ToolDreamer: Instilling LLM Reasoning Into Tool Retrievers

Saptarshi Sengupta[1*], **Zhengyu Zhou**[2], **Jun Araki**[2], **Xingbo Wang**[2], **Bingqing Wang**[2], **Suhang Wang**[1], **Zhe Feng**[2],
{sks6765,szw494}@psu.edu
{zhengyu.zhou2,jun.araki,xingbo.wang,bingqing.wang,zhe.feng2}@us.bosch.com
[1]The Pennsylvania State University, [2]Bosch Research North America,

Abstract

Tool calling has become increasingly popular for Large Language Models (LLMs). However, for large tool sets, the resulting tokens would exceed the LLM's context window limit, making it impossible to include every tool. Hence, an external retriever is used to provide LLMs with the most relevant tools for a query. Existing retrieval models rank tools based on the similarity between a user query and a tool description (TD). This leads to suboptimal retrieval as user requests are often poorly aligned with the language of TD. To remedy the issue, we propose *ToolDreamer*, a framework that conditions retriever models to fetch tools based on *hypothetical (synthetic)* TD generated using an LLM, i.e., descriptions of tools that the LLM feels will be potentially useful for the query. The framework enables a more natural alignment between queries and tools within the language space of TD's. We apply *ToolDreamer* on the ToolRet dataset and show that our method improves the performance of sparse and dense retrievers with and without training, showcasing its flexibility. With our proposed framework, we aim to offload a portion of the reasoning burden to the retriever so that the LLM may effectively handle a large collection of tools without inundating its context window.

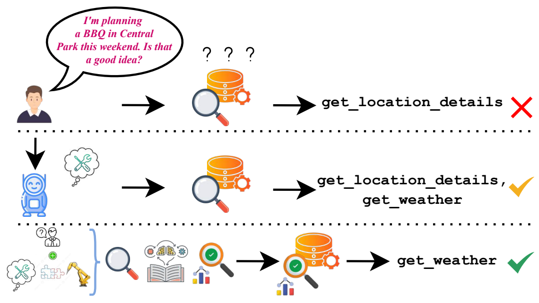
I'm planning a BBQ in Central Park this weekend. Is that a good idea?
? ? ?
get_location_details
get_location_details,
get_weather
get_weather

Figure 1: Illustration of the problem statement/motivation. Retrievers conditioned to learn query-tool relationships struggle to locate tools that are not readily obvious from the question (top). LLM-generated hypothetical tools can potentially be used to locate the right tool, but it still misses the mark as it has not been *conditioned* to learn such tool-to-tool relationships (middle). Thus, if we first *align* the hypothetical tool with its gold tool counterpart, and incorporate the query background to *train* the retriever, it would lead to a far superior model capable of accurately locating the correct tool (bottom).

1 Introduction

LLM-based *tool calling* (generating text as parsable function calls) (Masterman et al., 2024; Qu et al., 2025) has emerged as a viable solution to real-world environment interaction, allowing them to solve a variety of complex tasks from mathematics (Das et al., 2024) to molecular structure generation (Zhang et al., 2025c) by using tools. To enable tool calling, an LLM is provided with the JSON schema of each tool, consisting of function arguments, data types, and their purpose (Carrigan, 2024). Although tools are well-defined data structures, ultimately, for the LLM, they appear as additional tokens in its context window. When the number of tools is small, the entire set of tokens falls well within the model's processing limits. However, as the tool set expands, it puts pressure on the LLM, which ultimately leads to context overflow (out-of-memory (OOM)) (cf. App. B).

To address the OOM issue, LLMs are paired with external retrievers (Xu et al., 2025) which aim to select the right tool(s) for a query. Traditionally, tool retrievers are trained on triplets (question, gold tool, negative tool) by optimising the InfoNCE contrastive loss (van den Oord et al., 2019; Henderson et al., 2017) between the three components. This means that retrieval models learn to maximise the similarity between the questions and their corresponding gold (required) tools and minimise the similarity between the questions and negative (irrelevant) tools (Shi et al., 2025; Qu et al., 2024; Xu et al., 2024). However, we argue that this is subop-

[*]Work done during an internship at Bosch Research North America.

Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics Volume 1: Long Papers, pages 5465–5482
March 24-29, 2026 ©2026 Association for Computational Linguistics

timal for tool retrieval and illustrate this in Fig. 1 (top). In the given query, there is no obvious indication for weather information, which is the most relevant here, considering a BBQ is an outdoor event strongly influenced by weather conditions. Retriever models rely on semantic similarity and cues from the provided text. As such, they are unable to perform the necessary reasoning needed to select the correct tool, and consequently return an incorrect tool when using the query directly.

To alleviate the above issue, recently, *hypothetical tool-generation* has been proposed, i.e., leveraging LLM-reasoning to generate tools, potentially useful for a query. In this regard, Kachuee et al. (2025) propose the first study wherein they utilise an LLM's commonsense knowledge (Patil and Jadon, 2025) to generate useful tool descriptions for retrieval. However, the issue here is that the downstream retriever is *not optimised* for retrieval, as it does not undergo tool alignment training (learning to relate hypothetical tools to their gold counterpart). This defeats the purpose of generating such hypothetical tools, as the retriever is not conditioned to handle such hypothetical tool descriptions (Fig. 1 - middle).

Therefore, in this paper, we study a novel problem of retrieval optimization for retrieving tools based on hypothetical tools generated for a query. We see two challenges here: (i) data scarcity: considering the uniqueness of this problem, we do not find any existing datasets that enable training retrievers using question-hypothetical-gold tool samples, and (ii) knowledge infusion: determining the best way to integrate external knowledge as hypothetical tool descriptions into a retriever is a non-trivial task. To address these challenges, we propose a novel framework called *ToolDreamer*. With *ToolDreamer*, we first generate hypothetical tools for a question through LLM-prompting. These hypothetical tools are then *aligned*/mapped to their gold counterpart, using bipartite graph matching, to form the training target. The aligned tools and their associated questions form the dataset used to train the retriever model, addressing the first issue. There are various ways to represent the input hypothetical tools to the retriever. *ToolDreamer* couples questions with their aligned tools, by integrating LLM-generated hypothetical tool metadata, to train the retriever using an augmented InfoNCE loss that utilises both question and potential tools (§3.1.3), thus tackling the second issue (Fig. 1 - bottom).

Our **main contributions** are: (i) We study an important and novel problem: optimizing tool retrieval by investigating query-tool v/s hypothetical tool - gold tool mapping; (ii) We present *ToolDreamer*, a novel framework that can effectively retrieve tools based on hypothetical tools generated for a question; and (iii) Experimental results demonstrate the effectiveness of *ToolDreamer*.

2 Related Work

Dynamic Tool Creation. Recent years have seen the adoption of dynamic tool creation, i.e., using LLMs to synthesise new tools (Qian et al., 2023; Yuan et al., 2024; Cai et al., 2024; Wölflein et al., 2025). This not only removes the need to write code for an extensive number of tools, but also allows for on-demand tool creation, i.e., it is not necessary to determine the entire tool-set a priori. However, the major issue with these studies is the complexity and cost involved in crafting robust code, which requires several rounds of iteration and even multi-agent processes (Wölflein et al., 2025). On the other hand, **our framework does not task an LLM to generate the actual implementation of the tool**, but rather descriptions of tools that can potentially be used to solve that query, which is a much simpler proposition for them.

Tool Retriever Training. Current approaches for training retrievers (Shi et al., 2025; Qu et al., 2024; Xu et al., 2024) optimise a contrastive loss objective between (query, gold tool, negative tool) triplets, which we argue is sub-optimal due to the aforementioned reasons (misalignment between tool descriptions and user queries). Our framework improves on these methods by using a better aligned anchor term, i.e., instead of training on query-tool pairs, we train on tool-tool pairs.

Tool Retrieval With Auxiliary Information. Recently, there has been an uptick in methods for generating better search requests for tool retrieval. Chen et al. (2024b) propose a *training-free* framework where, instead of generating tools, they *generate queries* from tool descriptions. These queries are then appended to the tool description to form an augmented target for test-time questions. While the tool set now contains additional useful information, it comes in the form of queries, which, as we have shown (Fig. 1), is a suboptimal target for retrieval due to their nuanced nature. Braunschweiler et al. (2025) propose a *training-free ReAct* (Yao et al., 2023) agent that iteratively refines the query and tool set provided by the retriever. Similar to us, this framework uses an LLM to suggest potential tools for a query. However, they do not optimise their retriever using these potential tools, relying instead on several rounds of LLM reasoning, which can in turn accrue cost, particularly for API-based models. Kachuee et al. (2025) is the most similar to us in ideology. They leverage an LLM's commonsense knowledge to generate hypothetical tool descriptions for retrieval, and also train their LLM through SFT (Supervised Fine-Tuning) to generate high-quality tools. The key difference between our methods is that they train their LLM for tool-generation, using query-tool similarity (suboptimal as explained), and ignore retriever optimisation. Our proposed framework is inherently different from these works. As the retriever is not optimized for matching hypothetical tool descriptions to real tools, resulting in suboptimal retrieval performance, we propose a novel framework that fine-tunes the retriever and improves the performance with better retrieval query design.

