Beyond Single-Shot: Multi-step Tool Retrieval via Query Planning

Wei Fang and James Glass
Massachusetts Institute of Technology, Cambridge MA, USA
{weifang,glass}@mit.edu

Abstract

LLM agents operating over massive, dynamic tool libraries rely on effective retrieval, yet standard single-shot dense retrievers struggle with complex requests. These failures primarily stem from the disconnect between abstract user goals and technical documentation, and the limited capacity of fixed-size embeddings to model combinatorial tool compositions. To address these challenges, we propose TOOLQP, a lightweight framework that models retrieval as iterative query planning. Instead of single-shot matching, TOOLQP decomposes instructions into sub-tasks and dynamically generates queries to interact with the retriever, effectively bridging the semantic gap by targeting the specific sub-tasks required for composition. We train TOOLQP using synthetic query trajectories followed by optimization via Reinforcement Learning with Verifiable Rewards (RLVR). Experiments demonstrate that TOOLQP achieves state-of-the-art performance, exhibiting superior zero-shot generalization, robustness across diverse retrievers, and significant improvements in downstream agentic execution.[1]

1 Introduction

Large language models (LLMs) have evolved from simple text generation into integration within agentic frameworks that allow them to solve a variety of complex tasks such as math, reasoning, and coding, by interacting with external environments (Mialon et al., 2023; Yao et al., 2023). Integral to this paradigm shift is the ability to use tools, namely APIs, databases, and software tools, to extend the models' capabilities beyond their parametric knowledge (Qin et al., 2024a). As agentic workflows are being developed, the scale of these tool libraries are expanding rapidly, moving from dozens of hand-picked functions to massive, dynamic repositories containing tens of thousands of APIs. In these scenarios, it is computationally infeasible to fit the entire tool context, including documentation, tool-specific instructions, and in-context tool demonstrations, into the LLM's context window. Consequently, tool retrieval has been fundamental to the design of practical frameworks, retrieving relevant tools from the toolset as an initial step (Li et al., 2023b).

While recent work has adapted information retrieval (IR) techniques with ad-hoc tool-use datasets (Qu et al., 2024; Xu et al., 2024) to enhance tool retrieval, they along with approaches that perform well on conventional IR benchmarks are shown to exhibit poor performance on a wide variety of tool-use tasks and tools (Shi et al., 2025). These existing approaches typically employ dense embeddings with a standard single-shot retrieval step, and while they may be effective for simple, direct queries, they often fail significantly when applied to complex, compositional tasks.

We identify three fundamental challenges that stem from applying these single-shot retrieval paradigms to dynamic agentic workflows. First, semantic misalignment creates a critical disconnect between the high-level vocabulary of user intent and the technical specificity of tool schemas. For instance, a user may request to "make this audio recording high quality," while a relevant tool `scipy.signal.lfilter(b, a, x)` may be defined strictly by mathematical parameters like `b` (numerator) and `a` (denominator) along with technical descriptions such as "Filter data along one-dimension with an IIR or FIR filter." Standard dense retrievers fail to bridge the gap between the subjective goal ("high quality") and the implementation-level terminology (`lfilter`), and this is exacerbated by the heterogeneous nature of large-scale tool libraries, where documentation styles vary from verbose descriptions to raw,

[1]Source code is available at https://github.com/wfangtw/toolqp.

Findings of the Association for Computational Linguistics: ACL 2026, pages 42119–42144

schema-heavy protocols (Qin et al., 2024b; Shi et al., 2025). Furthermore, real-world tasks are inherently compositional, often requiring the simultaneous application of multiple distinct tools; for example, retrieving both a `WeatherAPI` and a `StockMarketDB` to "analyze how rain affects retail sales." However, a single fixed-dimensional vector lacks the capacity to encode the combinatorial diversity of multiple disparate tools (Weller et al., 2025), a limitation that is amplified as tool libraries scale. Finally, current single-shot methods lack interactive toolset awareness. They treat the repository as a static database and cannot handle internal constraints or changes. In contrast, interacting with the environment provides critical feedback on inter-tool dependencies (Xu et al., 2024), for example discovering that a `forecasting_tool` requires a specific `region_id` from a lookup utility, and allows the system to adapt to modifications within the toolset.

To address these limitations, we propose the Tool Query Planner (TOOLQP), a framework that formulates retrieval as an iterative planning process rather than a static single-shot semantic matching task. TOOLQP decomposes complex user requests into a logical sequence of high-level sub-tasks, interactively retrieving relevant tools for each step through a unified and light-weight model designed to interface with any existing retrieval system. This approach effectively bridges semantic gaps by inferring functional utility from abstract goals, circumvents compositional bottlenecks by retrieving conceptually-similar tools step-by-step rather than compressing them into a single vector, and resolves inter-tool dependencies and toolset modifications via dynamic environment feedback. Our design is inherently modular and generalizable, functioning as a complementary layer atop standard retrievers without requiring architectural changes to the underlying index or the downstream reasoning LLM. Furthermore, the explicit planning trajectory generated during retrieval could serve as valuable context for the downstream agent to ground its execution. Extensive experiments across a wide variety of tool-use tasks demonstrate that TOOLQP significantly improves both retrieval accuracy and downstream execution success rates compared to state-of-the-art baselines. Overall, our contributions are summarized as follows:

- We propose TOOLQP, a novel framework that fundamentally shifts tool retrieval from a static similarity matching task to a dynamic planning process. By reframing the problem, we enable the resolution of complex, compositional queries and facilitate the discovery of inter-tool dependencies that single-shot dense retrievers are inherently limited in addressing.

- We design TOOLQP as a modular, lightweight layer that integrates seamlessly with existing dense retrievers and downstream LLMs. Our approach leverages interactive feedback to adapt to heterogeneous tool documentation styles and diverse retrieval environments without requiring architectural modifications or fine-tuning of the underlying system.

- We demonstrate through extensive experiments on a diverse set of tool-use benchmarks that TOOLQP significantly outperforms state-of-the-art baselines. Our results show consistent improvements in both retrieval performance and downstream execution success rates, particularly in scenarios characterized by high compositional complexity and abstract user intent.

2 Related Work

Tool Learning and Retrieval. LLMs are increasingly employed in agentic frameworks that enable tool use for solving complex tasks (Gupta and Kembhavi, 2023; Mialon et al., 2023; Surís et al., 2023; Team et al., 2023; Wu et al., 2023; Cai et al., 2024; Qin et al., 2024a; Zhang et al., 2024). Conventional approaches include post-training fine-tuning (Parisi et al., 2022; Thoppilan et al., 2022; Patil et al., 2023; Schick et al., 2023; Dubey et al., 2024; Yang et al., 2023; Liu et al., 2025; Lin et al., 2025; Yang et al., 2025), or in-context learning with meta-prompts for zero-shot tool usage (Lu et al., 2023; Shen et al., 2023b; Song et al., 2023; Qin et al., 2024b; Zhuang et al., 2024). However, scaling to large toolsets (e.g., 52k+ in RapidAPI) is challenging due to limited context windows and performance degradation from long contexts (Liu et al., 2024a; Qu et al., 2024), an issue exacerbated when including necessary instructions and demonstrations (Hsieh et al., 2023; Xu et al., 2023). Furthermore, frequent updates to the toolset make retraining cost-prohibitive, necessitating zero-shot approaches (Fang et al., 2025; Qu et al., 2025). Contemporary tool-use frameworks address this via semantic retrievers, utilizing either conventional

