370	457

As demonstrated in Figure 4, structure-aware semantic tokenization is particularly well-suited for large-scale tool scenarios. However, we argue that it is meaningful to test Toolscaler on smaller toolsets to understand how well it performs in more

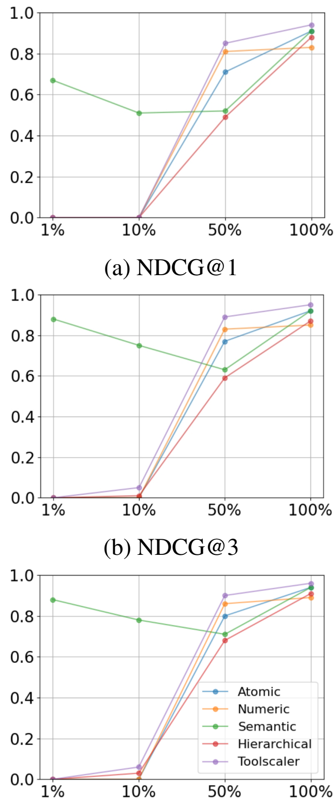
Table 11: Tool calling evaluation for different tokenization methods. $*$ indicates results reproduced from Wang et al. (2024b), where GPT-3.5 was used as the dialogue model, and GPT-3.5 **GT.** served as the reference model for SoWR. In contrast, our experiments are conducted using GPT-4o-mini. Bold values denote the highest performance, considering only the results reproduced in our experimental setting.

Tokenization	Setting	SoPR						SoWR					
		I1	I2	I3	I1-Tool.	I1-Cat.	I2-Cat.	I1	I2	I3	I1-Tool.	I1-Cat.	I2-Cat.
Numerical	GT.	23.21	14.15	12.30	25.42	25.49	15.59	20.86	15.09	22.95	24.05	20.92	13.71
Hierarchical	GT.	30.27	18.24	4.92	28.06	33.33	14.52	22.09	20.75	18.03	24.05	25.49	10.48
Semantic	GT.	51.74	34.59	21.58	36.81	**52.07**	29.84	39.87	36.79	27.87	29.75	45.10	25.00
Atomic	GT.	47.85	34.91	**29.23**	35.76	41.29	25.27	38.65	35.85	37.70	25.31	33.33	22.58
Toolscaler	GT.	**60.22**	**44.03**	27.87	**44.20**	51.09	**39.65**	**39.88**	**43.40**	**40.98**	**37.97**	**47.06**	**31.45**
Numerical*		34.76	29.87	46.99	-	-	-	25.77	33.02	29.51	-	-	-
Hierarchical*		50.20	45.60	32.79	-	-	-	38.04	43.40	29.51	-	-	-
Semantic*		58.79	45.28	44.81	-	-	-	49.69	57.55	26.23	-	-	-
Atomic*		58.08	56.13	44.81	-	-	-	47.85	57.55	29.51	-	-	-
Numerical		21.98	9.12	11.20	20.68	26.14	17.20	16.56	16.04	16.39	20.89	23.53	14.52
Hierarchical		39.16	20.28	17.49	36.29	31.81	14.92	29.45	28.30	26.23	29.11	24.83	14.52
Semantic		50.20	29.72	16.39	33.02	51.42	27.02	39.26	29.24	32.79	29.11	43.79	22.58
Atomic		52.97	45.13	36.34	45.36	55.56	45.56	36.20	42.45	**49.18**	32.91	42.48	37.90
Toolscaler		**62.78**	**52.04**	**41.26**	**52.53**	**57.19**	**56.99**	**42.94**	**46.23**	45.90	**42.41**	**47.71**	**37.90**

conventional or compact scenarios. Therefore, we conducted training and testing under different tool scales and compared them with other tokenization methods.

This experiment considers the tool retrieval task and trains models solely on Query-Tool pairs during the first round. We first partition the entire toolset into subsets of different sizes and filter the training and test data accordingly, as detailed in Table 13. Since randomly sampling tools may disrupt multi-tool collaboration patterns, we conduct both training and evaluation exclusively on the **I1** dataset.