Instructional Retrieval. A line of work related to the above is enhancing queries with additional instructions to guide retrieval. However, these works either use manually crafted (Weller et al., 2025; Sun et al., 2024) or LLM-generated instructions (Shi et al., 2025) to get retrieval directives. While more informative than the original query, these instructions may not match well with the tools space or fully decompose the query with tool description language, ending up with a suboptimal alignment in tool retrieval. In contrast, our approach (i) enforces the thought process of the LLM to better match the underlying meaning of the query with tools space (Fig. 1: the weather tool is required while the query does not mention weather at all.), and (ii) appropriately decomposes a query to cover cases requiring multiple tools to answer (Fig. 4).

3 The Proposed *ToolDreamer*

In this section, we detail the inner workings of our framework, starting with tool retriever training, followed by the inference phase. An overview of the framework is given in Figure 2.

3.1 Training Phase

Our objective is to obtain a better retriever that is trained to understand user queries and relationships between LLM-generated hypothetical tools and gold tools for the associated query. However, the main challenge is obtaining the training data for the retriever, as we do not have the ground truth hypothetical-to-gold tool matching for each query. To address the challenge, we propose a novel two-step approach for *training data generation*: given a question/query, an LLM generates potentially useful HT's. Next, we *align* the generated HT's with their gold counterpart using a bipartite graph matching algorithm to form the triplets (§1). Though the alignment between HT and GT might not be perfect, it still gives a rough and sometimes accurate preference/matching between HT and GT, making it possible to train the retriever. We *train our retriever* to assign high similarity between the aligned tools and low similarity between HT and randomly selected tools. This equips it with the ability to learn HT-GT relationships for a question and to perform a more targeted tool search.

3.1.1 Hypothetical Tool Generation

This is the first step of the training phase. Here, given a question, we prompt an LLM to generate tools it thinks could be relevant for solving the question (See Fig. 7 for Prompt Template). As we have access to the training data, we know a priori the number of tools required for the question. Thus, the LLM is instructed to generate hypothetical tools equal to the number of gold tools for the query. For the actual generation, we task the model to provide its thought process for the hypothetical tool, an appropriate name, and a description. The first two components are used because it has been shown (Yugeswardeenoo et al., 2024; Zhu et al., 2025) that LLM performance improves when providing a reasoning breakdown before the final answer. Additionally, as mentioned, we do not generate an implementation of the tool, but only metadata (thoughts + name + description).

Although LLMs have gotten better over the years, they are still prone to making mistakes and not following instructions (Tong et al., 2024; Chen et al., 2024a). We notice a similar phenomenon in our study. While our LLM (§4) can generate the right number of tools for the majority of questions, it falters for a very small fraction (0.3%). These questions are subsequently removed from the sampled dataset.

3.1.2 Tool Alignment

After generating each question's HT's, we align them with their gold counterpart. This is done to form the final training samples, where each question is augmented with its HT along with the target

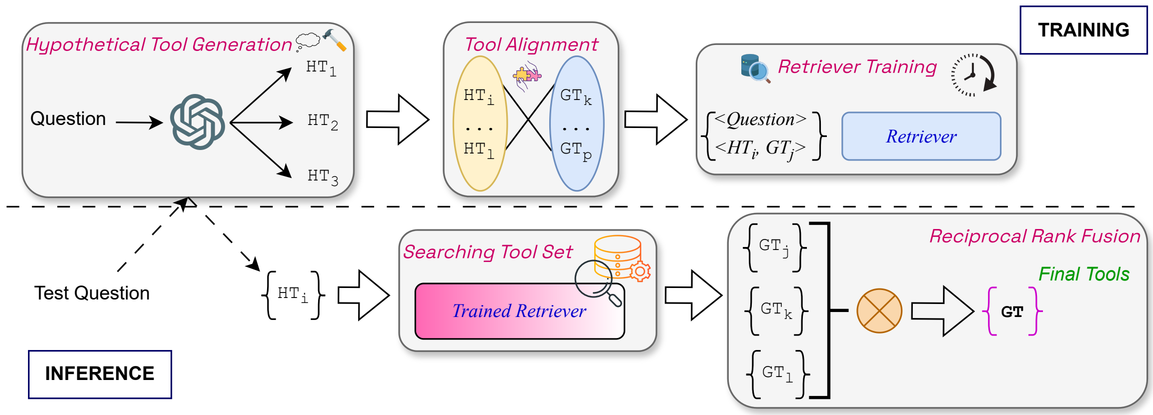

Figure 2: *ToolDreamer* overview. **Training Phase (Top)**: A strong LLM first generates hypothetical tools (HTs) for a question. These tools are then aligned with their gold tool (GT) counterparts. Finally, the aligned tool pairs are used to train the retrieval model. **Inference Phase (Bottom)**: At test time, an unseen question is provided to the same LLM, which generates a set of hypothetical tools for it. For each tool, the trained retriever collects a top-K list of tools from the database. A unified list is obtained through reciprocal rank fusion (RRF).

value (corresponding GT). First, we treat the hypothetical and gold tool sets as a bi-partite graph, i.e., a graph whose vertices can be partitioned into two distinct sets and edges always connect vertices from corresponding sets. We justify this assumption by noticing that the tools in each set belong to a distinct category (Hypothetical Tool (HT) v/s Gold Tool (GT)) and hence show no overlap apart from being related by a common theme, i.e., they are all tools. Thus, "connecting" tools can be viewed as a graph matching problem, i.e., finding a set of edges between the nodes such that no two edges have a common vertex. Simply put, obtaining edges between a unique pair of nodes.

We first compute a semantic similarity (using Qwen3-8B (Zhang et al., 2025b) embedding) matrix for each (hypothetical tool, gold tool) pair. From this matrix, we perform minimal cost matching using the **Hungarian algorithm**[1] (Kuhn, 1955). The algorithm solves the problem: given a task-cost matrix, what is the optimal way of assigning tasks to minimise overall cost? The assignment is done to create a **best possible** 1:1 mapping between elements. We leverage the same method to optimally connect tools based on their similarity and create our set of aligned (HT, GT) pairs.

[1]The Hungarian algorithm has a time complexity of $O(n^3)$ where n is the number of elements (rows/columns of the cost matrix) to align. This can potentially introduce significant computation overhead into the framework. However, this holds only if the cost matrix to optimize is significantly large. In `ToolRet`, the number of tools required for a question ranges between 1 (56% of questions) and 8 (0.1%), with the average being 2. As such, their resultant matrices do not take considerable time to process.

A natural concern here is *misalignment* (§4.3.2) (poor HT-GT match). As we deal with a square semantic similarity matrix, there will *always* be a match. However, it is a *best possible attempt* by the underlying matching algorithm and embedding model used to score HT-GT similarities. Noise can be greatly mitigated by using a strong embedding model and a reliable alignment method. As mentioned previously (§3.1), HT-GT matching might not be perfect. However, this imperfect mapping acts as a rough and often accurate proxy for the true alignment, making it possible to train the retriever. The ultimate goal is to train the retriever to assign high similarity between the aligned tools and low similarity between HT and random tools. We do this to make it perform a more targeted tool search, using the learned HT-GT relationships.

3.1.3 Retriever Training

A retriever that has not been conditioned to learn HT-GT relationships will offer reduced benefits when using HT as a search vector (Fig. 1). As such, it needs to be trained to identify the most relevant gold tool for a given question and its HT's. To train our retriever, we use the assembled training dataset, i.e., questions + aligned HT-GT. We optimise our retriever using InfoNCE loss. The basic idea is to teach the retriever to push the HT towards its GT, in embedding space, while simultaneously pushing the HT away from negative (irrelevant) tools.