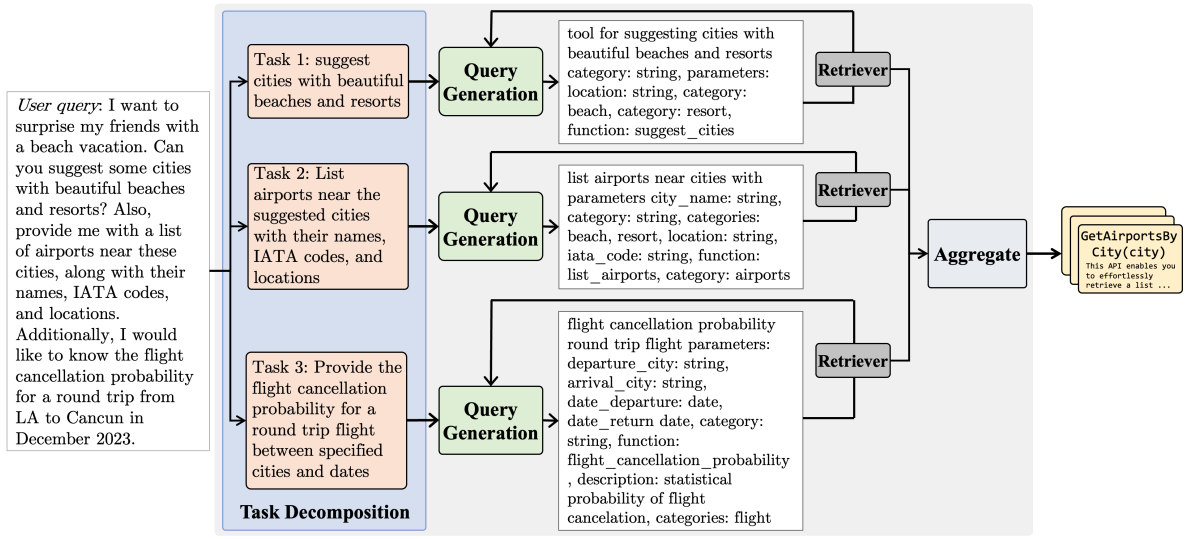

Figure 1: Overview of the TOOLQP framework. The Planner decomposes a complex user query (e.g., travel planning) into sequential sub-tasks. For each sub-task, it interactively generates queries, processes feedback from the dense retriever, and self-corrects if necessary, before aggregating the final set of relevant tools.

dense embeddings ill-suited for tools (Shi et al., 2025) or task-specific models lacking generalizability (Gao et al., 2024; Kong et al., 2024; Qin et al., 2024b; Wang et al., 2025). Tool retrieval is arguably more challenging than conventional IR due to multi-tool composition (Qu et al., 2024) and the semantic gap between user intent and technical documentations (Chen et al., 2024b), and TOOLQP is explicitly designed to address these challenges.

Generative Modeling for Retrieval. Document expansion, which appends generated queries to documents, and query expansion, which augments queries with relevant content, are established IR techniques that are simple yet effective (Maron and Kuhns, 1960; Qiu and Frei, 1993; Xu and Croft, 1996; Singhal and Pereira, 1999). Pseudo-relevance feedback similarly expands queries using top-ranked documents from a first-pass retrieval (Rocchio, 1971; Croft and Harper, 1979; Lavrenko and Croft, 2001; Wang et al., 2021; Yu et al., 2021; Li et al., 2023a). Recently, LLMs have advanced generative document expansion for IR and QA (Dai and Callan, 2019; Nogueira et al., 2019; Formal et al., 2021; Lewis et al., 2021), and query expansion via hypothetical document generation (Gao et al., 2023; Jagerman et al., 2023; Mackie et al., 2023; Shen et al., 2023a; Wang et al., 2023) or expansion term selection (Mao et al., 2021; Chuang et al., 2023). Alternatively, LLMs facilitate query rewriting–reformulating the user query entirely–and retrieval scoring and re-ranking (Yu et al., 2020; Sachan et al., 2022; Ma et al., 2023; Mao et al., 2023; Sun et al., 2023; Ye et al., 2023; Fang et al., 2024; Feng et al., 2024). Recent studies adapt these methods for tool retrieval via few-shot prompting (Chen et al., 2024b), or by incorporating them within the retriever fine-tuning process (Xu et al., 2024). Contemporaneous work further explores training with large-scale expansion augmentation (Sengupta et al., 2025; Lu et al., 2025). In comparison, TOOLQP learns robust query planning while being very lightweight, outperforming without prompting large models or requiring retriever fine-tuning.

3 TOOLQP: Query Planning

3.1 The Modeling Framework

We assume access to a tool retriever $\mathcal{E}$ that receives a user query q and returns a ranked list of tools from the tool set $\mathcal{D}$. A typical single-shot retrieval directly embeds q and all tools $d \in \mathcal{D}$ using the retriever $\mathcal{E}$ and selects the tools with highest similarity score as relevant tools. However, as previously discussed, this single-shot paradigm struggles to capture compositional intent and lacks interactivity required to resolve dynamic changes to the toolset or tool-specific dependencies. To address these limitations, we propose TOOLQP, a query planning framework that casts tool retrieval as a sequential decision-making process. Rather than treating the retriever as a static index, $\mathcal{E}$ is treated as a dynamic environment with which TOOLQP can interact with

and explore the tool space. The TOOLQP framework consists of three stages: *planning*, *query generation*, and *aggregation*, as illustrated in Fig. 1.

Planning by Task Decomposition. The primary challenge in tool retrieval is the semantic misalignment between high-level user intents and low-level tool representations. Directly querying the retriever with a complex instruction q often leads to suboptimal performance since the necessary tool keywords are usually absent from the user's phrasing. To bridge this gap, TOOLQP first generates a natural language plan $\mathcal{P}$, and then decomposes the user query q into a logical sequence of sub-tasks $\{s_n\}$. Unlike prior methods that rely on costly, prompt-engineered calls to large models to identify user intents, TOOLQP is a lightweight and modular solution that learns this decomposition capability directly and efficiently.

Interactive Query Generation. Guided by the generated plan $\mathcal{P}$, TOOLQP then targets each sub-task by interactively generating a sequence of search queries $\{q_t\}$. At each step t, the model observes the retrieval feedback $O_t = \mathcal{E}(q_t; \mathcal{D})$ before generating the next query q_{t+1}. This feedback loop addresses the limitations of single-shot retrieval by allowing the model to dynamically adjust its strategy based on the retrieval feedback. For example, if an initial query retrieves a relevant tool that requires a specific input argument, the model can generate subsequent queries to target that prerequisite tool. This iterative process continues until the model determines the sub-tasks in $\mathcal{P}$ have been sufficiently covered, resulting in a trajectory of query-retrieval pairs $\{(q_t, O_t)\}$.

Retrieval Aggregation. The final stage is to aggregate results from the query-retrieval trajectory into a single ranked list that can be used by a downstream LLM. While previous work have explored reciprocal rank fusion (RRF) or engineered complex rank-fusion algorithms to integrate multiple retrieval rankings, we find these methods unsuitable or unnecessary for our planning-based framework. Since the model may generate a varying number of queries for different sub-tasks, prior methods tend to bias the final ranking toward sub-tasks that require more query attempts. Therefore, we employ a simple yet robust *peak-rank* aggregation strategy: for every unique tool $d \in \bigcup O_t$, we assign its final rank based strictly on the highest rank it achieved across any single retrieval attempt. This effectively balances the retrieval results across each sub-task.

