Usage-Driven Tool Retrieval in LLMs: A Multi-Stage Approach

Anonymous ACL submission

Abstract

Recent advancements in function calling and tool use have significantly enhanced the capabilities of large language models (LLMs) by enabling them to interact with external information sources and execute complex tasks. However, the limited context window of LLMs presents challenges when a large number of tools are available, necessitating efficient methods to manage prompt length and maintain accuracy. Existing approaches, such as fine-tuning LLMs or leveraging their reasoning capabilities, either require frequent retraining or incur significant latency overhead. A more efficient solution involves training smaller models to retrieve the most relevant tools for a given query, although this requires high-quality, domain-specific data. To address those challenges, we present a novel framework for generating synthetic data for tool retrieval applications and an efficient data-driven tool retrieval strategy using small encoder models. Empowered by LLMs, we create ToolBank, a new tool retrieval dataset that reflects real human user usages. For tool retrieval methodologies, we propose novel approaches: (1) Tool2Vec: usage-driven tool embedding generation for tool retrieval, (2) ToolRefiner: a staged retrieval method that iteratively improves the quality of retrieved tools, and (3) MLC: framing tool retrieval as a multi-label classification problem. With these new methods, we achieve improvements of up to 27.28 in Recall@K on the ToolBench dataset and 30.5 in Recall@K on ToolBank.

1 Introduction

Recently, function calling and tool use has emerged as a powerful paradigm for using large language models (LLMs) (Patil et al., 2023; Schick et al., 2024; OpenAI, 2023; Cai et al., 2023). Rather than relying solely on the model's parametric knowledge, function calling and tool use enable the model to interact with the world (Erdogan et al., 2024; Chen and Li, 2024). This approach allows the model to achieve specific tasks, such as accessing information beyond the LLM's knowledge cut-off date, solving complex math problems, and executing complex planning (Trinh et al., 2024; Silver et al., 2024; Karpas et al., 2022; Chen et al., 2022).

However, since function calling requires passing in the tool's description and signature into the model's context window, it is often infeasible to put information about potentially thousands of functions due to context window limitations. Additionally, even when using models with longer context windows, long context inference leads to system-level and accuracy challenges, necessitating the need for smaller prompts (Kim et al., 2023; Jha et al., 2024; Erdogan et al., 2024). Therefore, selectively retrieving tools to present to the model can greatly reduce prompt lengths while preserving accuracy.

To address the issue of the limited context window in LLMs when the number of available tools exceeds the model's capacity, several methods have been proposed. One approach involves fine-tuning LLMs with new tokens that specify tools (Hao et al., 2024). However, due to the computational cost of fine-tuning LLMs, this may not be practical when new tools are frequently introduced. Another approach leverages the reasoning capabilities of LLMs, allowing the models to select the appropriate set of tools from a large pool (Du et al., 2024; Shinn et al., 2024; Wang et al., 2023; Yuan et al., 2024). Despite the LLMs' ability to learn and choose tools effectively, this method incurs significant latency overhead, making it less practical in various use cases where real-time responses are critical. Therefore, training a separate smaller model to retrieve tools that are most relevant to the given query has emerged as an efficient yet powerful solution (Qin et al., 2024; Qu et al., 2024; Anantha et al., 2023; Zheng et al.,

2024b). This method is the most efficient way to address the problem of prompt length exceeding the LLMs' context window. In addition to circumventing lengthy prompts, efficient tool retrieval systems can be developed by training small models specialized for specific domains (Erdogan et al., 2024; Chen and Li, 2024). The downside of these systems is the need of domain-specific data to effectively train retrieval models.

However, previous work (Erdogan et al., 2024) has demonstrated that high-quality tool retrieval data can be generated with LLMs (Lee et al., 2024; Chen et al., 2024; Wei et al., 2024), and users can train small and efficient tool retrieval models with the generated data for tool-augmented LLMs. By doing so, it not only optimizes the use of the context window but also enhances the performance and accuracy of LLMs through domain-specific expertise.

Based on this finding, we introduce a framework for creating large scale synthetic data for tool retrieval applications, as well as an efficient data-driven tool retrieval strategy using small encoder models. For dataset generation, we utilize the strong synthetic data generation capabilities of large language models (OpenAI, 2024; AI@Meta, 2024; Jiang et al., 2024a) to create tool retrieval dataset, ToolBank. On the tool retrieval side, we introduce novel approaches: (1) usage-driven tool embedding generation, Tool2Vec, (2) refining and improving the tool retrieval result, ToolRefiner. We show that those methods perform better than prior work relying on computing similarity between a user query and tool descriptions (Qin et al., 2024; Qu et al., 2024; Zheng et al., 2024b; Yuan et al., 2024).

Additionally, we frame tool retrieval as a multi-label classification problem and train multiple different classification models, giving way to an efficient staged retrieval method.

In more detail, we make the following contributions:

- We introduce a framework for creating domain-specific tool retrieval datasets and instantiate three new datasets for tool retrieval which, when judged for quality by GPT-4-turbo, scores a 60% win rate compared to ToolBench's queries (section 3).
- We propose usage-based tool retrieval, as opposed to description-based tool retrieval. Additionally, we hierarchically use classification models to iteratively improve the quality of retrieved tools (section 4).
- On the hardest ToolBench split, our method achieves over 25% higher Recall compared to ToolBench's retriever. Additionally, on our domain-specific datasets, our methods outperform description-based retrieval by over 30% (section 5).

2 Related Work

2.1 Function Calling and Tool Use

Function calling allows LLMs to interact with the world and agentic environments by filling in parameters to API functions and other tools. Typically, function descriptions and signatures are provided in the model's context window. For accurate function calling, models must be able to choose the proper functions for the task and be able to fill in the correct parameters to those functions. Large models such as GPT-4 have demonstrated impressive function calling capabilities (Kim et al.). However, smaller models (Patil et al., 2023; Srinivasan et al., 2023), such as 7B and 13B models, have also been developed specifically for function calling tasks. The ToolBench (Qin et al., 2024) dataset is a popular function calling dataset consisting of real-world APIs that was used to fine-tune a 7B LLaMA model for tool use.

2.2 Tool Retrieval

As discussed in subsection 2.1, function descriptions and signatures are provided in the model's context window for applications relying on function calling. However, real-world applications often have hundreds or thousands of tools (Qin et al., 2024). Providing information about all tools to the model may not be possible due to context length limits. Furthermore, even when using models with longer context windows, providing all the tools in the prompt leads to significant compute and memory overheads (Kim et al., 2023). To address this, various tool retrieval methods have been proposed to select and provide only the relevant tools for incoming user queries instead of providing them all.

A notable approach to enhance tool retrieval performance is leveraging another LLM. AnyTool (Du et al., 2024) proposes to use GPT-4 for API retrieval and to further enhance retrieval performance through an iterative self-reflection method. Similarly, (Xu et al., 2024) incorporates a refiner LLM

that iteratively refines user queries to boost retrieval performance. However, using LLMs for tool retrieval, along with iterative invocation, results in significant latency overhead of up to several seconds (Xu et al., 2024), limiting their use in various real-time applications.

Dense retrieval methods offer an efficient alternative, where each tool's description is embedded using an embedding model, and tools with the highest similarity to the embedding of the incoming user query are retrieved (Qin et al., 2024). ProTIP (Anantha et al., 2023) adapts a dense retrieval model for iterative multi-tool selection. ToolkenGPT (Hao et al., 2024) proposes to learn an embedding of each tool that can be immediately used as an input token to LLMs. COLT (Qu et al., 2024) improves tool retrieval performance by fine-tuning the pre-trained encoder model through four distinct stages: semantic learning, collaborative learning, list-wise learning, and contrastive learning.