Each HT is associated with three components, i.e., the thought process of the LLM behind suggesting that tool, the tool name, and its description. There are different ways to integrate this information (§1) into the retriever for selecting gold tools. We could use the HT name and description only as input to the retriever. However, the LLM-thoughts also contain important *reasoning information* for suggesting the necessity of the generated HT, which provides an additional signal to the retriever. Thus, to take advantage of the entire metadata, we format our input to the retriever as,

```
Thoughts:{} Tool Name:{} Tool Description:{}
```

We denote this input style as TND (Thought-Name-Description). As the query also provides an important signal on what tools to retrieve, we formulate another input style where we prepend the query (Q) to TND, denoting it as QTND.

Overall, our final loss function is given in Eq. 1. A = TND / QTND is the *anchor*, i.e., the reference for which the model has to be optimised; GT = Gold Tool; n_i is the i^{th} negative/irrelevant tool for a training sample, and *sim* = semantic similarity function, usually cosine similarity. $[\cdot]$ represents the embedding for the corresponding term. We provide training/hardware details in App. C.

$$-\log \frac{e^{sim([A],[GT])}}{e^{sim([A],[GT])} + \sum_{i=1}^{k} e^{sim([A],[n_i])}} \tag{1}$$

3.2 Inference Phase

Our inference phase also proceeds in three steps. At first, we generate HT for the test data, similar to the training phase. Next, we use our trained retriever model to perform tool-retrieval using the improved search objective (question + HT). Finally, each retrieved top-K tool list, corresponding to each HT, is merged to form a unified result.

3.2.1 Hypothetical Tool Generation

This step is similar to the training step, where an LLM (§4) is asked (Fig. 8) to generate hypothetical tools for the given query. During inference, we do not know a priori how many or what tools to retrieve. Thus, we do not tell the LLM how many hypothetical tools to generate. As such, it is tasked with generating as many tools as it feels are relevant for the query in an open-ended manner.

3.2.2 Tool Retrieval

As explained in the training phase, we investigate two retrieval settings, viz., using TND (HT Thought-Name-Description) and QTND (Query + HT-TND). For each strategy, the respective trained retriever returns a top-K list of tools, using the search vectors (HT's) for a question.

An LLM is not always able to follow instructions correctly. Consequently, for a very small fraction of questions, it does not generate an equal number of thoughts, names, and descriptions. As we require each HT to be associated with this metadata to form the final search vectors, to handle these instances, we ignore the HT generations and fall back on the base question as the search vector. Despite repeated prompt changes, the LLM was unable to generate the correct number of components for *all* questions, i.e., it was found to always falter for *some* questions. However, this is not a major issue as the number of such samples is not large enough to introduce significant performance degradation.

3.2.3 Retrieval Unification

For a given question, there are multiple search vectors, based on the number of generated hypothetical tools, formatted according to a particular strategy (TND/QTND). Thus, the retriever returns multiple top-K tool lists, one for each vector. As not all tools in these top-K lists are useful, we need to filter out non-important ones and unify the useful ones into a single list for the LLM. One straightforward way is to select the top-1 tool from each retrieved list. However, as these lists are generated for the same question, they might have dependencies and overlaps that such simple methods do not take into consideration. Thus, to combine each top-K retrieved tool list into a single one, we propose using **Reciprocal Rank Fusion** (**RRF**) (Cormack et al., 2009). RRF is a widely adopted technique (Samuel et al., 2025; Kuo et al., 2025; Kassaie et al., 2025) to unify multiple ranked lists. It works on the principle of creating a ranked list that takes into account the relative position of items across each list. In other words, if an item in each list is consistently ranked high, it will be reflected in the final list also.

RRF is traditionally used to unify ranked lists from different retrievers for the same query. However, in our case, we use different search vectors with corresponding retrieved lists for unification. This is thus beyond the original scope of RRF. However, we view the different search vectors as being connected to the *same input query*. As such, we loosely interpret RRF for our framework as simply an approach to unify multiple lists that are related to the original query. Moreover, using methods such as selecting the top-1 tool from each retrieved list would yield a total number of tools, the same as the number of generated HT's, which would be inconsistent with evaluation (§4). This concludes the inference phase of *ToolDreamer*.

4 Experiments

In this section, we conduct experiments to evaluate the effectiveness of the proposed framework. We aim to answer four research questions (RQ): (i) the benefit/performance gains obtained by using our framework over current approaches; (ii) the relative importance of each component of our framework; (iii) whether our method is flexible, i.e., independent of the choice of underlying models and (iv) additional considerations for usage in production.

4.1 Experiment Settings

Dataset. We evaluate *ToolDreamer* on `ToolRet` (Shi et al., 2025), a comprehensive benchmark that combines 26 (35 with subsets) existing tool-calling datasets such as UltraTool (Huang et al., 2024) and Gorilla (Patil et al., 2024). `ToolRet` has three high-level subsets. (i) **Web**: APIs in OpenAI JSON format to perform various database or file operations, e.g., `check_account_functionality` (*Verify the functionality of the account to confirm login status*); (ii) **Code**: Functions related to ML workflows, etc., e.g., `identify_visible_attributes` (*Identify the visible attributes of an object in an image*); (iii) **Customized**: Tools to enable a broad range of daily tasks, e.g., `convertCurrency` (*Convert currency values from one currency to another*). After filtration (App. E), web has ~37K tools/~5K queries; code has ~4K tools/~2K queries; customised has > ~3K tools/~1K queries. The training split of `ToolRet` consists of ~200K examples from which we sample 5K instances to optimise for API cost and data quality (App. E).

Evaluation Metrics. We evaluate retriever quality using 4 standard IR (Information Retrieval) metrics. These are (i) **NDCG@K/N@K** (Normalized Discounted Cumulative Gain): evaluates the *quality* of a ranked list by rewarding relevant items being placed at the top of the list and penalizing them being placed towards the bottom; (ii) **P@K** (Precision): The fraction of retrieved items that are relevant to the search; (iii) **R@K** (Recall): The fraction of relevant items that were retrieved, and (iv) **MRR** (Mean Reciprocal Rank): Inverse rank of the first relevant item in the retrieved list. For each measure, `@K` means the number of items that are retrieved. Following `ToolRet`, we set K to 10.

Baselines. In §2, we mention various related approaches for tool-retriever training. However, Xu et al. (2024) does not have publicly available code. As such, we use (i) Shi et al. (2025) (`ToolRet` (**TR**)), who train their models with both query-tool and (query + instruction) - tool similarity. However, we intentionally avoid incorporating the instructions to ensure a consistent evaluation with existing benchmarks which lack such annotations as our goal is to evaluate *ToolDreamer* under the minimal assumptions that broadly apply across tool datasets; (ii) `COLT` (Qu et al., 2024), a complex two-step framework that first tunes a model (`Contriever` (Izacard et al., 2022)) using standard InfoNCE with query-tool similarity, followed by further optimisation with a graph neural network (GNN) using various decompositions of the training dataset; (iii) use HT directly for retrieval similar to Kachuee et al. (2025) (**K** in Table 1).

HT Generation Model. We use GPT-4.1 (OpenAI, 2025) to generate hypothetical tools for both training and test data queries. Additionally, we show that *ToolDreamer* can also use open-source models for HT generation (§4.4). Thus, it is not restricted by the choice of LLM used.

4.2 *ToolDreamer* Performance Analysis (RQ1)

To evaluate the tool retrieval performance of *ToolDreamer*, we choose a dense Qwen3-8B and a sparse BM25 model (App. A). Table 1 shows the results of *ToolDreamer* against related baselines.

We evaluate our framework under training (using aligned tools) and no-training (zero-shot) settings. For the latter, it means directly feeding the search vectors (of the test set) to the models, using `TND/QTND` format, to gauge how well they work out-of-the-box with an optimised search component. As we can see from Table 1 (rows 1-6), *ToolDreamer* offers strong improvements for both models, ranging between 8% (Qwen3) and 19% (BM25) over their zero-shot results. This highlights the effectiveness of our method, and particularly the quality of the generated hypothetical tools, which help the models acquire better cues on what tools to retrieve. We notice that while the plain `TND` setting works well, greater improvements are generally reported by including the question in the search vector. This reaffirms our hypothesis that hypothetical tools alone are insufficient for retrieval (Fig. 1). While they provide an important signal, they benefit from coupling with the user query.