3.2 Data Generation & SFT

To train a TOOLQP model, it requires supervision for the modeling outputs we described earlier: the task decomposition plan $\mathcal{P}$, and the subsequent search query trajectory $\{(q_t, O_t)\}$. However, standard tool retrieval datasets typically provide only the user query q and the final set of ground-truth tools $\mathcal{T}^\star$. While Shi et al. (2025) previously paired each instance with a high-level instruction, which we can use as target for $\mathcal{P}$, we still lack the query trajectories required for training. We thus design a data synthesis pipeline to generate effective query trajectories from data compiled by Shi et al. (2025) using a teacher model. The process, illustrated in Alg. 1, involves three stages, which we describe below.

Plan Alignment. First, we must ground the high-level natural language plan $\mathcal{P}$ in the ground-truth tools $\mathcal{T}^\star$. We prompt a teacher model to parse $\mathcal{P}$ into a sequence of discrete sub-tasks and assign the corresponding subset of target tools $t \in \mathcal{T}^\star$ to each sub-task. This results in a reasonable task decomposition and also a clear mapping between each sub-task and the tools required to complete them.

Query Generation. Next, we generate the search queries necessary to retrieve the assigned tools for each sub-task. To prevent generating trivial queries such as using the exact target tool names, we use the teacher model to simulate the query generation process, without conditioning on the specific target tool names; specifically, we prompt the teacher model to generate N candidate queries for a sub-task based only on the sub-task description. This constraint forces the generation of queries that describe the desired functionality of the tool rather than memorized identifiers. However, for tools that are difficult to retrieve with generic descriptions, we adopt a curriculum learning approach: if the initial query candidates fail to retrieve the target tools (as determined in the verification step below), we iteratively re-prompt the teacher with more target information until a valid query is found.

Query Verification and Trajectory Construction. For each step, we input all N candidate queries to the retriever $\mathcal{E}$ to obtain the ranks of the target tools, and select the query candidate that achieves the highest recall and is higher than a set thresh-

Algorithm 1 TRAJECTORYGENERATION

Require: q: user query, $\mathcal{P}$: natural language plan, target tools $\mathcal{T}^\star$, teacher model $\mathcal{M}$

 # Parse plan into sub-tasks with their target tools

$\{(s_n, \mathcal{T}_n^\star)\}_{n=1}^{K} \leftarrow \text{PARSE}_{\mathcal{M}}(q, \mathcal{P})$

$\mathcal{H} \leftarrow [\mathcal{P}, s_1, \ldots, s_K]$

for $n \leftarrow 1$ **to** K **do**

  $\mathcal{A} \leftarrow [\,]$

  # Loop until retrieved tools rank well against targets

  while $\text{AVGRANK}(\mathcal{T}_n^\star, O_t) > r_\tau$ **do**

    # Expand context c, sample N candidates

    $c \leftarrow \text{ADDMOREINFO}(c, \mathcal{T}_n^\star)$

    $Q_{\text{Cand}} \leftarrow \{\text{GENERATE}_{\mathcal{M}}(q, s_{1:n-1}, c)\}_{i=1}^{N}$

    $\mathcal{O}_{\text{Cand}} \leftarrow \{\text{RETRIEVE}(Q_{\text{Cand}}^{(i)})\}_{i=1}^{N}$

    # Pick candidate with best retrieval match

    $i^\star \leftarrow \text{Best}_{i \in \{1,\ldots,N\}}\ \text{AVGRANK}(\mathcal{T}_n^\star, \mathcal{O}_{\text{Cand}}^{(i)})$

    $q_t, O_t \leftarrow Q_{\text{Cand}}^{(i^\star)}, \mathcal{O}_{\text{Cand}}^{(i^\star)}$

    $\text{APPEND}(\mathcal{A}, (q_t, O_t))$

  end while

  # With prob. p keep the trajectory incl. failed attempts

  if $\text{BERN}(p)$ **then**

    $\text{EXTEND}(\mathcal{H}, \mathcal{A})$

  else

    $\text{APPEND}(\mathcal{H}, (q_t, O_t))$

  end if

end for

return $\mathcal{H}$

old rank r_τ to be the valid query. To construct a full trajectory of $\{(q_t, O_t)\}$, we gather all valid queries q_t for all sub-tasks of the user query q. Additionally, we include query sequences that include *failed* attempts that yielded low ranks followed by a subsequent *successful* query to simulate an interactive trial-and-error process, providing the model with explicit training examples of self-correction. The resulting synthetic data trajectory, comprising the high-level plan P, sub-tasks $\{s_n\}$, and query-retrieval feedback sequence $\{(q_t, O_t)\}$, is used to train the model via standard supervised fine-tuning with maximum likelihood and teacher forcing.

3.3 Training – RLVR

While SFT provides a strong baseline capability by distilling the teacher's trajectories, to enable the model to explore the search space and discover query strategies that maximize retrieval performance, we further train the model using Reinforcement Learning with Verifiable Rewards (RLVR) (Lambert et al., 2024; Guo et al., 2025a; Yue et al., 2025). Specifically, we employ Group Relative Policy Optimization (GRPO) (Shao et al., 2024), which eliminates the need for a value network and leverages the deterministic nature of our retrieval environment. The training objective optimizes an overall reward $\mathcal{R} = \beta_1 \mathcal{R}_{\text{retrieval}} + \beta_2 \mathcal{R}_{\text{format}} + \beta_3 \mathcal{R}_{\text{plan}}$. The primary reward, $\mathcal{R}_{\text{retrieval}}$, is a sequence-level reward calculated via the nDCG@K and Recall@K of the final aggregated tool list against the ground truth, which encourages the model to optimize global coverage rather than individual steps. $\mathcal{R}_{\text{format}}$ is used to enforce formatting validity, while $\mathcal{R}_{\text{plan}}$ serves as a regularizer that computes the semantic similarity between the generated plan and the original high-level plan $\mathcal{P}$, preventing the model from deviating from the user's intent while exploring valid decompositions.

4 Experiments

4.1 Tool Retrieval

Benchmark and Setup. We evaluate TOOLQP on ToolRet (Shi et al., 2025), a comprehensive benchmark comprising 35 widely-used tool-calling datasets categorized into *Web*, *Code*, and *Custom* domains, with an overall toolset of 44k tools. For training TOOLQP, we utilize a subset of the training split compiled by Shi et al. (2025), which is sourced from the ToolBench, ToolACE, and APIGen training sets. Since these training sources fall under the Web category, we treat the corresponding test sets as *in-domain*, while the rest constitute a true *zero-shot transfer* setting. Following prior work, we report the standard IR metric Normalized Discounted Cumulative Gain (nDCG@K), and Completeness@K ($\mathbb{1}[R@K = 1]$), with $K = 10$ and macro-averaging across datasets within the same category. For this setting, we use the retriever `gte-Qwen2-1.5B-instruct` (Li et al., 2023c) (abbrev. `gte-Qwen`) as the base retriever for the baseline methods and TOOLQP. Additional details about the data and licenses can be found in Appx A.