Tool2Vec provides a different view of tool retrieval, which is tool embedding generated based on usage. It uses the user query embedding instead of tool description embedding to generate tool embeddings for retrieval. A notable work is EasyTool (Yuan et al., 2024) which enhances tool leverages LLMs to rewrite tool descriptions, reducing inconsistency, redundancy, and incompleteness, ultimately improving retrieval performance. While EasyTool also proposes LLMs generate usage examples, these serve to provide in-context examples rather than directly improving the performance of the retriever models as in our work.

3 Dataset Generation

We introduce a modular framework for generating custom datasets tailored for tool retrieval, with the goals of (1) demonstrating that users can create sufficiently large domain-specific datasets powered by LLMs (Lee et al., 2024; Chen et al., 2024) for small tool retrieval models, and (2) addressing the limitations inherent in existing benchmarks (Qin et al., 2024; Chen et al., 2023; Xu et al., 2024; Du et al., 2024), which often lack coherent tool integration and query naturalness.

Particularly for the second aspect, current benchmarks frequently pair tools without considering their natural co-occurrence, leading to impractical and inconsistent combinations (Qin et al., 2024; Huang et al., 2024; Qu et al., 2024). For example, a query from ToolBench— "Search for the companies that have been modified recently and fetch the lyrics for the song 'Bad' by Michael Jackson" pairs the `360 Business Tool` tool with the `Chart Lyrics` tool, reflecting a clear mismatch in tool relevance. This is because ToolBench randomly samples multiple tools from the tool pool, without much consideration of their co-ocurrance.

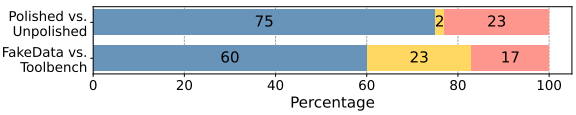

Figure 1: Comparison of naturalness, fluency, and coherence of queries. We first compare polished and unpolished queries within ToolBank, with blue/yellow/red bars indicating the number of times polished queries won, tied, or lost. Then, we compare queries from ToolBank to those from Toolbench, using the same color scheme to represent the outcomes. We randomly sample 100 queries from each dataset and ask GPT-4-turbo to judge which query is more natural, fluent, and coherent (Zheng et al., 2024a).

Moreover, due to the pairing of irrelevant tools, these benchmarks tend to be overly structured and verbose, resembling step-by-step queries rather than the more fluid, natural language typically used in real-world scenarios. For instance, a query such as "Please provide me with details of breweries that are dog-friendly and have a patio, and include race details for race ID 207660, covering horses, jockeys, trainers, and their positions," showcases an unnatural pairing of unrelated tools, driven by a rigid, instructional style.

To this end, we introduce a domain-specific tool retrieval dataset generation framework and accordingly a coherent and natural tool retrieval dataset ToolBank that addresses limitations of existing benchmarks. The dataset generation framework aims to create the tool retrieval dataset that respects the natural co-occurrence of tools while ensuring more natural, real-world query queries, which consists of the following two stages:

- **Query Generation**: In this stage, we first sample T tools randomly from the entire tool set. In contrast to previous works where LLMs are prompted to use all T tools to generate an query (Qin et al., 2024; Chen et al., 2023; Xu et al., 2024; Du et al., 2024), we allow them to select M tools, where $M < T$, that are coherent and contextually aligned. This approach promotes the natural co-occurrence of tools. We used $T = 10$ and $M \in [2, 5]$ throughout our generation process, where we found sufficiently large T critical for LLMs to select tools that align contextually. Ad-

ToolBench Queries	ToolBank Queries
I'm planning a surprise party for my best friend and I need some unique translations for invitation cards. Can you search for translations from English to Italian for the phrase 'You're invited!' using the search translations API? Also, calculate the love percentage between John and Alice	Reorganize a 3D array of sensor readings into shape (time, sensor, feature) to identify the indices of the maximum reading values across all sensors for each time step.
I want to flip a coin to make a decision. Can you provide me with the outcome of a coin flip, heads or tails? Additionally, I'm curious about the current exchange rate between two specific currencies, which I will provide later	Transform customer purchase history data from a broad to a deep format to identify trends in spending behaviors through percentile values of total purchase amounts.

Figure 2: Qualitative analysis comparing the queries in Toolbench and ToolBank. We randomly sample 2 examples from each dataset. Queries in Toolbench often follow an artificial pattern like "Do this, do this, and do this," resulting from random sampling of multiple tools from RapidAPI Hub. In contrast, ToolBank queries are more natural, resembling real human queries to LLMs, with coherent and related tools better aligned to user needs.

ditionally, we provided 5 randomly sampled in-context examples to enhance generation quality and diversity.

- **Query Polish**: Despite our query generation process improving tool co-occurrence, LLMs often produce step-by-step queries that seem unnatural. To address this, we introduce an additional step to polish these initial, often robotic queries into fluent and concise English that more closely mirrors user queries in natural settings.

We provide a qualitative analysis comparing ToolBank against Toolbench (Qin et al., 2024), one of the most widely adopted benchmarks for tool retrieval in Figure 1 and 2. The study illustrated in Figure 1 directly evaluates the naturalness, fluency, and coherence of queries. We randomly sample 100 queries from both the unpolished and polished versions of ToolBank, as well as 100 queries from Toolbench. GPT-4-turbo is then tasked with judging which queries are superior based on the aforementioned criteria (Zheng et al., 2024a). The results demonstrate that query Polish consistently generates queries that outscore both the baseline unpolished queries and the queries from the Toolbench dataset.

We also provide examples randomly sampled from Toolbench and ToolBank in Figure 2. We observe that the format of queries in Toolbench often follows the pattern “Do this, do this, and do this," which results from randomly sampling multiple tools from RapidAPI Hub. This format is somewhat artificial compared to how real human users give queries to LLMs for certain tasks. Additionally, some queries directly or indirectly mention the required APIs, simplifying tool retrieval. In contrast, the queries sampled from ToolBank are more natural, closely resembling how real human users are likely to ask LLMs to perform tasks. Furthermore, the tools required for each user query task in ToolBank are more coherent and related to each other, ensuring better alignment with user needs. For further detail, please refer to Appendix B.

4 Approaches

In this section, we propose two novel approaches to the tool retrieval problem: (i) usage-driven embedding generation (Section 4.1) and (ii) reformulation of tool retrieval as a tool classification problem (Section 4.2). Then, in Section 4.3, we demonstrate how these two methodologies can be combined to achieve high-performance tool retrieval.

4.1 Tool2Vec: Usage-Driven Embedding Generation

Previous tool retrieval methods have relied on tool descriptions to obtain embeddings of each tool for dense retrieval (Anantha et al., 2023; Qin et al., 2024; Qu et al., 2024; Yuan et al., 2024). However, this approach may be suboptimal due to the semantic disparity between tool descriptions and user queries. Figure 3 (Left) illustrates how tool descriptions and user queries can be disjoint in the embedding space, making tool retrieval based on embedding similarity challenging. This issue persists even when the descriptions are augmented with additional information, such as tool code, to improve retrieval performance (Yuan et al., 2024; Zheng et al., 2024b; Du et al., 2024).

To reduce the distributional gap between query and tool embeddings for retrieval, we propose Tool2Vec, the usage-driven tool embedding generation. Instead of using tool descriptions, we propose to use *user queries* to obtain tool embeddings. In more detail, if we have multiple user queries that use a specific tool, we use the average embeddings of those user queries as the Tool2Vec embedding that represents the tool. For example, in Figure 4, we have multiple user queries that use the tool `find_email_address`, such as

"What is Anna's email address?" In this case, we use an embedding model (e.g., E5 (Wang et al., 2022)) to obtain the embedding for each user query, and the average of these embeddings is used as the Tool2Vec embedding for the tool `find_email_address`. Likewise, the Tool2Vec embedding for the tool `find_weather` can be obtained the same way using the associated user queries. As shown in the figure, since the Tool2Vec embeddings of these tools are derived from user queries, they are closer to the incoming user query in the embedding space compared to embeddings derived from tool descriptions.