The last three rows of Table 1 show the impact

Train?	Split	Web				Code				Customized				Avg.			
	Model	N@10	P@10	R@10	MRR	N@10	P@10	R@10	MRR	N@10	P@10	R@10	MRR	N@10	P@10	R@10	MRR
	BM25 (0s)	27.25	5.72	36.25	28.10	29.50	4.90	37.38	29.83	31.90	8.70	36.50	39.63	29.55	6.44	36.71	32.52
×	BM25 (TND) (**K**)	28.39	6.31	39.85	28.29	33.91	6.47	47.15	32.25	27.61	8.41	35.34	31.31	29.97	7.06	40.78	30.62
	BM25 (QTND)	31.41	6.82	42.83	31.72	36.50	6.51	49.34	34.81	33.48	9.13	38.88	40.72	**33.80**$^{+14\%}$	**7.49**$^{+16\%}$	**43.68**$^{+19\%}$	**35.75**$^{+10\%}$
	Qwen3 (0s)	35.81	7.20	47.09	36.64	39.58	6.54	51.32	38.53	39.89	10.65	47.70	46.14	38.43	8.13	48.70	40.44
×	Qwen3 (TND) (**K**)	35.94	7.93	50.15	35.5	42.44	7.86	58.35	40.05	38.12	10.53	49.14	41.82	38.83	**8.77**$^{+8\%}$	**52.55**$^{+8\%}$	39.12
	Qwen3 (QTND)	37	7.55	48.93	37.68	45.3	7.76	58.21	43.90	41.56	10.82	50.04	47.47	**41.29**$^{+7\%}$	8.71	52.39	**43.02**$^{+6\%}$
	COLT (Phase-1)	33.27	6.06	38.8	36.15	8.66	1.37	11.87	7.92	23.41	6.21	26.67	29.55	21.78	4.55	25.78	24.54
	COLT (Phase-2)	0.2	-	0.21	-	0.17	-	0.31	-	0.42	-	0.56	-	0.26	-	0.36	-
	Qwen3 (Q) (**TR**)	38.15	7.80	49.91	39.07	39.29	6.60	50.84	38.28	42.17	11.30	49.80	48.86	39.87	8.57	50.18	42.07
✓	Qwen3 (TND)	37.27	8.2	51.23	36.98	42.39	8.03	58.27	39.65	39.71	11.25	50.87	43.49	39.79	9.16	53.46	40.04
	Qwen3 (QTND)	39.10	8.28	51.99	39.50	45.93	8.18	59.51	44.23	42.38	11.55	51.27	47.90	**42.47**$^{+7\%}$	**9.34**$^{+9\%}$	**54.26**$^{+8\%}$	**43.88**$^{+4\%}$

Table 1: Performance of BM25 and Qwen3 using *ToolDreamer*. Best scores are in bold. Percentage improvements in teal w.r.t either zero-shot (using questions only) performance (testing only) or when training on question-tool mapping (Qwen3). Q = Trained on question-tool mapping (`ToolRet` (**TR**) baseline); TND = Thought + Name + Descriptions; QTND = Question + TND; 0s = zero-shot. The training script provided by Qu et al. (2024) for phase-2 of COLT only reports R@10/N@10. For each split, we only use their associated tools when performing retrieval. Averages are computed using each split's performance.

of **training** Qwen with only questions (baseline from Shi et al. (2025)) v/s our aligned tools. Again, we see the benefit of our method in offering improvements. These results highlight two points: (i) using questions alone to train retrievers is suboptimal, and (ii) when augmented with LLM reasoning chains, retrievers can be trained to better hone in on the required tools. It should be noted here that we use slightly fewer samples to achieve these scores as compared to training on questions alone. This is because, as mentioned in sec. 3.1.1, we remove those questions from our sampled training dataset for which GPT is unable to generate the requested number of tools. This observation further showcases the quality of our generated tools and the sample efficiency of our framework. We provide an example of generated tools in App. F.

Next, we see how strong our framework is against `COLT`, another related baseline. Firstly, the results from phase-1 training (general query-tool alignment) are far superior to phase-2 (GNN-optimisation). This makes sense as they train their model using the standard query-tool InfoNCE objective, which works relatively well. However, the increased complexity of grouping tools, aligning them with queries, and using a GNN for further optimisation plummets performance. In this regard, *ToolDreamer* is a much simpler and scalable approach capable of handling a variety of questions.

Finally, we would like to clarify that in this work, we intentionally exclude the instruction field of `ToolRet` dataset with the aim of developing/evaluating *ToolDreamer* without assuming any a priori instruction directives. Note that most existing tool-calling datasets lack instruction field annotations and annotating query instructions for new real-world use cases can be resource-intensive (e.g., requiring human experts to manually craft seed instructions as in `ToolRet`). Without dependency on pre-exisitng instructions, *ToolDreamer* can be readliy applied to other datasets as well as to new tool-calling use cases.

4.3 Ablation Studies (RQ2)

We conduct various ablation studies to investigate the importance of each step of *ToolDreamer*: (i) the impact of HT quality (§4.3.1) (ii) using a weak discriminator for tool alignment along with the alignment algorithm itself (§4.3.2), and (iii) whether using an LLM for ranked list fusion offers additional benefits or not (§4.3.3).

4.3.1 Hypothetical Tool Quality

The foundation of *ToolDreamer* starts with the reasoning ability of an LLM, and the subsequent quality of generated HT's. Thus, it is expected that *higher quality HT's* equate to *better downstream performance*. To investigate this, we tasked GPT-4.1 to generate *inferior* quality tools by removing key requirements from the original prompt, using looser language and not providing any guiding examples (Fig. 5). By deliberately redacting guidance, we can **gauge the importance of writing well-crafted prompts** to generate higher quality HT's. Results from this test are shown in Table 2. As we can see, each model suffers from performance degradation when using inferior quality tools. Although a decrease of 2% might not seem substantial, recent studies such as Yazan et al.

Model	Tools	NDCG@10	P@10	R@10	MRR
BM25	Original	**29.97**	**7.06**	**40.78**	30.62
	Ablation	29.55$^{-1\%}$	6.77$^{-4\%}$	39.66$^{-3\%}$	**30.76**
Qwen3	Original	**38.83**	**8.77**	**52.55**	39.12
	Ablation	38.23$^{-2\%}$	8.33$^{-5\%}$	50.58$^{-4\%}$	**39.21**

Table 2: Impact of using poor-quality tools on performance. Best scores are in bold. Scores shown are averages from each split. Each setting uses the TND format to isolate the tool quality impact more effectively.

Alignment Method	N@10	P@10	R@10	MRR
Original	**39.79**	**9.16**	**53.46**	**40.04**
DPR/Hungarian	39.48$^{-1\%}$	9.10$^{-.7\%}$	53.0$^{-.9\%}$	39.81$^{-.6\%}$
Qwen/Greedy	39.70$^{-.2\%}$	9.15$^{-.1\%}$	53.22$^{-.4\%}$	40.0$^{-.1\%}$

Table 3: Impact of different alignment methods (embedding/matching algorithm) on training Qwen3. Base scores are provided for easier comparison. Each setting uses the TND format to isolate the tool-alignment quality impact more effectively.

(2024) indicate that even as little as two irrelevant results can have a catastrophic impact on the LLM's performance. As such, when using our framework, it is essential to generate quality HT's for augmenting the search vectors.

4.3.2 Weak Tool Alignment

The second training step of our framework is tool alignment, where we create pairs of (hypothetical, gold) tools, based on their semantic similarity and assignment using a graph matching algorithm. This step is essential as poor HT-GT alignment can hamper the retriever's ability to learn meaningful relationships between the tools and the related question. As such, we examine two aspects: (i) quality of the embedding model used to create tool representations and (ii) graph matching algorithm, i.e., the method used to align the generated tool embeddings. These tests reveal the impact of using quality embeddings for alignment - highly similar representations for dissimilar tools (poor embedding model) can cause the alignment method to make incorrect assignments. Additionally, a weaker alignment technique can get stuck in a local optimum and ignore the global best assignment. To test each component, we (i) swap out Qwen3 embeddings for the far inferior DPR model (Karpukhin et al., 2020) but use the Hungarian algorithm and (ii) use Qwen3 embeddings with a *greedy* matching where at each step, it chooses the tool pair with the highest similarity. This method of selection is similar to greedy decoding in LLMs (Song et al., 2025).

Results from this test are shown in Table 3. Interestingly, while there is technically a decrease, it is not as strong as using poor quality tools (§4.3.1). This indicates that while using a good embedding model/alignment method is necessary, even more so is having high-quality tools. These results suggest that even if there exists HT-GT misalignment, the very presence of the proposed external signal (HT) is enough to enhance retriever performance.

4.3.3 LLM List Fusion

Using LLMs to rank/rerank a top-K list of items (Gao et al., 2025; Gangi Reddy et al., 2024; Zhang et al., 2025a) has emerged as a novel solution to existing fusion mechanisms due to their superior reasoning capabilities. As such, we wanted to see how swapping out RRF for LLM-based fusion impacts overall retrieval performance.