Baselines. We compare TOOLQP against three categories of baselines. **Prompting methods** (using `Qwen3-30B-A3B-Instruct-2507`): (a) Q2E: direct query expansion (Jagerman et al., 2023); (b) Q2D: query expansion by hypothetical tool generation (Wang et al., 2023); (c) HyDE: using averaged embeddings of generated tools (Gao et al., 2023); (d) D2Q: document expansion (Nogueira et al., 2019); and (e) Re-Invoke: Intent extraction then D2Q pipeline (Chen et al., 2024b). **Re-ranking methods**: We re-score the top-20 results from `gte-Qwen` using cross-encoders, `bge-reranker-v2-m3` and `bge-reranker-v2-gemma` (Chen et al., 2024a). **Fine-tuning methods** (on the same data): (a) Q2P:

Method	In-Domain		Zero-Shot Transfer							
	Avg		Web*		Code		Custom		Macro-Avg	
	N@10	C@10	N@10	C@10	N@10	C@10	N@10	C@10	N@10	C@10
Base Retriever (`gte-Qwen`)	43.3	47.8	29.7	17.8	24.1	30.8	35.6	32.4	29.8	27.0
Prompting (`Qwen3-30B-A3B-Instruct`)										
Q2E/ZS	44.6	49.0	30.8	19.6	26.5	34.0	38.7	35.7	32.0	29.8
Q2D/ZS	52.0	56.8	24.5	18.1	22.2	29.1	33.3	33.5	26.7	26.9
HyDE/ZS	47.9	52.3	19.7	17.0	13.6	18.8	26.4	29.7	19.9	21.8
D2Q	51.5	54.0	26.6	16.3	25.4	32.0	33.3	30.8	28.4	26.4
Re-Invoke	51.5	56.8	28.1	18.5	26.0	33.5	36.8	31.7	30.3	27.9
Q2E/PRF	45.0	49.4	31.0	18.8	26.8	33.1	35.3	33.3	31.0	28.4
Q2D/PRF	46.9	50.2	30.5	18.9	26.1	31.1	37.0	33.7	31.2	27.9
HyDE/PRF	43.5	43.5	26.5	18.0	24.8	28.6	41.6	26.7	27.6	24.4
Re-ranking with cross-encoder										
`bge-m3`	49.4	52.5	32.7	18.5	24.7	31.6	32.9	30.2	30.1	26.8
`bge-gemma`	53.0	54.0	**34.9**	19.5	27.7	33.3	39.4	36.2	34.0	29.7
Fine-tuning (10k data on 1.5B/1.7B models)										
Q2P/SFT	46.6	51.2	30.7	19.7	29.4	38.0	39.7	35.6	33.2	31.1
`gte-Qwen`/ContrastiveFT	57.2	61.4	29.1	18.6	26.4	32.9	39.2	35.5	31.5	29.0
TOOLQP-FORMAT	**63.1**	**66.1**	22.6	17.6	23.4	30.0	32.2	30.1	26.1	25.9
TOOLQP	53.9	59.9	33.0	**23.1**	**32.0**	**41.2**	**45.8**	**43.0**	**36.9**	**35.8**

Table 1: Results on TOOLRET, a benchmark of 35 datasets. Web* denotes the zero-shot Web datasets.

A `Qwen3-1.7B` model fine-tuned to predict initial plans as expansions; and (b) ContrastiveFT: `gte-Qwen` fine-tuned via contrastive learning.

Implementation details are provided in Appx. B. We also include additional experiments that compare to different k for top-k re-ranking and to baselines fine-tuned with a larger amount of data in Appx. C.

TOOLQP. We implemented two versions of TOOLQP, both fine-tuned with the lightweight `Qwen3-1.7B`. The proposed method is denoted as TOOLQP, where we trained on 10k synthetic trajectories generated by `gpt-4.1-mini` (Achiam et al., 2023) based on ToolRet's training set. Additionally, we test whether additional formatting information aids in targeted domains, by additionally including the tool format definition as targets during SFT. This variation is denoted as TOOLQP-FORMAT. Detailed data generation and training settings for both SFT and RLVR can be found in Appx D.

Results. Tab. 1 presents the main retrieval results. TOOLQP and TOOLQP-FORMAT demonstrate superior performance across all settings. On in-domain splits, TOOLQP-FORMAT excels, improving over the base retriever by ∼20% and outperforming ContrastiveFT by ∼6%. This confirms that while standard contrastive fine-tuning improves representations, explicitly modeling tool schemas via generation greatly benefits task-specific applications. Crucially, for zero-shot generalization, TOOLQP achieves the highest performance, surpassing all prompting, fine-tuning, and re-ranking baselines. Notably, TOOLQP outperforms the state-of-the-art cross-encoder `bge-gemma` by ∼3% for N@10 and ∼6% for C@10. This indicates that interactive query planning is more effective at uncovering relevant tools than expensive post-hoc cross-attention re-scoring. Furthermore, TOOLQP achieves these gains while being highly efficient: using a lightweight 1.7B model, it significantly outperforms query expansion methods that rely on prompting much larger (30B) models for multiple passes and avoids the high inference latency of cross-encoder re-ranking.

4.2 Transfer to unseen base retriever

Setup. To validate the robustness of TOOLQP, we evaluate the trained query planner TOOLQP in a transfer setting where it interacts with different tool retriever models. In this setting, TOOLQP is trained with SFT data generated using the base retriever `gte-Qwen2-1.5B-instruct` and further fine-tuned via RLVR with the same retriever, but the query planner would be used at test time with a different retriever, i.e., changing the test environment. This evaluates whether the planned queries capture universal semantic properties or are over-

Method	In-Domain		0-Shot	
	ΔN@10	ΔC@10	ΔN@10	ΔC@10
Q2E	+1.8	+2.2	+2.3	+3.1
Q2P	+1.6	+2.2	+2.2	+3.6
D2Q	+3.1	+3.0	+2.2	+2.2
Re-Invoke	+5.2	+7.3	+4.7	+5.0
TOOLQP-FORMAT	**+10.2**	**+10.4**	+1.7	+4.2
TOOLQP	+6.0	+7.5	**+5.8**	**+7.4**

Table 2: Retriever transfer results on TOOLRET. TOOLQP is trained on `gte-qwen`-generated data, and used out-of-the-box directly with various retrievers at inference time. Average gains for NDCG@10 and Completeness@10 are reported.

fitted to the base retriever's specific embedding space. We evaluate on a diverse set of general and tool-focused embedding models previously benchmarked on ToolRet. We report the averaged gains in nDCG@10 and Completeness@10 over each corresponding base retriever. Detailed breakdowns are provided in Appx. E.

Results. As shown in Tab 2, TOOLQP and TOOLQP-FORMAT consistently outperform baseline expansion methods, even when the inference environment differs from training. TOOLQP-FORMAT achieves ∼9-10% gains in-domain, while TOOLQP obtains ∼5-7% improvements out-of-domain across diverse retrieval models. This confirms that TOOLQP's policy is highly robust and generalizable, providing significant value regardless of the underlying embedding model.

4.3 End-to-end Tool Calling

Benchmark and Setup. To assess the practical utility of TOOLQP, we evaluate end-to-end tool-use performance of `Qwen3-30B` on two standard tool-use benchmarks, using `gte-Qwen` as the retriever. First, we use API-Bank (Li et al., 2023b) (73 tools), testing on the retrieval-focused Level-2 subset and the Level-1 subset configured to force tool search. We report the average accuracy based on the top retrieved tool.