To further justify the benefits of Tool2Vec's usage-driven tool embedding generation, we perform an analysis as illustrated in Figure 3. The left figure is a t-SNE visualization of the embeddings of the user queries, Tool2Vec, and tool descriptions. It shows that the query embeddings form clusters, with Tool2Vec embeddings typically positioned at the centroids of these clusters. The tool description embeddings, however, are scattered outside of the distributions of instruction embeddings. Evidently, this is due to the semantic gap between the tool description and user query.

The right figure is the box plots with interquartile ranges (IQR) of the cosine similarity between the instruction and tool embeddings. It shows two distributions: 'Positive' for the similarity between instruction embeddings and the embeddings of tools used to process the given instructions, and 'Negative' for the similarity between instruction embeddings and the embeddings of tools not used. For Tool2Vec embeddings, the positive and negative distributions do not overlap, indicating a clear distinction. However, the cosine similarity distributions for tool descriptions show significant overlap between positive and negative, implying that the traditional tool description embeddings are less effective at distinguishing between relevant and irrelevant tools compared to the Tool2Vec embeddings.

4.2 Tool Retrieval as a Multi-Label Classification Problem

In this section, we suggest a reformulation of the multi-tool retrieval problem as a multi-label classification problem. Prior work on tool retrieval (Qin et al., 2024; Qu et al., 2024; Anantha et al., 2023; Zheng et al., 2024b) relies on metric learning techniques, including contrastive loss (Chopra et al., 2005) and triplet loss (Hoffer and Ailon, 2015), to produce useful embeddings from tool descrip-

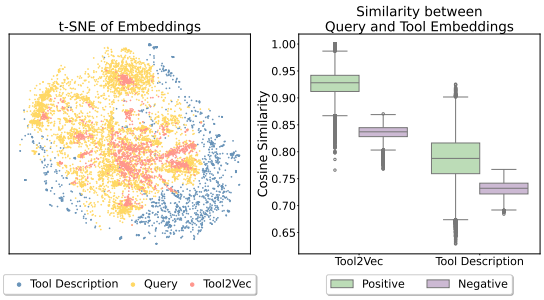

Figure 3: (Left) t-SNE visualization of embeddings for queries, Tool2Vec, and tool descriptions. (Right) Cosine similarity between instruction and tool embeddings. The figure displays two distributions for both Tool2Vec embeddings and tool description embeddings: one labeled 'Positive,' representing cosine similarity between queries and the embeddings of tools used for those instructions, and the other labeled 'Negative,' representing cosine similarity between instructions and the embeddings of tools not used for those queries.

tions. However, in settings where instructions and associated tool labels are abundant, tool retrieval can alternatively be formulated as a multi-label classification problem. Furthermore, given the rise of synthetic data generation methods (Chen et al., 2024; Lee et al., 2024; Wei et al., 2024; Cao et al., 2023), it has become possible to construct such labeled high-quality pairs synthetically with the competent LLMs (AI@Meta, 2024; OpenAI, 2024; Jiang et al., 2023, 2024b), as demonstrated in Section 3.

Given this classification viewpoint of tool selection, there are multiple applicable modeling strategies. One straightforward approach involves training a model that takes the instruction as input and outputs the classification logits for each tool, as illustrated in the left figure of Fig.5. When a user query is provided, such as "What is the weather today?", we assign a label of 1 to all required tools and a label of 0 to unused tools to training the multi-label classifier. In this example, the `find_weather` tool receives a label of 1, while other tools receive a label of 0. To achieve this, we fine-tune the pre-trained BERT-base model (He et al., 2023; Devlin et al., 2019), which features a $H \times T$ classification head operating on the output [CLS] token. Here, H represents the dimension of the [CLS] token, and T denotes the total number of tools in the dataset.

4.3 ToolRefiner

We introduce ToolRefiner, an approach that enhances tool retrieval performance on top of any tool retrieval method by combining the methods outlined in 4.1 and 4.2. As a high-level summary, candidate tools are retrieved with efficient retrieval

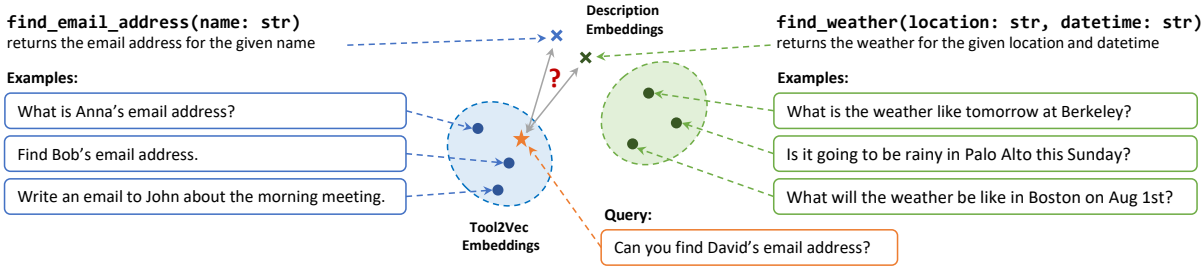

Figure 4: Illustration of how user query embeddings are used as tool embeddings. The embeddings of example queries in the left side of figure corresponds to the tool `find_email_address` Tool2Vec embedding. If multiple queries use the same tool, their embeddings are averaged. Likewise, the Tool2Vec embedding of `find_weather` is the average of the embeddings from the examples shown on the right side of the figure. The disjoint embedding distributions reflect the different semantics of the two sets of examples. However, the description embeddings of those two tools are not close to each cluster because of the semantic domain gap between query and tool description, which leads to the suboptimal retrieval performance.

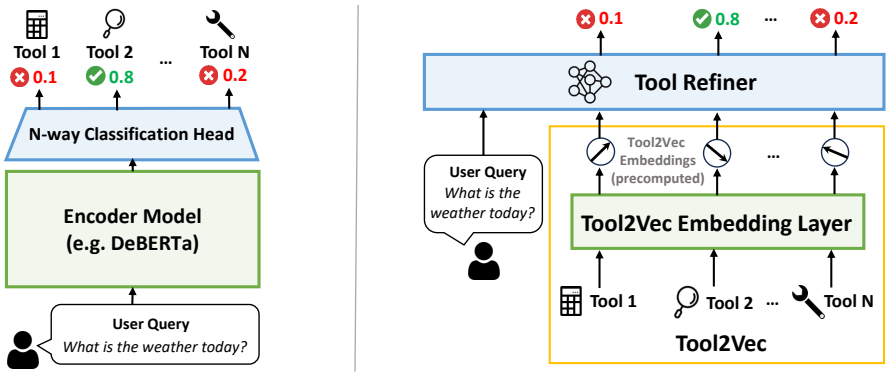

Figure 5: (Left) Illustration of MLC: The encoder model (e.g., DeBERTa) takes user query tokens as input and outputs the probability of each tool. We fine-tune the pre-trained encoder model using binary cross-entropy loss for each tool. (Right) Illustration of ToolRefiner: The fine-tuned encoder model takes the user query and Tool2Vec embeddings of retrieved tools as inputs. We precompute the Tool2Vec embeddings and use them in conjunction with the user query. The pre-trained encoder model is then fine-tuned with softmax loss.

methods such as cosine-similarity-based retrieval. ToolRefiner then classifies whether the retrieved tools are relevant or not.