Here, we provide the set of retrieved tools for a question and ask (Fig. 6) the LLM (GPT-4.1) to rank the top-10 in decreasing order of relevance with a score between 0 (least likely) and 1 (highly relevant). There are two cases to deal with: (i) when the number of retrieved tools = 10 (either using the question as the search vector or 1 HT) - the LLM is asked to rerank it based on query relevance, and (ii) number of tools > 10 (multiple HT's) - the LLM selects the top-10 among the entire set.

Table 4 shows the results of this test. As seen, LLM reranking provides additional improvements over RRF. This makes sense as RRF is a simple statistical technique that simply considers the overall ranking of elements across lists, whereas an LLM does deeper reasoning to figure out which tools actually make sense for a query. However, these results come with caveats such as additional API costs, ignoring instructions, and hallucinations (spurious generations) (Huang et al., 2025), which we elaborate in App. D. Considering these issues, we prioritise RRF over LLM-ranking in *ToolDreamer* as it offers reproducible and deterministic guarantees. That said, it is easy to equip *ToolDreamer* with LLM-ranking, should cost/instruction-refusal not be a pressing concern.

4.4 Varying HT Generator (RQ3)

In our experiments, we use GPT-4.1 to create our HT's. However, this might be a bottleneck in cases where API-calling might be an issue, either due to cost or privacy concerns (Yao et al., 2024). Thus, we wanted to see how switching out GPT-4.1 for

Approach	NDCG@10	P@10	R@10	MRR
TND/RRF	39.79	9.16	53.46	40.04
QTND/RRF	42.47	9.34	54.26	43.88
TND/LLM-F	46.48	**10.16**$^{+9\%}$	57.64	46.83
QTND/LLM-F	**46.53**$^{+10\%}$	10.14	**57.67**$^{+6\%}$	**46.89**$^{+7\%}$

Table 4: Impact of using different ranked-list fusion methods. LLM-F = LLM (GPT-4.1) Fusion. All scores are from the trained Qwen3 model using TND/QTND vectors. Improvements are shown against the best baseline (QTND).

Model	Setting	N@10	P@10	R@10	MRR
BM25	TND / GPT	33.91	6.47	47.15	32.25
	QTND / GPT	36.50	6.51	49.34	34.81
	TND / Qwen	31.43	5.52	42.68	30.19
	QTND / Qwen	35.27	6.00	47.47	33.68
NV-Emb	TND / GPT	44.52	8.22	60.31	42.61
	QTND / GPT	47.41	8.07	60.76	46.53
	TND / Qwen	43.13	7.35	56.57	42.42
	QTND / Qwen	46.11	7.56	59.16	45.31

Table 5: Impact of using different Hypothetical Tool Generators. Setting = Search Vector / HT Generator.

an open-source model impacted performance. To this end, we use Qwen3-32B (with the same prompt (Fig. 8) as GPT-4.1) as the HT generator and BM25 and NVIDIA `NV-Embed-2` (Lee et al., 2025) as the retrievers. Each model is evaluated with inference mode (no training) using both `TND/QTND` settings. A different dense retriever is used here to gauge the universality of our approach. We use the code split of `ToolRet` for these tests. The results are presented in Table 5. As we see, *ToolDreamer* is flexible enough to work with different LLMs without much loss of performance. Thus, there is no necessity to stick to API models for HT generation, which shows the cost-benefit of using *ToolDreamer*.

4.5 Production Considerations (RQ4)

As we report our main scores using an API-based model, it is necessary to understand the associated costs, as higher monetary requirements can make the framework unsuitable for real-time production demands. Implementing *ToolDreamer* cost us less than $5 using OpenAI's batch mode and the cheaper GPT-4.1. Thus, we can claim that our framework is highly cost-effective.

To gauge wall-time, we consider the query *Can you tell me about the historical events of April 21st?* from the dataset. Basic retrieval (Qwen) takes ~0.04 seconds. With *ToolDreamer*, it takes us ~8 seconds to generate the hypothetical tools using Qwen3-32B (on 1 A100 card) and ~2.5 seconds with GPT. Semantic search + RRF takes an additional 0.5 seconds, bringing the total to ~3 - 8.5 seconds. While this does introduce some latency, it is unlikely that in production, clients will constantly be asking questions that require tool usage, which helps justify the time constraints.

5 Conclusion

In this paper, we present *ToolDreamer*, a flexible tool-retrieval framework that proposes a new lens for search, i.e., conditioning retrievers on *hypothetical tools* potentially useful for a query, instead of the query itself, for a more natural search alignment. Through extensive experiments, we highlight the benefits of our method over models that rely on traditional query-tool similarity. Having a better retriever is closely linked with LLM tool-calling ability. *ToolDreamer* is a step in this direction to provide LLMs with scoped tool sets connected to user queries. Future work will explore alternative loss objectives and tool alignment algorithms to further improve retriever efficiency.

Limitations

We identify two limitations with our framework and offer potential workarounds. First, *ToolDreamer* requires *high-quality hypothetical tools* to work well (§4.3.1). This requires a few rounds of testing to standardise the prompt, by experimenting with format, instructions, and examples. However, considering the importance of prompt quality (Long et al., 2025) for most generative tasks, we find this requirement reasonable for *ToolDreamer*. Second, compared to regular retrieval, RRF adds another layer of processing to increase the evaluation time complexity. However, from our evaluations (§4.5), we did not find this to be a significant bottleneck considering the capability of modern hardware.

References

Norbert Braunschweiler, Rama Doddipatla, and Tudor-catalin Zorila. 2025. ToolReAGt: Tool retrieval for LLM-based complex task solution via retrieval augmented generation. In *Proceedings of the 3rd Workshop on Towards Knowledgeable Foundation Models (KnowFM)*, pages 75–83, Vienna, Austria. Association for Computational Linguistics.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2024. Large language models as tool makers. In *The Twelfth International Conference on Learning Representations*.

Matthew Carrigan. 2024. Tool Use, Unified — huggingface.co. https://huggingface.co/blog/unified-tool-use.

QiHong Chen, Jiachen Yu, Jiawei Li, Jiecheng Deng, Justin Tian Jin Chen, and Iftekhar Ahmed. 2024a. A deep dive into large language model code generation mistakes: What and why? *Computing Research Repository*, arXiv:2411.01414. Version 2.

Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammadhossein Bateni, Chen-Yu Lee, and Tomas Pfister. 2024b. Re-invoke: Tool invocation rewriting for zero-shot tool retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 4705–4726, Miami, Florida, USA. Association for Computational Linguistics.

Jooyoung Choi, Hyun Kim, Hansol Jang, Changwook Jun, Kyunghoon Bae, Hyewon Choi, Stanley Jungkyu Choi, Honglak Lee, and Chulmin Yun. 2025. Lgai-embedding-preview technical report. *ArXiv*, abs/2506.07438.

Gordon V. Cormack, Charles L A Clarke, and Stefan Buettcher. 2009. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, page 758–759, New York, NY, USA. Association for Computing Machinery.

Debrup Das, Debopriyo Banerjee, Somak Aditya, and Ashish Kulkarni. 2024. MATHSENSEI: A tool-augmented large language model for mathematical reasoning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 942–966, Mexico City, Mexico. Association for Computational Linguistics.

Kenneth Enevoldsen, Isaac Chung, Imene Kerboua, Márton Kardos, Ashwin Mathur, David Stap, Jay Gala, Wissam Siblini, Dominik Krzemiński, Genta Indra Winata, Saba Sturua, Saiteja Utpala, Mathieu Ciancone, Marion Schaeffer, Diganta Misra, Shreeya Dhakal, Jonathan Rystrøm, Roman Solomatin, Ömer Veysel Çağatan, and 63 others. 2025. MMTEB: Massive multilingual text embedding benchmark. In *The Thirteenth International Conference on Learning Representations*.

Revanth Gangi Reddy, JaeHyeok Doo, Yifei Xu, Md Arafat Sultan, Deevya Swain, Avirup Sil, and Heng Ji. 2024. FIRST: Faster improved listwise reranking with single token decoding. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8642–8652, Miami, Florida, USA. Association for Computational Linguistics.

Jingtong Gao, Bo Chen, Xiangyu Zhao, Weiwen Liu, Xiangyang Li, Yichao Wang, Wanyu Wang, Huifeng Guo, and Ruiming Tang. 2025. Llm4rerank: Llm-based auto-reranking framework for recommendations. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, page 228–239, New York, NY, USA. Association for Computing Machinery.

Yilin Geng, Haonan Li, Honglin Mu, Xudong Han, Timothy Baldwin, Omri Abend, Eduard H. Hovy, and Lea Frermann. 2025. Control illusion: The failure of instruction hierarchies in large language models. *CoRR*, abs/2502.15851.