Next, we evaluated on StableToolBench (Guo et al., 2025b) (STB), the stable version of the most widely-used ToolBench (Qin et al., 2024b) in the past few years. We evaluated on the subsets I2-Category (13k tools) and I3-Instruction (1.6k tools). Compared to API-Bank, these tasks are much more complex and compositional, and require multiple steps of reasoning and tool call execution. We follow the official ReAct (Yao et al., 2023) inference pipeline, and report the solvable pass rate.

Retrieval Method	API-Bank		STB	
	`Qwen3-30B`		`Qwen3-30B-ReAct`	
	Level-1	Level-2	I2-Cat	I3-Inst
Base Retriever	47.6	57.8	51.9	40.3
Q2E	48.3	59.5	48.3	38.6
Q2D	46.6	58.5	51.6	40.4
Re-Invoke	41.7	60.5	57.9	28.6
Q2P	48.0	56.1	51.9	41.0
TOOLQP	**53.2**	**60.7**	**60.2**	**48.3**

Table 3: End-to-end tool-calling performance for `Qwen3-30B` with different retrieval methods on API-Bank and StableToolBench (STB). Accuracy is reported for API-Bank and Solvable Pass Rate is reported for StableToolBench.

Results. Tab. 3 highlights the downstream impact of improved retrieval. On the smaller API-Bank, TOOLQP achieves the highest accuracy with a ∼3-6% accuracy gain. This is notable as TOOLQP is designed to operate and excel for large toolsets and not for one as small as 73 tools, which suggests utility even in simpler scenarios. On the larger and more complex STB, TOOLQP achieves substantial gains of ∼8-9% tool-use pass rate over the base retriever. Notably, TOOLQP is the most consistent method; unlike baselines that occasionally degrade performance on some subsets, our planner provides robust context that reliably aids the downstream agent. This is achieved with a lightweight 1.7B planner, and importantly, generalizes well even though STB utilizes a tool embedding format different from which TOOLQP was trained on, showcasing its robustness.

4.4 Ablation Studies

We conduct the following ablation studies to validate our design choices, with results summarized in Tab. 4.

Prompting vs SFT vs RLVR. We first investigate whether explicit training is necessary by comparing TOOLQP against TOOLQP-PROMPTING, a version of our pipeline that uses `Qwen3-30B` to plan queries zero-shot, and further compare against only bootstrapping with synthesized data with SFT. We find that TOOLQP-PROMPTING already achieves big improvements, especially on zero-shot generalization, where it outperforms all prompting baselines shown earlier. This demonstrates the effectiveness and necessity of the query planning framework. However, by fine-tuning with the proposed data generation method, we can further improve performance, especially within the domain. On the

Method	In-Domain		0-Shot	
	N@10	C@10	N@10	C@10
Base Retriever (`gte-Qwen`)	43.3	47.8	29.8	27.0
Prompting vs SFT vs RLVR				
TOOLQP-PROMPTING	45.6	52.4	34.0	33.0
TOOLQP-SFT	52.4	57.9	35.4	34.4
TOOLQP-RLVR	**53.9**	**59.9**	**36.9**	**35.8**
Aggregation Methods				
Reciprocal Rank Fusion	50.9	56.3	35.4	34.5
Multi-view Fusion	54.1	60.2	37.0	35.5
Peak-rank	53.9	59.9	36.9	**35.8**
Cross-encoder (`bge-gemma`)	**58.2**	**62.2**	**37.3**	**35.8**
Retrieval with/without user query				
TOOLQP w/o user query	52.3	57.7	34.8	33.3
TOOLQP	**53.9**	**59.9**	**36.9**	**35.8**
Interactivity vs parallel expansion				
TOOLQP-PARALLEL	-	-	35.3	33.8
TOOLQP	-	-	**36.9**	**35.8**
Trajectory synthesis–excluding failed attempts				
TOOLQP-SFT	-	-	**35.3**	**34.4**
excl. failed att.	-	-	34.8	34.0
TOOLQP-RLVR	-	-	**36.9**	**35.8**
excl. failed att.	-	-	35.8	34.8
TOOLQP-FORMAT *on Similar vs Dissimilar tool format*				
Web*-similar	-	-	+4.3	+6.7
Web*-dissimilar	-	-	-12.3	-3.3

Table 4: Ablation studies.

other hand, we observe that the bulk of in-domain performance gains come from the SFT stage, while RLVR is critical for refining the policy to achieve optimal out-of-domain performance.

Choice of Aggregation Method. Next, we compare our chosen aggregation method, the simple peak-rank fusion, to the widely-used reciprocal rank fusion (RRF) (Cormack et al., 2009), and a multi-view fusion method that joins retrieved lists from extracted user intents (Chen et al., 2024b). Peak-rank outperforms RRF by a wide margin, likely by mitigating the frequency bias for sub-tasks that require more query attempts, and performs on par with the more complex multi-view fusion. We also tested using the `bge-reranker-v2-gemma` cross-encoder for fusing the lists, useful when more compute is available, which further boosts in-domain performance.

Importance of user query. Furthermore, we test whether relying solely on the sub-task queries is sufficient. We find removing that information degrades the performance, since decomposition may be incomplete or erroneous, and including the original user query could act as an anchor. We also note that all query expansion baselines we implemented do include the original query to achieve their best (and reported) results.

Interactivity vs. Decomposition. To isolate the value of decomposition from interactivity, we show the zero-shot performance comparison of TOOLQP versus a trained parallel version – generating plan and sub-goals first, and then generating query for each sub-goal in parallel. The results suggest that the parallel version, while being more efficient (per query), cannot train to utilize feedback or dependency as effectively as the sequential version.

Trajectory Synthesis. We test whether including failed attempts during trajectory synthesis can improve the training of TOOLQP and enable stronger error-correction behavior. We evaluate tool retrieval performance by comparing with a model trained without them. We observe that the bootstrapped SFT performance gap is much smaller than the final RL-tuned one, which suggests failed attempts learned from SFT help steer TOOLQP's error correction capabilities mainly during the RL stage.

Format Injection for TOOLQP Training. TOOLQP-FORMAT is tested as an extension designed specifically for in-domain scenarios where data is sufficient and inference tools follow a strict tool format (the in-domain testing set still contains many unseen tools). We therefore examine under what conditions the additional format information helps the most. For the Web* category, it is both the largest split (16 dataset subsets) and the most heterogeneous, containing tools with documentation ranging from REST API docs to python-defined tools to free-form natural language descriptions and differs from the schema that TOOLQP-FORMAT was trained on. The results are shown at the bottom of Tab. 4, where we separate out the 5 subsets with formats similar to training, which clearly shows that TOOLQP-FORMAT degrades when tested on different tool schemas. Note that the difference is measured against the base retriever. This shows that TOOLQP-FORMAT does not generalize well OOD, but may be quite useful in scenarios where tool formats are guaranteed to match.

Method	Model Size	Runtime (s/q)	Compute (GPU-s/q)	Avg # Retr. Calls	Retrieval (N@10)	E2E Tool-Use (Avg)
Q2E	30B-MoE (4 GPUs)	0.76	3.03	5.0	32.0	48.7
Re-Invoke	30B-MoE (4 GPUs)	**0.57**	2.30	2.1	30.3	47.2
Cross-Enc	2B (1 GPU)	1.82	1.82	**1.0**	34.0	N/A
TOOLQP	**1.7B (1 GPU)**	1.44	**1.44**	1.7	**36.9**	**55.6**

Table 5: Latency Analysis on ToolRet Zero-shot categories. Note for setting: we use 1 (or 4 for 30B) A6000 GPU, tested on a single query input; Q2E sampling is parallel, cross-enc (`bge-gemma`) uses a batch size of 4.