As illustrated in Fig. 5, we compute the Tool2Vec embedding of each tool by the method introduced in Section 4.1. The tool retriever retrieves top-N tools based on the user query. Then ToolRefiner takes the user query and the Tool2Vec embeddings of the retrieved N tools to classify which tools are needed to process the user query. We fine-tune the pre-trained DeBERTa-V3 (He et al., 2023) xsmall model to get ToolRefiner. Similar to MLC, we assign a label of 1 to all required tools and a label of 0 to unused tools. For instance, if the user query is "What is the weather today?", the `find_weather` tool receives a label of 1, while other tools receive a label of 0. We then calculate and minimize the softmax loss. We observe that softmax loss performs better than binary cross-entropy loss. The same trends have been observed in other domains (Joulin et al., 2016; Mahajan et al., 2018). Notably, if the Tool2Vec embedding is pre-computed, ToolRefiner can be used on top of any other retrieval methods to improve the performance.

This approach is analogous to passage reranking (Nogueira and Cho, 2019; Yilmaz et al., 2019), which determines the ranking of retrieved documents based on their similarity to the query. Similar to passage reranking, the Tool2Vec is trained as a classification model. The key difference is that while traditional passage rerankers evaluate and order the similarities of retrieved documents one by one in relation to the query, ToolRefiner simultaneously reranks all retrieved tools. Furthermore, the reranker operates directly on Tool2Vec embeddings.

5 Experiments

In this section, we describe experimental results that validate the effectiveness of our proposed methods, Tool2Vec, MLC, and ToolRefiner on various benchmarks including ToolBench (Qin et al., 2024) and ToolBank.

Table 1: Comparison of tool retrieval results on the ToolBench dataset. We compared our methods against two baselines: the ToolBench retriever (Qin et al., 2024) and COLT (Qu et al., 2024). Evaluation metrics include Recall@K, where K values are 3, 5, and 7. In the table, R@K stands for Recall@K. The best-performing method is highlighted in boldface, while the second-best performing method is underlined. We reproduce the ToolBench retriever results based on the original codebase. For the other baseline method, COLT, we report the numbers available in the paper (Qu et al., 2024).

	Method	Toolbench I1			Toolbench I2			Toolbench I3		
		R@3	R@5	R@7	R@3	R@5	R@7	R@3	R@5	R@7
Baseline	ToolBench Retriever	79.97	90.19	93.21	67.25	78.25	85.75	54.07	63.88	73.73
	COLT	-	-	-	75.72	85.03	-	76.63	85.50	-
Ours	Tool2Vec	85.88	93.29	94.42	72.79	79.67	82.75	75.23	84.90	86.60
	MLC	91.80	96.00	96.67	80.67	85.63	87.46	**81.35**	86.27	88.27
	ToolRefiner + Tool2Vec	89.63	95.33	96.17	76.83	84.42	86.38	80.58	**87.80**	**89.70**
	ToolRefiner + MLC	**91.84**	**96.83**	**97.01**	**82.89**	**87.92**	**88.96**	79.83	86.91	88.98

Table 2: We compare tool retrieval outcomes using the ToolBank dataset. The baseline consists of methods that identify tools based on their descriptions. We evaluate performance using the evaluation metric Recall@K for K values of 3, 5, and 7. The results are organized into three sections: the first three columns show outcomes using BankNumpy, the following three columns display the results with BankPandas, and the final three columns present the results for BankAWS. We present the E5-base results fine-tuned with the tool description as the baseline. The best-performing method is highlighted in boldface, while the second-best performing method is underlined.

	Method	BankNumpy			BankPandas			BankAWS		
		R@3	R@5	R@7	R@3	R@5	R@7	R@3	R@5	R@7
Baseline	Description-Based Retriever	50.82	64.09	71.84	27.86	34.90	40.00	41.92	46.46	49.13
	COLT	67.43	81.32	83.31	41.63	52.45	53.71	66.16	77.90	83.10
Ours	Tool2Vec	52.97	64.18	71.11	36.52	42.01	45.17	55.38	63.14	67.98
	MLC	70.35	80.78	84.73	41.49	49.69	54.34	70.99	79.69	82.91
	ToolRefiner + Tool2Vec	71.61	79.52	82.22	42.94	47.65	49.33	69.12	74.08	75.43
	ToolRefiner + MLC	**73.82**	**84.24**	**87.47**	**47.76**	**55.28**	**59.13**	**72.42**	**81.17**	**84.49**

5.1 ToolBench

In Table 1, we evaluate our proposed methods in the ToolBench dataset (Qin et al., 2024), comparing their performance against two established baselines: the ToolBench Retriever (Qin et al., 2024), and COLT (Qu et al., 2024). We observe that our methods constantly outperform the baselines with large margins.

5.1.1 Experimental Details

For benchmarking, we use the ToolBench dataset, which is the current standard benchmark for multi-tool retrieval. The data set is divided into three subsets (I1, I2, and I3), and each subset corresponds to different levels in the RapidAPI Hub tool hierarchy. As the subset number increases from I1 to I3, the tools used are sampled from higher levels of the hierarchy. This means that I3 involves more complex or broadly categorized tools compared to I1 and I2.

For all methods used in these experiments, pre-trained encoder models are fine-tuned to each subset of the dataset. Specifically, the ToolBench retriever is a fine-tuned SentenceBERT (Reimers and Gurevych, 2019), which itself is a fine-tuned BERT-base model (Devlin et al., 2019) with a model size of BERT-base (110M parameters). The COLT retriever is a fine-tuned Contriever (Izacard et al., 2022), which is also a fine-tuned BERT model of the same size of the BERT base model.

For our methods, MLC and ToolRefiner, we use DeBERTaV3 (He et al., 2023). Specifically, MLC uses DeBERTaV3-base (86M parameters) and ToolRefiner uses DeBERTaV3-xsmall (22M parameters). To get Tool2Vec embedding, we fine-tune pre-trained E5-base (Wang et al., 2022) model. The model is fine-tuned with triplet loss for one epoch.

5.1.2 Result Analysis

Table 1 presents the performance comparison. The first two rows show the baseline methods: ToolBench retriever (Qin et al., 2024) and COLT retriever (Qu et al., 2024). The last four rows display our methods: Tool2Vec, MLC, ToolRefiner combined with Tool2Vec, and ToolRefiner combined with MLC. We use Recall@K as the evaluation metric, with K values of 3, 5, and 7. We do not include nDCG, as used in (Qin et al., 2024; Qu et al., 2024), because it is not suitable for the tool retrieval context where tool relevance is binary and the order of retrieved tools does not matter. Instead, we choose Recall as the primary evaluation metric. The results for the ToolBench retriever are reproduced using the original codebase, while the Recall values for COLT are taken from (Qu et al., 2024) since the codebase is unavailable to reproduce the

results.

MLC and ToolRefiner consistently outperform the baseline methods by significant margins across all ToolBench subsets. Tool2Vec outperforms the ToolBench retriever across all subsets but falls short of the COLT retriever. Comparing the third and fifth rows in Table 1, ToolRefiner achieves up to 3.8 additional Recall@K across all subsets. For MLC, ToolRefiner shows improvements of up to 2.3 Recall@K for subsets I1 and I2.

5.2 ToolBank

In this section, we benchmark the methods introduced in Section 4 with our new dataset, ToolBank. The results are summarized in Table 2. The baseline is a description based retrieval method. Our methods always perform better than the baseline.

5.2.1 Experimental Details

The baseline used in this experiment is E5-base model, fine-tuned with the description of tools. The model is fine-tuned with triplet loss for one epoch. Similar to Section 5.1, we fine-tune pre-trained encoder model for MLC, Tool2Vec, and ToolRefiner.