As shown in Figure 4, when the toolset size is limited to 1% or 10% of the full set, only the Semantic maintains high retrieval performance. Other approaches, which rely on learning dedicated representations, perform poorly, likely due to insufficient data for effective training. However, as the toolset scales up to 50% (over 20,000 tools) or even more, the retrieval performance of Semantic begins to decline, while other methods, particularly structure-aware semantic tokenization, show marked improvements. This suggests that tool names alone are not sufficient to capture the nuanced differences among a large number of tools, often leading to ambiguity. In contrast, structure-aware semantic tokenization progressively outperforms other strategies as the toolset grows. This improvement is attributed to its ability to encode fine-grained semantic and structural relationships, enabling the model to distinguish among a large and diverse set of tools with minimal confusion. Its hierarchical and consistent representation further facilitates generalization and scalability across large tool corpora.



(a) NDCG@1

(b) NDCG@3

(c) NDCG@5

Figure 4: NDCG performance of different tokenization methods on **I1** (In-domain), segmented by toolset sizes at 1%, 10%, 50%, and 100% (464, 4,632, 23,145, and 46,985 tools, respectively).

```
{
    "product_id": "api_53da6825-ded3-497c-9b9e-ef8920352d35",
    "tool_description": "Tools for face transformation",
    "home_url": "https://rapidapi.com/toonify-toonify-default/api/toonify/",
    "name": "Toonify",
    "title": "Toonify",
    "pricing": "FREEMIUM",
    "tool_name": "Toonify",
    "score": {
        "avgServiceLevel": 100,
        "avgLatency": 9064,
        "avgSuccessRate": 75,
        "popularityScore": 9.7,
        "__typename": "Score"
    },
    "host": "toonify.p.rapidapi.com",
    "api_list": [
     {
         "name": "caricature_v0_caricature",
         "url": "https://toonify.p.rapidapi.com/v0/caricature",
         "description": "Caricature transformation",
         "method": "POST",
         "required_parameters": [
           {
             "name": "image",
             "type": "BINARY",
             "description": "",
             "default": ""
           }
         ],
         "optional_parameters": [
           {
             "name": "return_aligned",
             "type": "BOOLEAN",
             "description": "Flag to returned cropped and aligned version of the input image",
             "default": false
           },
           ……
         ],
         “code”: “import requests\n\nurl = \”https://toonify.p.rapidapi.com/v0/caricature\“\nquerystring = {\”image\“: \”\“,
                 \”return_aligned\“: false, \”face_index\“: 0, ……",
         "convert_code": "import requests\n\nurl = \"https://toonify.p.rapidapi.com/v0/caricature\"\nquerystring =
                         {\"image\": \"\", \"return_aligned\": false, \"face_index\": 0, ……",
         "test_endpoint": "",
         "statuscode": 1,
         "schema": ""
     },
     ……
    ],
    "category_name": "Video_Images"
}
```


Figure 5: A real RESTful API example. The RESTful API contains one API function (tool).


```
{
    "conversations": [
        {
            "role": "user",
            "content": "My friends and I are organizing a hackathon on ‘web development' and
                       'mobile app development’.
                        We need some inspiration and guidance. Can you fetch the top stories on
                        these topics from Medium.com?",
            "loss": false
        },
        {
             "role": "assistant",
             "content": "<a_186><b_393><c_204><d_29>",
             "loss": true
        }
    ]
}
```