4.5 Qualitative Analysis

Figure 2 visualizes TOOLQP discovering implicit dependencies. The task "modify password" requires a token (via `GetUserToken`), a prerequisite absent from the user query. While baselines fail by focusing solely on keywords, TOOLQP identifies this relation and successfully retrieves the authentication tool alongside the target. This highlights TOOLQP's capacity to uncover latent inter-tool dependencies. Four additional examples taken from the test set are analyzed in Appx. G.

4.6 Latency Analysis

Since TOOLQP operates as tool planner that performs multi-step retrieval, compared to single-shot tool retrieval we incur latency costs and require computation. Thus, to show practicality of TOOLQP, we include the following latency analysis (Tab. 5) on the zero-shot ToolRet test set, comparing our proposed tool planner to common post-retrieval improvement methods. While prompting baselines achieve better runtimes for a single query (by parallelizing for Q2E), they rely on a much larger 30B-parameter MoE model hosted on 4 GPUs due to VRAM constraints compared to TOOLQP. Additionally, when considering the total computational costs, TOOLQP is the most efficient, which allows for much higher concurrent system throughput in practical scenarios. At the same time, we achieve significantly improved tool retrieval and end-to-end tool use performance gains.

User Query	Modify password	
Targets	`ModifyPassword(token,old_password,new_password)` `GetUserToken(username,password)`	
Method	**Query**	**Top-5 Retrieved**
TOOLQP	`<sub_goal>` Modify a user's password by processing the username and new password inputs `</sub_goal>` tool for modifying user password with inputs username string and new password string, password modification, user authentication, password update	`ModifyPassword`, `ForgotPassword`, `RegisterUser`, `GetUserToken`, `OpenBankAccount`
dense	modify password	`ModifyPassword`, `ForgotPassword`, `ModifyAgenda`, `ModifyRegistration`, `OpenBankAccount`
Q2E	tools to modify password, change password, update password, reset password, password management tools, password reset utilities, password change methods, secure password modification tools	`ModifyPassword`, `ForgotPassword`, `ModifyAgenda`, `ModifyAlarm`, `OpenBankAccount`
Q2D	`{"name": "modify_password", "description": "A tool to update or change a user's password.", "parameters": {"new_password": {"description": "The new password to set for the user.", "type": "string"}, ...`	`ModifyPassword`, `ForgotPassword`, `ModifyAgenda`, `ModifyAlarm`, `RegisterUser`
Q2P	Given a 'password modification' task, retrieve tools that allow users to change their passwords by processing the username and new password inputs	`ModifyPassword`, `ForgotPassword`, `ModifyAgenda`, `ModifyRegistration`, `OpenBankAccount`
Re-Invoke	modify password	`ModifyPassword`, `ForgotPassword`, `ModifyAlarm`, `ModifyAgenda`, `ModifyReminder`

Figure 2: TOOLQP uncovers a latent tool dependency.

5 Conclusion

We introduce TOOLQP, a framework that formulates tool retrieval as an iterative planning process. By decomposing instructions and leveraging interactive feedback, TOOLQP bridges the semantic gap between abstract user intents and technical tool documentation. This is enabled through a robust training pipeline using synthetic trajectories and RLVR. Empirically, TOOLQP outperforms state-of-the-art baselines on ToolRet, demonstrating superior zero-shot generalization. Furthermore, our analysis confirms that the learned planning policy is highly robust, transferring effectively to unseen retrieval models and significantly improving success rates in downstream end-to-end agentic tasks. Lightweight and modular, TOOLQP offers a scalable solution for robust agents operating over massive tool libraries.

Limitations

Our synthetic data generation utilized a single base retriever due to compute constraints, implicitly biasing the planner to a specific embedding space; future work should explore multi-retriever synthesis to foster a more robust, agnostic policy. Similarly, for the trajectory synthesis teacher model, we went with a small gpt-4.1-mini rather than a larger, more expensive frontier model; while we achieved substantial performance gains, a systematic study could be beneficial. Additionally, TOOLQP resolves inter-tool dependencies implicitly via interaction, whereas integrating explicit Tool Knowledge Graphs could guide the planning process more efficiently than trial-and-error. In terms of scope, our framework targets massive, dynamic tool libraries, meaning the complexity of iterative planning may be unnecessary for smaller toolsets compared to direct prompting. Finally, developing an adaptive planning mechanism that dynamically bypasses or simplifies decomposition for simple queries would further optimize the latency-performance trade-off.

Ethics Statement

We used publicly available models and datasets for training and evaluation, and did not collect data or any personal information in this work. Self-generated data by these public models were used to train a generation model, which has a chance of producing harmful or hallucinated content, even when generation is conditioned on curated public data inputs and filtered with ground truth targets from public data. They could potentially be misused and pose ethical risks typical of large language models when deployed in real-world applications, if not thoroughly audited.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

William Brown. 2025. Verifiers: Environments for llm reinforcement learning. https://github.com/PrimeIntellect-ai/verifiers.

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*.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024a. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2318–2335, Bangkok, Thailand. Association for Computational Linguistics.

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.

Yung-Sung Chuang, Wei Fang, Shang-Wen Li, Wen-tau Yih, and James Glass. 2023. Expand, rerank, and retrieve: Query reranking for open-domain question answering. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 12131–12147, Toronto, Canada. Association for Computational Linguistics.

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.

W.B. Croft and D.J. Harper. 1979. Using probabilistic models of document retrieval without relevance information. *Journal of Documentation*, 35(4):285–295.

Zhuyun Dai and Jamie Callan. 2019. Context-aware sentence/passage term importance estimation for first stage retrieval. *arXiv preprint arXiv:1910.10687*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Wei Fang, Yung-Sung Chuang, and James Glass. 2024. Joint inference of retrieval and generation for passage re-ranking. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 2289–2298, St. Julian's, Malta. Association for Computational Linguistics.

Wei Fang, Yang Zhang, Kaizhi Qian, James R. Glass, and Yada Zhu. 2025. PLAY2PROMPT: Zero-shot tool instruction optimization for LLM agents via tool play. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 26274–26290, Vienna, Austria. Association for Computational Linguistics.

Jiazhan Feng, Chongyang Tao, Xiubo Geng, Tao Shen, Can Xu, Guodong Long, Dongyan Zhao, and Daxin

Jiang. 2024. Synergistic interplay between search and large language models for information retrieval. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9571–9583, Bangkok, Thailand. Association for Computational Linguistics.

Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. Splade: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 2288–2292, New York, NY, USA. Association for Computing Machinery.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2023. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1762–1777, Toronto, Canada. Association for Computational Linguistics.

Shen Gao, Zhengliang Shi, Minghang Zhu, Bowen Fang, Xin Xin, Pengjie Ren, Zhumin Chen, Jun Ma, and Zhaochun Ren. 2024. Confucius: Iterative Tool Learning from Introspection Feedback by Easy-to-Difficult Curriculum. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):18030–18038.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025a. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Zhicheng Guo, Sijie Cheng, Yuchen Niu, Hao Wang, Sicheng Zhou, Wenbing Huang, and Yang Liu. 2025b. StableToolBench-MirrorAPI: Modeling tool environments as mirrors of 7,000+ real-world APIs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 5247–5270, Vienna, Austria. Association for Computational Linguistics.