For all subsets in this data, we split the training set into an 8:2 ratio for training and validation. We conduct hyperparameter tuning using the validation set and report performance on the test set using the best-performing hyperparameters. To avoid overfitting, we only evaluate the test set once across all experiments.

5.2.2 Result Analysis

In Table 2, the first row is the result with the description-based baseline and other rows are results with our methods, Tool2Vec, MLC, ToolRefiner combined with Tool2Vec, and ToolRefiner combined with MLC. All of our methods outperform the baseline by up to 30 additional Recall@K. We observe that ToolRefiner improves the retrieval results consistently for both Tool2Vec and MLC. Especially, the improvement is remarkable when ToolRefiner is used with Tool2Vec, which have the gain up to 21 for Recall@K.

We observe that our models perform worse on the Pandas dataset; specifically, the ToolRefiner combined with MLC achieves 25% less Recall@3 on BankPandas dataset than both the BankNumpy and BankAWS datasets. BankPandas dataset contains various data types like time series, periods, intervals, and indexes; hence, the model is mostly confused about which data type to operate on. For the further detail, please refer to Appendix B.4.2

5.3 Additional Results

ToolLens. ToolLens is another tool retrieval benchmark (Qu et al., 2024). We compare our methods to other baselines including description-based retriever and COLT. ToolRefiner + MLC outperforms COLT, which is SOTA baseline for ToolLens by 2.19% for R@3 and 0.98% for R@5. Details on this experiment is available in Appendix A.2.

Analysis on Our Methods. We find that ToolRefiner + Tool2Vec excels in handling queries of varying complexity and maintains a more uniform failure rate across diverse tools, averaging fewer mistakes. In contrast, Tool2Vec struggles with simpler queries, frequently misclassifies specific tools, and exhibits a higher overall failure rate. Complete analyses are available in Appendix A.4.

Other results. We demonstrate reduced latency and GPU memory consumption (see Appendix A.6). To investigate the importance of tool retrieval performance, we benchmark ours on TinyAgent dataset (Erdogan et al., 2024) (see Appendix A.3). Our approach yields a 3% gain in R@3 and a 7% increase in success rate compared to a description-based baseline, highlighting the critical impact of enhanced tool retrieval on end-to-end performance.

6 Conclusions

We propose a framework for creating high-quality synthetic datasets for tool retrieval, as well as a method for leveraging these datasets to train small models for usage-based tool retrieval. In order to build specialized tool retrieval applications, domain-specific dataset generation is critical. Furthermore, to avoid context window limitations and system overheads caused by long prompts, efficient tool retrieval with small models is a necessary component of many function calling systems. LLMs demonstrate impressive synthetic dataset generation capabilities which we used to create specialized tool retrieval dataset, ToolBank. Additionally, our data-driven retrieval strategy outperforms description-based retrieval by as much as 30% on these datasets. On ToolBench, our retrieval method achieves over 25% higher Recall than ToolBench's description-based retriever.

7 Limitations

While our proposed framework demonstrates promising results in tool retrieval tasks, several limitations merit discussion. First, the reliance on synthetic data generation, although effective for constructing large, domain-specific datasets, raises concerns about data quality and representativeness. LLMs that generate these synthetic examples may introduce hidden biases, potentially misrepresenting real-world user queries or overfitting to the model's inherent language patterns. Second, usage-based embeddings presuppose the availability of sufficient usage examples for each tool; in scenarios where certain tools are infrequently utilized or newly introduced, the limited number of usage samples can degrade embedding quality and retrieval accuracy. Third, the multi-stage retrieval strategy, while more efficient than fully prompting LLMs for tool selection, still requires careful fine-tuning of multiple components (e.g., Tool2Vec, MLC, ToolRefiner), increasing system complexity. Additionally, although our approach scales better than large-model-based retrieval methods, maintaining separate fine-tuned retrievers for different domains may become resource-intensive when adapting to a broad array of specialized tool sets. Finally, the long-tail nature of tool usage—where certain tools are only needed under niche or highly specific queries—remains challenging. Future work may explore more robust methods to handle out-of-distribution queries and further reduce reliance on synthetic data, thereby enhancing both the adaptability and reliability of usage-based tool retrieval.

References

AI@Meta. 2024. Llama 3 model card.

Raviteja Anantha, Bortik Bandyopadhyay, Anirudh Kashi, Sayantan Mahinder, Andrew W Hill, and Srinivas Chappidi. 2023. Protip: Progressive tool retrieval improves planning. *arXiv preprint arXiv:2312.10332*.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2023. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*.

Yihan Cao, Yanbin Kang, Chi Wang, and Lichao Sun. 2023. Instruction mining: When data mining meets large language model finetuning. *Preprint*, arXiv:2307.06290.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. 2022. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397*.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. 2024. Alpagasus: Training a better alpaca with fewer data. In *The Twelfth International Conference on Learning Representations*.

Wei Chen and Zhiyuan Li. 2024. Octopus v2: On-device language model for super agent. *arXiv preprint arXiv:2404.01744*.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, et al. 2023. T-eval: Evaluating the tool utilization capability step by step. *arXiv preprint arXiv:2312.14033*.

Sumit Chopra, Raia Hadsell, and Yann LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 539–546. IEEE.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Yu Du, Fangyun Wei, and Hongyang Zhang. 2024. Anytool: Self-reflective, hierarchical agents for large-scale api calls. *arXiv preprint arXiv:2402.04253*.

Lutfi Eren Erdogan, Nicholas Lee, Siddharth Jha, Sehoon Kim, Ryan Tabrizi, Suhong Moon, Coleman Hooper, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2024. Tinyagent: Function calling at the edge. https://bair.berkeley.edu/blog/2024/05/29/tiny-agent/.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2024. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023. DeBERTav3: Improving deBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *The Eleventh International Conference on Learning Representations*.

Elad Hoffer and Nir Ailon. 2015. Deep metric learning using triplet network. In *Similarity-based pattern recognition: third international workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*, pages 84–92. Springer.

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

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*.

Siddharth Jha, Lutfi Eren Erdogan, Sehoon Kim, Kurt Keutzer, and Amir Gholami. 2024. Characterizing prompt compression methods for long context inference. *arXiv preprint arXiv:2407.08892*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024a. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024b. Mixtral of experts. *Preprint*, arXiv:2401.04088.

Armand Joulin, Laurens Van Der Maaten, Allan Jabri, and Nicolas Vasilache. 2016. Learning visual features from large weakly supervised data. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VII 14*, pages 67–84. Springer.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, et al. 2022. Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*.

Sehoon Kim, Coleman Hooper, Thanakul Wattanawong, Minwoo Kang, Ruohan Yan, Hasan Genc, Grace Dinh, Qijing Huang, Kurt Keutzer, Michael W Mahoney, et al. 2023. Full stack optimization of transformer inference: a survey. *arXiv preprint arXiv:2302.14017*.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. An llm compiler for parallel function calling. In *Forty-first International Conference on Machine Learning*.

Nicholas Lee, Thanakul Wattanawong, Sehoon Kim, Karttikeya Mangalam, Sheng Shen, Gopala Anumanchipali, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. 2024. Llm2llm: Boosting llms with novel iterative data enhancement.

Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens Van Der Maaten. 2018. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European conference on computer vision (ECCV)*, pages 181–196.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*.

OpenAI. 2023. Function calling and other api updates.

OpenAI. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Colt: Towards completeness-oriented tool retrieval for large language models. *Preprint*, arXiv:2405.16089.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Conference on Empirical Methods in Natural Language Processing*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

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.

Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B Tenenbaum, Leslie Kaelbling, and Michael Katz. 2024. Generalized planning in pddl domains with pretrained large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20256–20264.