Zhichao Geng, Dongyu Ru, and Yang Yang. 2024. Towards competitive search relevance for inference-free learned sparse retrievers. *CoRR*, abs/2411.04403.

Michael Günther, Jackmin Ong, Isabelle Mohr, Alaeddine Abdessalem, Tanguy Abel, Mohammad Kalim Akram, Susana Guzman, Georgios Mastrapas, Saba Sturua, Bo Wang, and 1 others. 2023. Jina embeddings 2: 8192-token general-purpose text embeddings for long documents. *arXiv preprint arXiv:2310.19923*.

Keno Harada, Yudai Yamazaki, Masachika Taniguchi, Takeshi Kojima, Yusuke Iwasawa, and Yutaka Matsuo. 2025. Curse of instructions: Large language models cannot follow multiple instructions at once.

Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-Hsuan Sung, László Lukács, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient natural language response suggestion for smart reply. *arXiv preprint arXiv:1705.00652*.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*, 43(2).

Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, Xin Jiang, Ruifeng Xu, and Qun Liu. 2024. Planning, creation, usage: Benchmarking LLMs for comprehensive tool utilization in real-world complex scenarios. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 4363–4400, Bangkok, Thailand. Association for Computational Linguistics.

HuggingFace. 2025. LoRA (Low-Rank Adaptation) - Hugging Face LLM Course — huggingface.co. https://huggingface.co/learn/llm-course/en/chapter11/4#using-trl-with-peft. [Accessed 24-09-2025].

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*.

Mohammad Kachuee, Sarthak Ahuja, Vaibhav Kumar, Puyang Xu, and Xiaohu Liu. 2025. Improving tool retrieval by leveraging large language models for query generation. In *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*, pages 29–38, Abu Dhabi, UAE. Association for Computational Linguistics.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Besat Kassaie, Andrew Kane, and Frank Wm. Tompa. 2025. Exploiting query reformulation and reciprocal rank fusion in math-aware search engines. In *Proceedings of the 2025 ACM Symposium on Document Engineering*, DocEng '25, New York, NY, USA. Association for Computing Machinery.

H. W. Kuhn. 1955. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97.

Yuan-Ching Kuo, Yi Yu, Chih-Ming Chen, and Chuan-Ju Wang. 2025. MMLF: Multi-query multi-passage late fusion retrieval. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 6587–6598, Albuquerque, New Mexico. Association for Computational Linguistics.

Carlos Lassance, Hervé Déjean, Thibault Formal, and Stéphane Clinchant. 2024. Splade-v3: New baselines for splade. *Preprint*, arXiv:2403.06789.

Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. 2025. NV-embed: Improved techniques for training LLMs as generalist embedding models. In *The Thirteenth International Conference on Learning Representations*.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Do Xuan Long, Duy Dinh, Ngoc-Hai Nguyen, Kenji Kawaguchi, Nancy F. Chen, Shafiq Joty, and Min-Yen Kan. 2025. What makes a good natural language prompt? In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 5835–5873. Association for Computational Linguistics.

Xing Han Lù. 2024. Bm25s: Orders of magnitude faster lexical search via eager sparse scoring. *Preprint*, arXiv:2407.03618.

Tula Masterman, Sandi Besen, Mason Sawtell, and Alex Chao. 2024. The landscape of emerging ai agent architectures for reasoning, planning, and tool calling: A survey. *ArXiv*, abs/2404.11584.

Microsoft. 2025. Azure OpenAI in Azure AI Foundry Models Quotas and Limits — learn.microsoft.com. `https://learn.microsoft.com/en-us/azure/ai-foundry/openai/quotas-limits?tabs=REST`. [Accessed 05-10-2025].

OpenAI. 2024. text-embedding-3-small. `https://platform.openai.com/docs/models/text-embedding-3-small`. [Accessed 23-09-2025].

OpenAI. 2025. Introducing GPT-4.1 in the API — openai.com. `https://openai.com/index/gpt-4-1/`. [Accessed 12-07-2025].

Avinash Patil and Aryan Jadon. 2025. Advancing reasoning in large language models: Promising methods and approaches. *arXiv preprint arXiv:2502.03671*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive APIs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Cheng Qian, Chi Han, Yi Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. 2023. CREATOR: Tool creation for disentangling abstract and concrete reasoning of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 6922–6939, Singapore. Association for Computational Linguistics.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Towards completeness-oriented tool retrieval for large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, page 1930–1940, New York, NY, USA. Association for Computing Machinery.

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.

Saron Samuel, Dan DeGenaro, Jimena Guallar-Blasco, Kate Sanders, Seun Eisape, Arun Reddy, Alexander Martin, Andrew Yates, Eugene Yang, Cameron Carpenter, David Etter, Efsun Kayi, Matthew Wiesner, Kenton Murray, and Reno Kriz. 2025. Mmmorrf: Multimodal multilingual modularized reciprocal rank fusion. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '25, page 4004–4009, New York, NY, USA. Association for Computing Machinery.

SentenceTransformers. 2025. Training Overview; Sentence Transformers documentation — sbert.net. https://sbert.net/docs/sentence_transformer/training_overview.html#training-arguments. [Accessed 24-09-2025].

Tao Shen, Guodong Long, Xiubo Geng, Chongyang Tao, Yibin Lei, Tianyi Zhou, Michael Blumenstein, and Daxin Jiang. 2024. Retrieval-augmented retrieval: Large language models are strong zero-shot retriever. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 15933–15946, Bangkok, Thailand. Association for Computational Linguistics.

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. 2025. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24497–24524, Vienna, Austria. Association for Computational Linguistics.

Yifan Song, Guoyin Wang, Sujian Li, and Bill Yuchen Lin. 2025. The good, the bad, and the greedy: Evaluation of LLMs should not ignore non-determinism. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4195–4206, Albuquerque, New Mexico. Association for Computational Linguistics.

Weiwei Sun, Zhengliang Shi, Wu Jiu Long, Lingyong Yan, Xinyu Ma, Yiding Liu, Min Cao, Dawei Yin, and Zhaochun Ren. 2024. MAIR: A massive benchmark for evaluating instructed retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14044–14067, Miami, Florida, USA. Association for Computational Linguistics.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Yongqi Tong, Dawei Li, Sizhe Wang, Yujia Wang, Fei Teng, and Jingbo Shang. 2024. Can LLMs learn from previous mistakes? investigating LLMs' errors to boost for reasoning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3065–3080, Bangkok, Thailand. Association for Computational Linguistics.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2019. Representation learning with contrastive predictive coding. *Preprint*, arXiv:1807.03748.

Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. 2021. MiniLMv2: Multi-head self-attention relation distillation for compressing pre-trained transformers. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 2140–2151, Online. Association for Computational Linguistics.

Orion Weller, Benjamin Chang, Sean MacAvaney, Kyle Lo, Arman Cohan, Benjamin Van Durme, Dawn Lawrie, and Luca Soldaini. 2025. FollowIR: Evaluating and teaching information retrieval models to follow instructions. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11926–11942, Albuquerque, New Mexico. Association for Computational Linguistics.

Georg Wölflein, Dyke Ferber, Daniel Truhn, Ognjen Arandjelovic, and Jakob Nikolas Kather. 2025. LLM agents making agent tools. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 26092–26130, Vienna, Austria. Association for Computational Linguistics.

Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. 2025. Seal-tools: Self-instruct tool learning dataset for agent tuning and detailed benchmark. In *Natural Language Processing and Chinese Computing*, pages 372–384, Singapore. Springer Nature Singapore.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 641–649, New York, NY, USA. Association for Computing Machinery.

Qiancheng Xu, Yongqi Li, Heming Xia, and Wenjie Li. 2024. Enhancing tool retrieval with iterative feedback from large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 9609–9619, Miami, Florida, USA. Association for Computational Linguistics.

Zhichao Xu, Fengran Mo, Zhiqi Huang, Crystina Zhang, Puxuan Yu, Bei Wang, Jimmy Lin, and Vivek Srikumar. 2025. A survey of model architectures in information retrieval. *CoRR*, abs/2502.14822.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023.

React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, 4(2):100211.

Mert Yazan, Suzan Verberne, and Frederik Situmeang. 2024. The impact of quantization on retrieval-augmented generation: An analysis of small llms. In *IR-RAG@SIGIR*.

Peng Yu, En Xu, Bin Chen, Haibiao Chen, and Yinfei Xu. 2025. Qzhou-embedding technical report. *Preprint*, arXiv:2508.21632.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi Fung, Hao Peng, and Heng Ji. 2024. CRAFT: Customizing LLMs by creating and retrieving from specialized toolsets. In *The Twelfth International Conference on Learning Representations*.