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

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. 2023. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675*.

Rolf Jagerman, Honglei Zhuang, Zhen Qin, Xuanhui Wang, and Michael Bendersky. 2023. Query Expansion by Prompting Large Language Models. *arXiv preprint*. ArXiv:2305.03653 [cs].

Yilun Kong, Jingqing Ruan, YiHong Chen, Bin Zhang, Tianpeng Bao, Shi Shiwei, du Guo Qing, Xiaoru Hu, Hangyu Mao, Ziyue Li, Xingyu Zeng, Rui Zhao, and Xueqian Wang. 2024. TPTU-v2: Boosting task planning and tool usage of large language model-based agents in real-world industry systems. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 371–385, Miami, Florida, US. Association for Computational Linguistics.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, and 1 others. 2024. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.

Victor Lavrenko and W. Bruce Croft. 2001. Relevance based language models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, page 120–127, New York, NY, USA. Association for Computing Machinery.

Patrick Lewis, Yuxiang Wu, Linqing Liu, Pasquale Minervini, Heinrich Küttler, Aleksandra Piktus, Pontus Stenetorp, and Sebastian Riedel. 2021. PAQ: 65 million probably-asked questions and what you can do with them. *Transactions of the Association for Computational Linguistics*, 9:1098–1115.

Hang Li, Ahmed Mourad, Shengyao Zhuang, Bevan Koopman, and Guido Zuccon. 2023a. Pseudo relevance feedback with deep language models and dense retrievers: Successes and pitfalls. *ACM Trans. Inf. Syst.*, 41(3).

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023b. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023c. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: A python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 2356–2362, New York, NY, USA. Association for Computing Machinery.

Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, Jun Wang, and Weinan Zhang. 2025. Robust function-calling for on-device language model via function masking. In *The Thirteenth International Conference on Learning Representations*.

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

Weiwen Liu, Xu Huang, Xingshan Zeng, xinlong hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Wang Xinzhi, Yong Liu, Yasheng Wang, and 8 others. 2025. ToolACE: Winning the points of LLM function calling. In *The Thirteenth International Conference on Learning Representations*.

Zuxin Liu, Thai Quoc Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh R N, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. 2024b. APIGen: Automated PIpeline for generating verifiable and diverse function-calling datasets. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*.

Xuan Lu, Haohang Huang, Rui Meng, Yaohui Jin, Wenjun Zeng, and Xiaoyu Shen. 2025. Tools are underdocumented: Simple document expansion boosts tool retrieval. *arXiv preprint arXiv:2510.22670*.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5303–5315, Singapore. Association for Computational Linguistics.

Iain Mackie, Shubham Chatterjee, and Jeffrey Dalton. 2023. Generative relevance feedback with large language models. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 2026–2031, New York, NY, USA. Association for Computing Machinery.

Kelong Mao, Zhicheng Dou, Fengran Mo, Jiewen Hou, Haonan Chen, and Hongjin Qian. 2023. Large language models know your contextual search intent: A prompting framework for conversational search. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1211–1225, Singapore. Association for Computational Linguistics.

Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. 2021. Generation-augmented retrieval for open-domain question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4089–4100, Online. Association for Computational Linguistics.

M. E. Maron and J. L. Kuhns. 1960. On relevance, probabilistic indexing and information retrieval. *J. ACM*, 7(3):216–244.

Grégoire Mialon, Roberto Dessi, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru, Roberta Raileanu, Baptiste Roziere, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. 2023. Augmented language models: a survey. *Transactions on Machine Learning Research*. Survey Certification.

Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. 2019. Document expansion by query prediction. *arXiv preprint arXiv:1904.08375*.

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

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

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, and 22 others. 2024a. Tool learning with foundation models. *Preprint*, arXiv:2304.08354.

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

Yonggang Qiu and Hans-Peter Frei. 1993. Concept based query expansion. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '93, page 160–169, 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. 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, pages 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. From exploration to mastery: Enabling LLMs to master tools via self-driven interactions. In

The Thirteenth International Conference on Learning Representations.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

J. J. Jr. Rocchio. 1971. Relevance feedback in information retrieval. *The SMART Retrieval System : Experiments in Automatic Document Processing*.

Devendra Sachan, Mike Lewis, Mandar Joshi, Armen Aghajanyan, Wen-tau Yih, Joelle Pineau, and Luke Zettlemoyer. 2022. Improving passage retrieval with zero-shot question generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3781–3797, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Saptarshi Sengupta, Zhengyu Zhou, Jun Araki, Xingbo Wang, Bingqing Wang, Suhang Wang, and Zhe Feng. 2025. Tooldreamer: Instilling llm reasoning into tool retrievers. *arXiv preprint arXiv:2510.19791*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Tao Shen, Guodong Long, Xiubo Geng, Chongyang Tao, Tianyi Zhou, and Daxin Jiang. 2023a. Large language models are strong zero-shot retriever. *arXiv preprint arXiv:2304.14233*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023b. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*.

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.

Amit Singhal and Fernando Pereira. 1999. Document expansion for speech retrieval. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, page 34–41, New York, NY, USA. Association for Computing Machinery.

Yifan Song, Weimin Xiong, Dawei Zhu, Cheng Li, Ke Wang, Ye Tian, and Sujian Li. 2023. Restgpt: Connecting large language models with real-world applications via restful apis. *arXiv preprint arXiv:2306.06624*.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT good at search? investigating large language models as re-ranking agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14918–14937, Singapore. Association for Computational Linguistics.

Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. Vipergpt: Visual inference via python execution for reasoning. *arXiv preprint arXiv:2303.08128*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, and 1 others. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, and 1 others. 2022. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*.

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 preprint arXiv:2212.03533*.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. Improving text embeddings with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11897–11916, Bangkok, Thailand. Association for Computational Linguistics.

Liang Wang, Nan Yang, and Furu Wei. 2023. Query2doc: Query expansion with large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9414–9423, Singapore. Association for Computational Linguistics.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. 2025. Toolgen: Unified tool retrieval and calling via generation. In *The Thirteenth International Conference on Learning Representations*.

Xiao Wang, Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2021. Pseudo-relevance feedback for multiple representation dense retrieval. In *Proceedings of the 2021 ACM SIGIR International Conference on Theory of Information Retrieval*, ICTIR '21, page 297–306, New York, NY, USA. Association for Computing Machinery.

Orion Weller, Michael Boratko, Iftekhar Naim, and Jinhyuk Lee. 2025. On the Theoretical Limitations of Embedding-Based Retrieval. *arXiv preprint*. ArXiv:2508.21038 [cs].

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

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding. *Preprint*, arXiv:2309.07597.

Jinxi Xu and W. Bruce Croft. 1996. Query expansion using local and global document analysis. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '96, page 4–11, 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.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023. On the tool manipulation capability of open-source large language models. *arXiv preprint arXiv:2305.16504*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. Gpt4tools: Teaching large language model to use tools via self-instruction. *Advances in Neural Information Processing Systems*, 36.

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*.

Fanghua Ye, Meng Fang, Shenghui Li, and Emine Yilmaz. 2023. Enhancing conversational search: Large language model-aided informative query rewriting. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5985–6006, Singapore. Association for Computational Linguistics.

HongChien Yu, Chenyan Xiong, and Jamie Callan. 2021. Improving query representations for dense retrieval with pseudo relevance feedback. In *Proceedings of the 30th ACM International Conference on Information &; Knowledge Management*, CIKM '21, page 3592–3596, New York, NY, USA. Association for Computing Machinery.