Venkat Krishna Srinivasan, Zhen Dong, Banghua Zhu, Brian Yu, Damon Mosk-Aoyama, Kurt Keutzer, Jiantao Jiao, and Jian Zhang. 2023. Nexusraven: a commercially-permissive language model for function calling. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. 2024. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. *ArXiv*, abs/2212.03533.

Lai Wei, Zihao Jiang, Weiran Huang, and Lichao Sun. 2024. InstructionGPT-4: A 200-instruction paradigm for fine-tuning miniGPT-4.

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*.

Zeynep Akkalyoncu Yilmaz, Shengjin Wang, Wei Yang, Haotian Zhang, and Jimmy Lin. 2019. Applying bert to document retrieval with birch. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, pages 19–24.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Kan Ren, Dongsheng Li, and Deqing Yang. 2024. EASYTOOL: Enhancing LLM-based agents with concise tool instruction. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024a. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36.

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*.

A Additional Results

A.1 Ablation Studies

This section details the ablation studies using the ToolRefiner introduced in Section. 4.3 on the Toolbench dataset. First, we investigate the effectiveness of Tool2Vec embeddings compared to tool description embeddings and find that Tool2Vec consistently outperforms tool description embeddings. Then, we explore the impact of the number of initially retrieved candidate tools on overall retriever performance. We observe that increasing the number of candidate tools consistently enhances performance up to a certain point, after which the improvement plateaus and the retrieval metrics degrade.

A.1.1 Tool2Vec vs. Tool Description Embeddings

In this ablation study, we demonstrate the effectiveness of using Tool2Vec embeddings to the tool description embeddings for ToolRefiner. Specifically, we train two ToolRefiner models, one with Tool2Vec embeddings and one with the tool description embeddings, on Toolbench I3 dataset. We retrieve top-64 tools first by cosine similarites between Tool2Vec embeddings and query embeddings. We observe that ToolRefiner trained with Tool2Vec embeddings outperforms ToolRefiner trained with tool description embeddings across most retrieval settings. Our results are shown in Table 3.

A.1.2 Analysis on the Impact of the Number of Candidate Tools

In this set of experiments, we explore the impact of the number of candidate tools on the overall retrieval performance of ToolRefiner. Specifically, for each query, we retrieve the top-N tools either from the output of MLC or from cosine-similarity-based retrieval between the user query embedding and the Tool2Vec. Then, we fine-tune the pretrained DeBERTa-v3 xsmall model with these N tools. We vary the value of N across 8, 16, 32, 64, and 128 and evaluate performance on the Toolbench dataset.

In Table 4, one key observation is the initial improvement in performance as N increases. This trend is consistent across all datasets and retrieval methods, but the performance improvement plateaus after a certain N value, with peak performance achieved at N=32 or 64 configurations. Specifically, for Toolbench I1 and I2, the best-performing N value is 64 for both retrieval methods, while for Toolbench I3, the best-performing N value is 64 for the Tool2Vec-based retriever and 32 for the MLC-based retriever. This is because the retrieval performance improves as N increases. However, the performance of the ToolRefiner method decreases for large N across all datasets and retrieval methods, indicating that including too many candidate tools can overwhelm the language model and lead to confusion and suboptimal performance.

Moreover, comparing the performance of different retrieval methods, the MLC-based retriever consistently outperforms the Tool2Vec-based retriever for Toolbench I1 and I2 datasets across most of the top-N settings, while the Tool2Vec-based retriever outperforms the MLC-based retriever for the Toolbench I3 dataset. This suggests that the choice of retrieval method can significantly impact the performance of the ToolRefiner method, and the optimal N value may vary depending on the dataset and retrieval method used.

From these observations, we can conclude that it is critical to carefully select the appropriate N value when training a tool retriever. While lower N values enable faster inference, they may result in worse performance when dealing with a large number of tools. Conversely, including too many candidate tools can confuse ToolRefiner, leading to worse performance than the performance with smaller N. This indicates the importance of balancing the trade-off between performance and efficiency when designing a tool retriever for a given dataset.

A.2 ToolLens

We benchmark our methods against two baselines similar to Section 5. The results from Table 5 demonstrate that our proposed methods substantially improve over the description-based retrieval baseline on the ToolLens benchmark. Specifically, the description-based retriever scores an N@3 of 83.68 and R@3 of 83.45, which are substantially lower compared to both COLT and our approaches. By contrast, COLT achieves stronger performance for lower-rank retrieval (e.g., 93.65 in R@3) but lacks data for R@7.

Among our methods, ToolRefiner + MLC stands out with the highest R@3 and R@5 scores (95.84 and 98.73, respectively) and also delivers a strong result at R@7 (98.58). ToolRefiner + Tool2Vec performs comparably, surpassing COLT in most

Table 3: Performance comparison of ToolRefiner with Tool2Vec embeddings and tool description embeddings on ToolBench I3. The first row represents ToolRefiner fine-tuned with Tool2Vec tool embeddings using Tool2Vec-based retrieval, the second row represents ToolRefiner fine-tuned with description embeddings using Tool2Vec-based retrieval, and the third row represents ToolRefiner fine-tuned with description embeddings using description based retrieval. For each row, we fine-tune the E5-base embedding model specifically for each use case to compute the embeddings.

Embedding for ToolRefiner	**Retrieval Method**	**Recall @ 3**	**Recall @ 5**	**Recall @ 7**
Tool2Vec	Tool2Vec	80.58	87.80	89.70
Tool Description	Tool2Vec	71.55	82.27	87.28
Tool Description	Tool Description	66.00	74.60	76.55

Table 4: Comparison of ToolRefiner in Section. 4.3 performance on the Toolbench dataset across multiple top-N candidate tool configurations. We use an MLC-based retriever and a Tool2Vec-based retriever to retrieve a set of N candidate tools where N varies from 8 to 128. Our evaluation metric is Recall@K, where K are values 3, 5, and 7. The best-performing top-N configuration for each retriever method is highlighted in boldface.

Method	Top-N	**Toolbench I1**			**Toolbench I2**			**Toolbench I3**		
		R@3	R@5	R@7	R@3	R@5	R@7	R@3	R@5	R@7
MLC Retriever	8	91.18	95.17	96.33	81.96	87.54	88.21	70.53	82.90	86.63
	16	91.59	96.42	**97.08**	81.96	87.54	88.21	78.13	86.43	87.95
	32	91.43	96.25	**97.08**	81.67	87.33	88.58	**79.83**	**86.81**	**88.98**
	64	**91.84**	**96.83**	97.01	**82.89**	**87.92**	**88.96**	76.75	85.88	86.80
	128	90.67	96.25	96.67	80.17	87.17	89.12	77.08	85.72	87.98
Tool2Vec Retriever	8	87.01	93.79	95.00	75.25	81.25	82.75	74.00	87.38	89.22
	16	89.76	94.79	94.96	77.96	83.21	84.46	74.00	87.38	89.22
	32	**90.05**	94.46	95.25	76.88	83.17	84.33	79.50	87.77	89.53
	64	89.63	**95.33**	**96.17**	76.83	**84.42**	**86.38**	**80.58**	**87.80**	**89.70**
	128	87.84	94.87	95.42	**77.17**	82.42	83.87	78.17	87.55	89.30

Table 5: Comparison of tool retrieval results on ToolLens. We report NDCG as well as Recall.

Method	**ToolLens**					
	N@3	R@3	N@5	R@5	N@7	R@7
COLT	94.53	93.65	96.91	97.75	N/A	N/A
Description-Based Retriever	83.68	83.45	91.12	91.92	93.83	94.72
ToolRefiner + Tool2Vec	93.88	93.61	96.43	97.20	96.79	97.73
ToolRefiner + MLC	95.97	95.84	98.14	98.73	97.74	98.58

Table 6: Comparison of tool retrieval (R@3) and success rate (SR) for a description-based baseline versus our method.