Dharunish Yugeswardeenoo, Kevin Zhu, and Sean O'Brien. 2024. Question-analysis prompting improves LLM performance in reasoning tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pages 402–413, Bangkok, Thailand. Association for Computational Linguistics.

Haobo Zhang, Qiannan Zhu, and Zhicheng Dou. 2025a. Enhancing reranking for recommendation with LLMs through user preference retrieval. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 658–671, Abu Dhabi, UAE. Association for Computational Linguistics.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025b. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

Zimin Zhang, Qianli Wu, Botao Xia, Fang Sun, Ziniu Hu, Yizhou Sun, and Shichang Zhang. 2025c. Automated molecular concept generation and labeling with large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 6918–6936, Abu Dhabi, UAE. Association for Computational Linguistics.

Minjun Zhu, Yixuan Weng, Linyi Yang, and Yue Zhang. 2025. DeepReview: Improving LLM-based paper review with human-like deep thinking process. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 29330–29355, Vienna, Austria. Association for Computational Linguistics.

Shengyao Zhuang, Xueguang Ma, Bevan Koopman, Jimmy Lin, and Guido Zuccon. 2024. PromptReps: Prompting large language models to generate dense and sparse representations for zero-shot document retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 4375–4391, Miami, Florida, USA. Association for Computational Linguistics.

A Benchmarking Current Retrievers

We benchmark several popular retriever models to establish a baseline and determine the best-performing ones to improve on. The selection criteria for the models were based on their performance on the BEIR (Thakur et al., 2021) and MTEB (Enevoldsen et al., 2025) leaderboards, which are considered the gold standard for ranking retriever models. We categorise them as,

- **Sparse**: Models that rely on surface-level token-matching for search. We use the popular BM25 (Lù, 2024) for this category
- **Semi-Sparse**: These are neural models that produce high-dimensional vectors (mostly 0's) but *learn* the non-zero elements to capture better semantics while optimising for search efficiency. We select SPLADE (Lassance et al., 2024) and opensearch (Geng et al., 2024) for this category
- **Dense**: Models that generate low-dimensional embeddings aimed at capturing deeper semantics over simple token matching. These include DPR, BGE (Xiao et al., 2024), Jina (Günther et al., 2023), MiniLM (Wang et al., 2021), NV-Embed, LGAI-Embedding (Choi et al., 2025), QZhou-Embedding (Yu et al., 2025) and Qwen3 (Zhang et al., 2025b). We only use one proprietary model, i.e., OpenAI's `text-embedding-3-small` (OpenAI, 2024) to maintain cost.

The results from benchmarking are shown in Table 6. Firstly, we notice the competitiveness of sparse retrievers for tool-retrieval. This aligns with observations by related studies (Shen et al., 2024; Zhuang et al., 2024) which show how simple term-matching based models generalise to different tasks, better than dense models. In our tests, we find BM25 and opensearch matching the performance of MiniLM and even `text-embedding-3-small`, which not only highlights their tool-retrieval capabilities but also as an effective solution for settings where training dense models is infeasible.

However, when scaling up model size and moving beyond lexical features, we find that retrieval performance improves dramatically. This is evident from the gap between the best sparse (opensearch) and best dense (Qwen3) model, showing as much as 20% increase (for MRR). Thus, it is clear that while sparse encoders are reasonably effective, to observe true tool-retrieval capabilities, dense models are fundamentally necessary.

For testing *ToolDreamer*, we choose BM25 and Qwen3 as the best sparse/dense models, respectively. While SPLADE and opensearch are categorised as *semi-sparse*, they still involve some form of contrastive learning to represent tokens. As such, we wanted to see how our method performs when using a *pure* lexical model. Improvements for BM25 would additionally indicate the quality of the generated tools, as it means that the LLM effectively generates descriptions similar in phrasing to the expected tools.

B LLM Performance v/s Tool Number

To highlight the issue of tool limits, we examine how GPT-4.1 (OpenAI, 2025), and Qwen3's (Yang et al., 2025) performance changes as the number of tools increases (c.f. Fig. 3). We use `SealTools` (Wu et al., 2025), a dataset for LLM-tool call training for illustration. For each query, we randomly sample *negative* (irrelevant) tools to meet the tool counts (10-128). For example, if a query has 1 gold tool, we randomly sample 9 negative tools and shuffle their order to prevent location bias (Liu et al., 2024). As we can see, at relatively small scales (10-128 tools), each model handles tool calling relatively well, showing consistent performance across varying tool counts. However, beyond 50 (for Qwen3) and 128 (for GPT-4.1), each model hits their limit (Qwen: 32K context window; GPT: 1M), being unable to register more tools. GPT has another issue - since it is an API-based model, OpenAI/Azure restricts the maximum number of tools to 128 (Microsoft, 2025). Thus, despite theoretically being able to support even more tools, given a 1M context window, GPT cannot be provided with more tools. These issues reinforce the necessity for tool retrieval.

C Training/Hardware Details

We train our models using LoRA (Hu et al., 2022) [rank=8, alpha=16, dropout=0.1] for 1 epoch, with a learning rate of $1e-5$ and batch size 1. These parameters were selected after brief experimentation with standard training values (SentenceTransformers, 2025; HuggingFace, 2025), which showed to work the best. We run our training experiments on 4 NVIDIA A100 80 GB cards and inference on 1 of them.

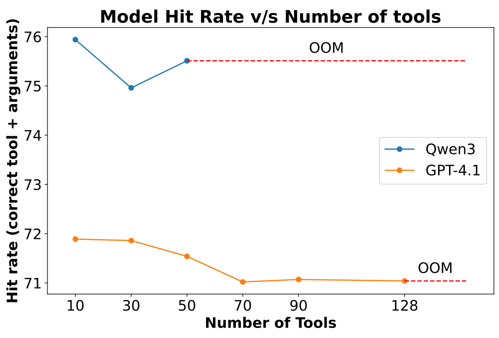

Figure 3: Impact of scaling the number of tools on LLM hit-rate (correct tool call with appropriate arguments).

D Note on LLM-Fusion (Re-Ranking)

Here, we provide our justification for not using LLM-fusion in *ToolDreamer*. First, using an LLM for ranking top-K lists requires additional API calls, which incur cost and can be prohibitive for a large number of queries. Second, LLMs are occasionally prone to missing instructions (Harada et al., 2025; Geng et al., 2025), i.e., ignoring key requirements in the prompt. We notice a similar trend where for many samples, GPT under-reported (less than 10) or over-reported (more than 10) tools. For the former case, we use each generated tool, and for the latter, we retain the top 10. Although this setup works, it is slightly *unfair* to compare it with RRF since it *always* works with 10 tools impacting the metrics accordingly. Finally, although LLMs have become more reliable, the risk of *hallucination*, i.e., spurious responses (Huang et al., 2025), is not completely mitigated. As a result, LLM-based ranking *may* introduce tools that are absent from the provided list, which is a major drawback. While hallucination concerns exist for hypothetical tool generation also, there we encourage open-ended creativity, which is detrimental for a systematic process such as list ranking.

E Dataset Filtration

To solve the retrieval problem, we need to ensure two things: (i) each tool must have a valid (non-empty) description to build the tool database index, and (ii) each positive/negative tool for a query must