Figure 6: Datasets examples for tool retrieval training. We use "$user$" role to represent inputs and "$assistant$" role to represent outputs.

```
{
    "conversations": [
        {
            "from": "system",
            "value": "You are an AutoGPT, capable of utilizing numerous tools and
functions to complete the given task.\n1.First, I will provide you with the task
description, and your task will commence.\n2.At each step, you need to determine the
next course of action by generating an action token.\n3.Following the token, you will
receive the documentation of the action corresponding to the token. You need to
generate the input of the action, transitioning you to a new state. Subsequently, you
will make decisions about the next steps, and repeat this process.\n4.After several
iterations of generating actions and inputs, you will ultimately complete the task and
provide your final answer.\n\nRemember:\n1.The state changes are irreversible, and you
cannot return to a previous state.\n2.Keep your actions concise, limiting them to best
suits the current query.\n3.You can make multiple attempts. If you plan to try
different conditions continuously, perform one condition per try.\n4.If you believe you
have gathered enough information, generate the action \"<a_296><b_157><c_77><d_372>
with argument give_answer\" to provide your answer for the task.\n5.If you feel unable
to handle the task from this step, generate the action \"<a_296><b_157><c_77><d_372>
with argument give_up_and_restart\".\nLet's Begin!\nTask description: You should use
actions to help handle the real time user querys. Remember:\n1.ALWAYS generate
\"<a_296><b_157><c_77><d_372>\" at the end of the task. And the final answer should
contain enough information to show to the user. If you can't handle the task, or you
find that actions always fail(the function is not valid now), use action
<a_296><b_157><c_77><d_372> with give_up_and_restart.\n2.Only generate actions and
inputs.",
            "loss": false
        },
        {
            "from": "user",
            "value": "\nCan you help me get the content type of a file located at
'https://www.example.com/document.pdf'? I also need to generate a webpage snapshot as
an image. The source URL is 'https://www.example.com'. Please provide me with the image
snapshot.\nBegin!\n",
            "loss": false
        },
        {
            "from": "assistant",
            "value": "I believe the assistant called the \"<a_56><b_313><c_151><d_97>\"
function with the argument \"{\\n  \"source\":
\"https://www.example.com/document.pdf\"\\n}\" to retrieve the content type of the file
located at 'https://www.example.com/document.pdf'. This API call is necessary because
it provides a streamlined way to fetch the content type information of a given file. By
providing the source URL of the file as the argument, the assistant aims to obtain the
content type to further process the file.",
            "loss": false
        },
        {
            "from": "user",
            "value": "Generate the action.",
            "loss": false
        },
        {
            "from": "assistant",
            "value": "<a_56><b_313><c_151><d_97>",
            "loss": false
        },
        {
            "from": "user",
            "value": "Please give the input. Here is the documentation: {'name':
'<a_56><b_313><c_151><d_97>', 'description': 'File: get content type', 'required':
[{'name': 'source', 'type': 'STRING', 'description': '', 'default':
'http://google.com'}], 'optional': [{'name': 'cache-key', 'type': 'STRING',
'description': '', 'default': 'c7b920f57e553df2bb68272f61570210'}]}",
            "loss": false
        },
        {
            "from": "assistant",
            "value": "{\n  \"source\": \"https://www.example.com/document.pdf\"\n}",
            "loss": false
        },
        {
            "from": "function",
            "value": "{\"error\": \"\", \"response\": \"{'source': 'http://google.com',
'mime_type': 'text/html', 'timestamp': '2023-07-10T19:12:09.624239', 'request_id':
'31dc3777-b494-4a84-aef8-189f9f1b2f22'}\"}"
        },
        {
            "from": "assistant",
            "value": "I can see that the assistant called the
\"<a_438><b_147><c_346><d_187>\" function with the argument \"{\\n  \"source\":
\"https://www.example.com\"\\n}\" to generate a webpage snapshot as an image. It seems
the assistant wants to retrieve an image snapshot of the webpage located at
'https://www.example.com'. This API call is necessary because using the
\"<a_438><b_147><c_346><d_187>\" function provides a streamlined way to capture webpage
snapshots as images. By providing the source URL of the webpage as the argument, the
assistant aims to obtain the desired image snapshot for further use or analysis.",
        },
        ……
        {
            "from": "assistant",
            "value": "Since the previous attempt to generate a webpage snapshot as an
image failed, I will now try a different approach by using the
`<a_438><b_176><c_299><d_245>` function instead of `<a_438><b_147><c_346><d_187>`. By
directly generating a PDF from the webpage source, I can obtain a different output
format for further analysis. This alternative approach could potentially provide a
successful solution to capture the desired image snapshot. Let's execute the function
with the argument \"{\\n  \"source\": \"https://www.example.com\"\\n}\" and observe the
outcome.",
            "loss": true
        },
        {
            "from": "user",
            "value": "Generate the action.",
            "loss": false
        },
        {
            "from": "assistant",
            "value": "<a_296><b_157><c_77><d_372>",
            "loss": true
        },
        {
            "from": "user",
            "value": "Please give the input. Here is the documentation: {'required': [],
'optional': [{'name': 'give_answer', 'description': 'Output the answer', 'type':
'string'}, {'name': 'give_up_and_restart', 'description': 'Unable to handle the task
from this step', 'type': 'string'}]}",
            "loss": false
        },
        {
            "from": "assistant",
            "value": "{\n  \"return_type\": \"give_answer\",\n  \"final_answer\": \"The
content type of the file located at 'https://www.example.com/document.pdf' is
'text/html'. Unfortunately, I encountered difficulties in generating the webpage
snapshot as an image. Could you please provide me with an alternative source URL for
generating the image snapshot?\"\n}",
            "loss": true
        }
    ]
}
```