Method	**R@3**	**SR**
Description-based Retrievers	0.941	0.759
MLC +ToolRefiner	0.972	0.827

metrics, but slightly trails behind the ToolRefiner + MLC variant. This consistent outperformance illustrates the advantages of multi-stage retrieval, where refined embeddings and classification-driven reranking work in tandem to improve overall accuracy. By leveraging usage-based embeddings and iterative refinement, our systems can more precisely capture the query-tool relationships, thereby boosting recall across multiple ranking thresholds.

A.3 TinyAgent Dataset

We conduct an end-to-end evaluation using the benchmark introduced in (Erdogan et al., 2024). This benchmark assesses tool learning performance for controlling macOS devices using Mac applications as tools. This benchmark evaluates not only tool retrieval accuracy but also the correctness of generated task graphs to measure end-to-end performance.

In the Table 6, we compare the success rate (i.e. correct execution graph construction) and tool retrieval performance of the baseline (description-based retrieval) against our MLC-based method. Our approach improves tool retrieval performance (R@3) by 3%, which directly translates to a significant 7% increase in the success rate. In summary, enhanced tool retrieval performance is critical in overall tool learning performance.

A.4 Analysis on Our Methods

In this section, we conduct a series of analyses to investigate why the ToolRefiner + Tool2Vec performs better than Tool2Vec across all ToolBank datasets. Particularly, we aim to pinpoint the specific tools that both methods struggle with, quantify the mistakes, and asses how query complexity affects tool retrieval performance. Our results show that ToolRefiner + Tool2Vec is better able to handle complex queries and maintain consistent performance across a diverse set of tools, while Tool2Vec struggles with simpler queries and makes errors on certain tools more frequently.

In our initial analysis, we aimed to identify the tools that ToolRefiner + Tool2Vec and Tool2Vec most frequently failed to retrieve. Specifically, the model fails to retrieve a tool when the tool is one of the ground truth tools but isn't retrieved. We then divided the number of these failures by the total occurrences of each tool in the dataset to calculate the percentage failure rate for each tool. For all experiments, we retrieved the top-5 tools, which is the maximum number of tools any data point in ToolBank needs to retrieve.

In Table 7, we show the distribution statistics of the percentage failure rates of all tools in ToolBank subsets. We observe that ToolRefiner + Tool2Vec demonstrates a more uniform failure rate distribution, with a relatively low mean and standard deviation. It rarely makes more than five errors per tool and averages 2.07 mistakes across all datasets. This suggests that ToolRefiner + Tool2Vec has a robust understanding of a broad range of tools, managing to maintain relatively low failure rates consistently.

On the other hand, Tool2Vec exhibits significant variability in its performance. Certain tools are prone to high failure rates, with some reaching up to 50 mistakes, while others have no errors at all. From Table 7, we observe that Tool2Vec's failure rate distribution is highly skewed, meaning that there are some tools that are responsible for a majority of Tool2Vec's errors. Furthermore, on average, Tool2Vec makes 7.86 mistakes per tool, indicating a less consistent performance across the board. This variability might be due to Tool2Vec's handling of tool embeddings, where it fails to adequately differentiate between tools with similar functionalities. In contrast, ToolRefiner + Tool2Vec effectively separates embeddings of tools with overlapping or similar use cases, which can be closely clustered in the Tool2Vec space. By diversifying these embeddings, ToolRefiner + Tool2Vec reduces the likelihood of confusion and errors, particularly in complex query scenarios.

Table 7: Comparison of the distribution of percentage failure rates of ToolRefiner + Tool2Vec and Tool2Vec across all tools in ToolBank. We calculate the percentage failure rate for a tool as the number of times the method fails to retrieve the tool divided by the number of times it was used in the entire dataset.

Data	**ToolRefiner + Tool2Vec**		**Tool2Vec**	
	Mean	Std.	Mean	Std.
BankNumpy	3.55	6.54	9.04	37.88
BankPandas	2.97	6.24	12.06	26.16
BankAWS	1.42	1.61	7.01	15.13

In our further analysis in Figure 6, we focus on the length of the queries where the methods fail and discover that ToolRefiner + Tool2Vec generally makes errors on longer queries, averaging nearly 20 tokens more than those where Tool2Vec failed. This finding implies that while ToolRefiner + Tool2Vec is equipped to handle more complex and lengthy queries, Tool2Vec tends to struggle with simpler, shorter queries. The ability of ToolRefiner + Tool2Vec to process longer and potentially more complex queries underscores its enhanced capability to manage intricate or verbose user requests effectively.

The disparities in percentage failure rate distribution and the correlation with query length suggest that ToolRefiner + Tool2Vec's superior performance can primarily be attributed to its refined handling of challenging queries and its robustness across a diverse set of tools.

A.5 Various Embedding Models

The experiments performed in section 5 rely on E5-base as an embedding model. To demonstrate the effectiveness of Tool2Vec compared to description-based retrieval, we show the results with other embedding models in Table 8. Tool2Vec consistently outperforms description-based retrieval across model families and sizes.

A.6 Resource Consumption

Table 9 reports the retrieval latency and memory usage of four methods on an NVIDIA A6000 with ToolBench I3: Tool2Vec, MLC, ToolRefiner, and a description-based reranker. We observe that Tool2Vec, MLC, and ToolRefiner are significantly more resource efficient than the description-based

Table 8: Comparison of Tool2Vec retrieval and description-based retrieval across various embedding models on ToolBench's I3 split. Models are evaluated without any fine-tuning. Tool2Vec consistently outperforms description-based retrieval on both open source and closed embedding models.

Method	R@3	R@5	R@7	R@10	R@12
E5-small + Tool2Vec	63.12	75.95	82.27	85.87	86.73
E5-small + Descriptions	20.62	30.45	37.27	42.42	46.87
E5-base + Tool2Vec	62.48	76.10	80.17	84.80	86.45
E5-base + Descriptions	32.12	38.97	43.92	50.63	54.80
E5-large + Tool2Vec	60.40	71.20	77.92	84.18	85.93
E5-large + Descriptions	33.28	42.12	48.45	56.48	60.25
Mxbai-embed-large + Tool2Vec	59.23	67.97	76.30	80.78	83.68
Mxbai-embed-large + Descriptions	40.03	48.25	53.58	60.73	64.93
Text-embedding-3-small + Tool2Vec	63.15	73.33	80.78	84.13	85.70
Text-embedding-3-small + Descriptions	41.12	54.47	57.65	65.67	68.78

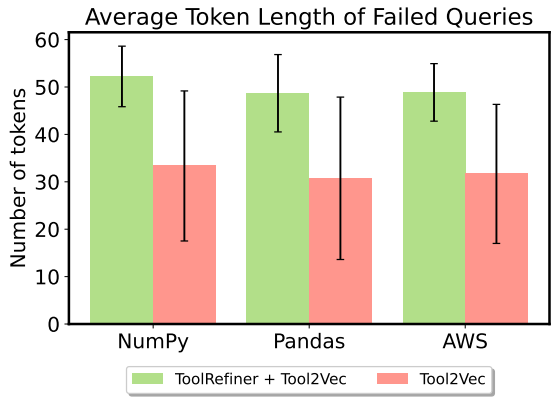

Figure 6: Illustration of average token length of failed queries for ToolRefiner + Tool2Vec combined with Tool2Vec and Tool2Vec on ToolBank is analyzed. We visualize the mean as a bar plot and the standard deviation as an error bar within each bar. We observe that Tool2Vec struggles with shorter and simpler queries, while ToolRefiner + Tool2Vec tends to make mistakes on longer and more complex queries.

reranker. For the second retrieval stage, ToolRefiner takes 21.64 ms and 0.88 GB when ranking 64 candidate tools, while the description-based reranker approach requires 337.64 ms and 6.17 GB for the same set of 64 candidates. This is because ToolRefiner is reranking tool embeddings, while the description-based reranker is reranking full tool descriptions.