	Split	Web				Code				Customized				Avg.			
	Model	N@10	P@10	R@10	MRR	N@10	P@10	R@10	MRR	N@10	P@10	R@10	MRR	N@10	P@10	R@10	MRR
Sparse	BM25	27.25	5.72	36.25	28.10	29.50	4.90	37.38	29.83	31.90	8.70	36.50	39.63	29.55	6.44	36.71	32.52
	SPLADE-v3	29.22	6.08	39.52	29.56	26.45	4.96	37.24	25.52	36.89	10.12	46.24	41.16	30.85	7.05	41.00	32.08
	opensearch-embedding	31.97	6.63	42.64	32.32	27.58	5.10	38.58	25.45	38.32	10.69	46.62	43.54	32.62	7.47	42.61	33.77
Dense	DPR	7.41	1.59	11.01	7.35	9.76	1.95	13.35	9.16	11.93	2.75	15.73	13.87	9.70	2.10	13.36	10.13
	BGE-Large	29.16	5.97	39.46	29.66	30.40	5.52	42.18	28.15	34.86	9.47	41.94	41.10	31.47	6.99	41.19	32.97
	jina-v2	28.34	5.51	37.40	28.97	28.41	5.02	37.90	27.90	35.23	9.46	44.47	39.91	30.66	6.66	39.92	32.26
	all-MiniLM-L6-v2	29.53	5.77	39.05	30.38	27.23	4.92	37.11	26.85	33.15	8.83	40.81	38.76	29.97	6.51	38.99	32.00
	NV-Embed-v2	33.27	6.59	43.65	33.80	38.79	6.66	52.05	36.60	41.31	10.60	50.60	47.10	37.79	7.95	**48.77**	39.17
	`text-embedding-3-small`	32.72	6.42	42.47	33.30	26.86	5.00	37.19	25.15	38.01	9.94	44.96	44.24	32.53	7.12	41.54	34.23
	LGAI-Embedding	32.63	6.47	43.08	32.82	34.02	5.69	44.01	33.55	37.23	9.90	45.69	42.77	34.63	7.35	44.26	36.38
	QZhou-Embedding	23.67	4.61	31.76	23.8	21.95	4.19	30.02	21.95	30.14	7.84	37.07	35.11	25.25	5.55	32.95	26.95
	Qwen3-8B	35.81	7.20	47.09	36.64	39.58	6.54	51.32	38.53	39.89	10.65	47.70	46.14	**38.43**	**8.13**	48.70	**40.44**

Table 6: Zero-shot retriever benchmarking on each split of `ToolRet`. Best scores are in bold.

also have a valid description to perform retrieval. Accordingly, we filter the dataset to include tools and queries that satisfy these conditions. We found that the customised split had no noisy samples, and only the web and code split needed cleaning. This removed 264 tools/77 queries and 1 tool from each split, respectively, maintaining a count close to the original number of samples.

The provided training dataset has ~200K samples. However, many samples had profanities, repeated queries, negative tools appearing in the positive tools list, and questions with 0 negative tools (after removing the overlapping tools). To keep our data generation reasonable and devoid of such entries, we filter and sample 5K unique queries from the base set.

F Generated Hypothetical Tools

We provide an example of generated tools for a query from the Web split of ToolRet in Figure 4. As we can see, the query is quite dense and requires multiple tools to be solved. The associated gold tools take care of each aspect of the request. The generated hypothetical tools then align with each gold tool based on GPT's reasoning. These tools are not aligned in the figure. However, their language provides enough clues to the embedding model to draw precise similarity boundaries.

G Prompts

Below, we provide the prompts used to generate the hypothetical tools for *ToolDreamer*.

H Paper Code

The code for this paper is publicly available at `https://github.com/boschresearch/ToolDreamer`.

Example of Generated Tool

Query: *I'm organizing a charity event and need some rewards for the attendees. Can you suggest some rewards available on the Rewards as a Service platform that I can offer as incentives? Also, fetch a random kitten image to include in the event promotion materials. Lastly, provide me with the order history for my account to see if I have any pending orders.*

Gold Tool 1: *Fetches random kitten image!*
Gold Tool 2: *Get more information about all the orders placed on this customer and account*
Gold Tool 3: *Gets the list of rewards available for the platform*

Hypothetical Tool 1: *Retrieves a catalog of rewards currently offered by the Rewards as a Service platform.*
Hypothetical Tool 2: *Returns a random kitten image suitable for use in event promotion or other materials.*
Hypothetical Tool 3: *Retrieves the order history, including pending and completed orders, for a specified account on the Rewards as a Service platform.*

Figure 4: An example of hypothetical tools for a given query.

Prompt For Inferior Quality Tools

```
You are an assistant that comes up with tools. The tools should be related in some way to solving
a query. Just give a short explanation, a name, and a description for each tool.

Output Format
Thought: [say something about the tool]
Tool Name: [give it a name]
Tool Description: [say what it does]
```

Figure 5: Prompt for generating inferior quality hypothetical tools for test data (ablation study).

Prompt For LLM Reranking

You are given a **question** and a **set of tools**, each with a description. Your task is to evaluate the tools and rank them by how useful they are for answering the question.
Instructions:

1. If there are **10 tools or fewer**, rank and score **all of them**.
2. If there are **more than 10 tools**, consider the full set and return **only the top 10** ranked by usefulness.
3. Use a **score between 0.00 and 1.00**, where:
   - ★ 1.00 = extremely useful / directly relevant
   - ★ 0.00 = not useful at all
4. Start with a **concise reasoning summary** (2–4 sentences maximum).
5. Then, present the ranked list in the following format, sorted from most to least useful:

   ```
   Tool: [tool name]
   Score: [0.00–1.00]
   ```

Figure 6: Prompt for LLM-reranking (ablation study).

Prompt For Training Data

You are a **tool-crafting assistant**. Your job is to design a precise set of **hypothetical tools** that collectively solve a given user query. Each tool must behave as an **independent, stateless function** with full access to all structured data relevant to its subtask. Assume **no external internet access**.
TASK INSTRUCTIONS

1. **Query Breakdown**
   Analyze the user query step by step. Identify all explicit and implicit **subtasks** required to generate a complete answer.
   - ★ If multiple subtasks are tightly related, keep them separate unless they are inseparable.
   - ★ If there appear to be fewer subtasks than the number of tools requested, dig deeper to uncover **hidden or supporting subtasks**.
2. **Tool Design**
   For **each subtask**, propose **exactly one tool** to handle that specific function.
3. **Design Principles**
   - ★ Tools must be: **modular**, **reusable**, and **implementation-agnostic**.
   - ★ Use **general tool names** unless a domain-specific name is clearly justified (e.g., ‘weatherAPI_California’). If specificity is used, explain why.
   - ★ Tools should focus on a **single responsibility** each.
4. **Naming Rules**
   - ★ Choose either ‘camelCase’ or ‘snake_case’ and stick with it throughout.
   - ★ Tool names must be **concise, descriptive, and aligned directly to the subtask**.
   - ★ Avoid overly broad names (e.g., ‘processData’) or overly detailed names (e.g., ‘getSanFranciscoHourlyWeatherFromNOAA’).
5. **Tool Count Compliance**
   - ★ Always produce **exactly the number of tools specified** in the query.
   - ★ If subtasks seem fewer than required, introduce additional implicit tools (e.g., parsing input, validating scope, formatting output).

OUTPUT FORMAT
For each tool, provide the following fields **in order**:
```
Thought: <Explain what this subtask is and why this tool is needed>
Tool Name: <Concise, consistent name>
Tool Description: <Briefly describe what the tool does>
```

EXAMPLE
QUERY: What is the average temperature in San Francisco? | TOOLS NEEDED: 1
```
Thought: The query requires access to temperature data for a specific location. A tool is needed to fetch weather data for San Francisco.
Tool Name: weatherAPI\_California
Tool Description: Retrieves current and historical temperature data for locations in California.
```

Figure 7: Prompt for generating hypothetical tools for training data.

Prompt For Test Data

You are a tool-crafting assistant. Your goal is to design a set of **hypothetical tools** that can collectively solve a given user query. These tools behave like **independent, stateless functions** and have access to **all relevant structured data** needed to complete their task. No external internet access is assumed.
High-Level Strategy

1. **Break down** the query into distinct, logically separable **subtasks**.
2. For **each subtask**, propose **exactly one tool** that performs **only that task**.
3. Tools must be **modular, reusable, and generic** where possible (e.g., 'getPopulationData'). If a subtask clearly requires **specialized or scoped** logic (e.g., region-specific), it's acceptable to use a **specialized tool**, but its name must reflect its limited scope (e.g., 'weatherAPI_California', 'fetchStockData_TechSector').
4. All tools must be:
   - ★ **Implementation-agnostic**
   - ★ **Independent**
   - ★ **Named consistently** using 'camelCase' or 'snake_case'
5. **Every query must result in at least one tool** being proposed.

Output Format
For each tool, respond using the following format:
```
Thought: [Explain why this tool is necessary and what subtask it addresses]
Tool Name: [Concise and consistent tool name]
Tool Description: [Describe what this tool does in one or two sentences, avoiding implementation
details]
```

Example
User Query:
```
What is the average temperature in San Francisco?
```
Assistant Response:
```
Thought: To answer this query, we need to retrieve historical and/or current temperature data for a
specific location in California.
Tool Name: weatherAPI\_California
Tool Description: Retrieves current and historical temperature data for locations in California.
```
Additional Tips

- ★ Do **not** suggest tool pipelines or combine subtasks into a single tool.
- ★ If the query involves **data transformation** (e.g., averaging, filtering, ranking), propose a separate tool for that.
- ★ Treat each tool as **stateless and composable**.

Figure 8: Prompt for generating hypothetical tools for test data.