Figure 7: An example for tool calling training.

Q	Can you help me get the content type of a file located at 'https://www.example.com/document.pdf'? I also need to generate a webpage snapshot as an image. The source URL is 'https://www.example.com'. Please provide me with the image snapshot. Begin!
p_1	I believe the assistant called the "PragmavantApi&&/file/content-type" function with the argument "{"source": "https://www.example.com/document.pdf"}" to retrieve the content type of the file located at 'https://www.example.com/document.pdf'. This API call is necessary because it provides a streamlined way to fetch the content type information of a given file. By providing the source URL of the file as the argument, the assistant aims to obtain the content type to further process the file.
	Generate the action.
d_1	PragmavantApi&&/file/content-type
	Please give the input. Here is the documentation: {'name': 'PragmavantApi&&/file/content-type', 'description': 'File: get content type', 'required': [{'name': 'source', 'type': 'STRING', 'description': '', 'default': 'http://google.com'}], 'optional': [{'name': 'cache-key', 'type': 'STRING', 'description': '', 'default': 'c7b920f57e553df2bb68272f61570210'}]}
c_1	{"source": "https://www.example.com/document.pdf"}
f_1	{"error": "", "response": "{'source': 'http://google.com', 'mime_type': 'text/html', 'timestamp': '2023-07-10T19:12:09.624239', 'request_id': '31dc3777-b494-4a84-aef8-189f9f1b2f22'}"}
p_2	I can see that the assistant called the "PragmavantApi&&/web/snapshot" function with the argument "{"source": "https://www.example.com"}" to generate a webpage snapshot as an image. It seems the assistant wants to retrieve an image snapshot of the webpage located at 'https://www.example.com'. This API call is necessary because using the "PragmavantApi&&/web/snapshot" function provides a streamlined way to capture webpage snapshots as images. By providing the source URL of the webpage as the argument, the assistant aims to obtain the desired image snapshot for further use or analysis.",
	Generate the action.
d_2	PragmavantApi&&/web/snapshot
	... ...
c_5	Finish
	Please give the input. Here is the documentation: {'required': [], 'optional': [{'name': 'give_answer', 'description': 'Output the answer', 'type': 'string'}, {'name': 'give_up_and_restart', 'description': 'Unable to handle the task from this step', 'type': 'string'}]}
$\mathcal{A}$	{"return_type": "give_answer", "final_answer": "The content type of the file located at 'https://www.example.com/document.pdf' is 'text/html'. Unfortunately, I encountered difficulties in generating the webpage snapshot as an image. Could you please provide me with an alternative source URL for generating the image snapshot?"}

Figure 8: An example of a trajectory demonstrates the typical stages that current LLM-based agents take when using tools.