Method	Avg Latency (ms)	Memory (GB)
Tool2Vec	0.21	0.48
MLC	36.45	0.70
ToolRefiner (64 tools)	21.64	0.88
Description-Reranker (bsz 64)	337.64	6.17

Table 9: Retrieval time and memory usage on ToolBench I3 with an NVIDIA A6000. Our methods consume significantly less compute and memory compared to a baseline of directly ranking tool descriptions with respect to their relevance to a search query.

B Details on Dataset Generation

B.1 Tool Selection Criteria

For tool collection, we crawl each library's official API reference and retrieve detailed information about function descriptions, arguments, and example code snippets. For NumPy, we exclude the numpy.ctypeslib, numpy.dtypes, numpy.emath, numpy.rec, and numpy.version modules since they don't provide rich functions or are outdated. For Pandas, we only use the public sub-packages and exclude the pandas.core, pandas.compat, and pandas.util modules. For Boto3, we include functions for five popular AWS services: EC2, RDS, IAM, S3, and SNS.

B.2 Parameters and LLMs to Generate Dataset

For dataset generation, we used $T = 10$ and $M = 2 - 5$, which means that at each iteration, we sample 10 tools from the tool pool and let the language model choose 2-5 tools to generate instructions. For both the Instruction Generation and Instruction Polish stages, we use Llama-3-70B-Instruct (AI@Meta, 2024).

B.3 Dataset Statistics

We collect 520 NumPy tools, 1600 Pandas tools, and 1000 Boto3 tools and curate 20,000 NumPy queries, 70,000 Pandas queries, and 73,000 AWS queries. There are 19530 tool combinations in our NumPy dataset, 69550 in Pandas dataset, and 70816 in AWS dataset. This means that almost all of our data represents distinct usage scenarios and queries.

B.4 Qualitative Analysis

B.4.1 Comparing Polished and Unpolished Queries

We provide a qualitative analysis of ToolBank and investigate the effect of the Query Polish step. We randomly sample three queries from ToolBank and present them before and after the polishing step in Figure 7. The left column in Figure 7 shows the queries before applying the Query Polish step, while the right column shows the queries after polishing. Before applying Query Polish, the queries exhibit a rigid and instructional style, similar to the Toolbench examples in Table 2. However, after applying Query Polish, the queries become more natural and user-friendly, better reflecting how real human users would interact with LLMs.

B.4.2 Qualitative Analysis on ToolBank

In this section, we compare BankNumpy, BankPandas, and BankAWS, which are subsets of ToolBank, and provide insights into why the tool retrieval performance on BankPandas is worse than on the other subsets. This performance degradation can largely be attributed to the similarity between operations in BankPandas. Specifically, BankPandas contains various data types like time series, periods, intervals, and indexes. For these data types, there are some common set of operations that apply to all of them such as `.equals`, `.argmin`, or `.all`. This results in instructions that are very close to each other in meaning but use different data types. Hence, the model gets confused about which data type to operate on. For example, the first part of "Load a delimited data file with specific columns and data types, counting the total number of entries, for the next fiscal quarter starting from the first business day of the year based on a given timestamp." query requires a call to `pandas.read_csv` tool which returns a `pandas.DataFrame` object and a subsequent call to `pandas.DataFrame.size` tool to count the total number of entries. However, ToolRefiner + Tool2Vec model makes the mistake of calling `pandas.Series.size` after calling `pandas.read_csv`. This doesn't become an issue with BankNumpy and BankAWS since BankNumpy tools operate almost entirely on the array type and BankAWS contains only the 5 most popular AWS services, creating a clear distinction between tools.

B.5 Prompt for Data Generation

B.5.1 Query Generation

The following prompt is used to generate user query for ToolBank.

B.5.2 Query Polish

The following prompt is used to polish user query for ToolBank.

Before Query Polish	After Query Polish
Determine the upper triangular elements of a square matrix, compute the average of these elements excluding zero values, and store the result in a compressed archive file.	Compute the average of the non-zero upper triangular elements of a square matrix and store the result in a compressed archive file.
Transform a daily weather dataset into a uniform array for analysis, ensuring correct time zone information, and then calculate the 25th percentile of temperature readings over the entire dataset while ignoring non-numeric entries.	Calculate the 25th percentile of temperature readings in a uniform daily weather dataset, ignoring non-numeric entries and ensuring correct time zone information.
Create a new IPAM scope, grant permission to attach a network interface to an instance, and modify the instance's placement attributes to use the new IPAM scope.	Create a new IPAM scope for attaching a network interface to an instance with modified placement attributes.

Figure 7: Qualitative analysis comparing the queries before and after the Query Polish stage. We randomly sample an example from each of BankNumpy, BankPandas, and BankAWS datasets.

```
You are an expert in utilizing a library of functions and generating diverse scenarios where a
set of selected functions are applied to solve real-world problems. You will be provided with
a set of functions and their descriptions, and will be tasked with selecting a subset of these
functions to craft detailed scenarios. You will generate clear and detailed user instructions,
list the names of the relevant functions, and explain how these functions can be applied to
complete the task. These tasks should demonstrate a wide range of functionalities and real-life
applications to ensure variety and utility.
Guidelines:
- The instructions must be clear and comprehensive, allowing users to understand how to apply the
functions without ambiguity. However, the instructions shouldn't be robotic and shouldn't sound
like 'step-by-step' instructions. For example, instead of writing "Calculate the non-negative
square root of an array element-wise, then round the resulting array to the nearest even value,
and return the indices that would sort the array along a specified axis." which breaks down each
step mechanically, you MUST instead write a more natural and fluid instruction like "Sort the
array along a specified axis after calculating the non-negative square root of each element and
rounding the result to the nearest even value.
- You MUST select and sequence the functions in a way that demonstrates their interdependency.
Ideally, a function's output should be the input to another function (or multiple functions),
creating a chain of operations that solve the task at hand. In other words, the functions you
select must not be selected randomly but instead be used to solve coherent multi-step problems.
- The explanations should logically connect the functions to the tasks, demonstrating the workflow
clearly.
- Your response should be returned as a single JSON object, representing a unique user instruction.
Diversity in function use and application context is crucial; avoid repetition of similar tasks
or functional applications to ensure a broad coverage of the capabilities of the functions.
Here is an example output of a list of JSON objects representing very distinct and detailed
tasks:
examples
You MUST only return a single JSON object - do not add any extra text before and after the json
object. The instructions that you generate MUST be very diverse and distinct from each other and
MUST be as different as possible from the examples above.
library - specific - instructions
```

Figure 8: Query Generation Prompt

```
You are an expert at refining user instructions to make them more coherent and less robotic. You
will be given a user instruction and will be tasked to refine the user instruction if it:
- Sounds too robotic or step-by-step like saying 'Do this, do that, and then do this'. In other
words, the instructions shouldn't break down each step mechanically but be more fluid.  For
example, instead of writing "Analyze the lyrics of the song 'XYZ', generate a playlist based on
the emotions and themes found, and create a Spotify playlist with the recommended songs." you
would write "Create a Spotify playlist based on the emotions and themes found in the lyrics of
the song 'XYZ'.
- Has conditional statements like 'if this, then do that' or 'when this happens, do that'. It
should be more direct and non-conditional.
If none of the above applies to an instruction, you should mark it as good, and provide a
reasoning for why it is good. Here example outputs of a JSON object representing a refined user
instruction:
```
$incontextexamples$

Figure 9: Query Polish Prompt