Shi Yu, Jiahua Liu, Jingqin Yang, Chenyan Xiong, Paul Bennett, Jianfeng Gao, and Zhiyuan Liu. 2020. Few-shot generative conversational query rewriting. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 1933–1936, New York, NY, USA. Association for Computing Machinery.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. 2025. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Tianhua Zhang, Jiaxin Ge, Hongyin Luo, Yung-Sung Chuang, Mingye Gao, Yuan Gong, Yoon Kim, Xixin Wu, Helen Meng, and James Glass. 2024. Natural language embedded programs for hybrid language symbolic reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4131–4155, Mexico City, Mexico. Association for Computational Linguistics.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. 2024. Toolchain*: Efficient action space navigation in large language models with a* search. In *The Twelfth International Conference on Learning Representations*.

A Dataset Details and Licenses

We use ToolRet for tool retrieval evaluation, which aggregates 35 previously published datasets; full dataset statistics are documented in Shi et al. (2025). ToolRet is licensed under Apache License 2.0. For end-to-end tool use evaluation, we used API-Bank (Li et al., 2023b) and StableToolBench (Guo et al., 2025b), which are licensed under MIT License and Apache License 2.0, respectively.

B Baseline Implementation Details and Prompts

We include the implementation details for baselines used in our evaluation in Tab. 6. We include all prompts at the end of the appendices.

C Additional Baseline Configurations on ToolRet

In additional to the main experimental results, we compare TOOLQP to different re-ranking settings and to fine-tuning with additional data. Specifically for re-ranking, we re-rank the top-100 retrieved tools with the same cross-encoders instead of 20. For fine-tuning, we strengthen our baselines Q2P and Contrastive Fine-tuning by training on the full ToolRet training set, with 200k instances. The results are shown in Table 7.

Baseline	Method	Base Model	Details	Prompts
Q2E (Jagerman et al., 2023)	ZS/PRF Prompting	`Qwen3-30B`	Dense fusion of $N = 5$ query expansions concatenated to user query.	See Figs. 8, 9
Q2D (Wang et al., 2023)	ZS/PRF Prompting	`Qwen3-30B`	Dense fusion of $N = 5$ query expansions (hypothetical documents) concatenated to user query.	See Figs. 10, 11
HyDE (Gao et al., 2023)	ZS/PRF Prompting	`Qwen3-30B`	Dense fusion of $N = 5$ hypothetical documents *and* user query.	See Figs. 10, 11
D2Q (Nogueira et al., 2019)	ZS Prompting	`Qwen3-30B`	Averaged embeddings of $N = 10$ hypothetical queries concatenated to tool documentation.	See Fig. 12
Re-Invoke (Chen et al., 2024b)	FS Prompting	`Qwen3-30B`	Same embeddings as D2Q.	See Figs. 12, 13
Q2P	Supervised Fine-tuning	`Qwen3-1.7B`	10k ToolRet-train data; trained with Huggingface TRL's `SFTTrainer` with the following configurations: 3 epochs, warmup ratio 0.03, batch size 64, learning rate $2e-5$	N/A
`gte-Qwen/ContrastiveFT`	Noise-contrastive Fine-tuning	`gte-Qwen`	10k ToolRet-train data; trained with FlagEmbedding with the following configuration: `decoder_only`, `train_group_size=8`, `learning_rate=1e-6`, `num_train_epochs=2`, `warmup_ratio=0.1`.	N/A
TOOLQP-PROMPTING	ZS Prompting	`Qwen3-30B`	Sampling temperature set to 0 for planning and 0.5 for query generation.	See Fig. 14

Table 6: Baseline details.

Method	In-Domain		Zero-Shot Transfer							
	Avg		Web*		Code		Custom		Macro-Avg	
	N@10	C@10	N@10	C@10	N@10	C@10	N@10	C@10	N@10	C@10
Base Retriever (`gte-Qwen`)	43.3	47.8	29.7	17.8	24.1	30.8	35.6	32.4	29.8	27.0
Re-ranking with cross-encoder (top-20)										
`bge-m3`	49.4	52.5	32.7	18.5	24.7	31.6	32.9	30.2	30.1	26.8
`bge-gemma`	53.0	54.0	**34.9**	19.5	27.7	33.3	39.4	36.2	34.0	29.7
Re-ranking with cross-encoder (top-100)										
`bge-m3`	51.3	55.5	31.0	19.5	24.2	30.7	30.9	27.5	28.7	25.9
`bge-gemma`	56.9	60.2	34.6	21.3	28.1	35.0	39.1	37.9	33.9	31.4
Fine-tuning (10k data on 1.5B/1.7B models)										
Q2P/SFT	46.6	51.2	30.7	19.7	29.4	38.0	39.7	35.6	33.2	31.1
`gte-Qwen/ContrastiveFT`	57.2	61.4	29.1	18.6	26.4	32.9	39.2	35.5	31.5	29.0
TOOLQP-FORMAT	63.1	66.1	22.6	17.6	23.4	30.0	32.2	30.1	26.1	25.9
TOOLQP	53.9	59.9	33.0	**23.1**	**32.0**	**41.2**	**45.8**	**43.0**	**36.9**	**35.8**
Fine-tuning (Full 200k ToolRet data on 1.5B/1.7B models)										
Q2P/SFT	47.0	51.1	26.5	18.0	29.2	37.0	40.7	37.4	33.6	31.7
`gte-Qwen/ContrastiveFT`	**68.2**	**71.5**	30.4	22.4	29.8	36.5	43.5	41.4	34.6	33.4

Table 7: Additional results on TOOLRET. Web* denotes the zero-shot Web datasets. nDCG@10 is denoted as N@10, and Completeness@10 is denoted as C@10.

Comparing re-ranking top-20 against top-100, we observe that while in-domain retrieval performance are improved with the additional 5x compute, for zero-shot generalization it instead

harmed performance, and significantly underperforms TOOLQP. When fine-tuning on 20x the amount of data, Q2P shows a very slight improvement while ConstrastiveFT shows more gains. In domain, although it achieves the best performance, it is much less data-efficient than TOOLQP-FORMAT, and also much more compute-heavy with contrastive losses instead of standard negative likelihood training. For generalizability, it still underperforms TOOLQP trained on much less data, further showcasing the effectiveness and efficiency of TOOLQP.

D TOOLQP Training Details

D.1 Data

For training, we subsampled the training data provided in Shi et al. (2025), which is compiled from 3 datasets: ToolBench (Qin et al., 2024b), ToolACE (Liu et al., 2025), and APIGen (Liu et al., 2024b). The original training set contains roughly 200k instances, which we subsample and select 10k for data generation and training.

D.2 Query Sequence Generation

Detailed illustration of the algorithm used to generate query trajectories can be found in Alg. 1. Probability p of keeping failed attempts is set to 0.4, and the rank threshold r_τ is set to 5. `gpt-4.1-mini` is used as the teacher model with default sampling parameters.

In Fig. 7, we show an example query trajectory sequence, which is used for SFT with maximum likelihood with teacher forcing. Prompts for the teacher model used in the described algorithm can be found in Fig. 15 at the end of appendices.

D.3 SFT

We train TOOLQP and TOOLQP-FORMAT with Huggingface TRL's `SFTTrainer`, with the following configurations: 3 epochs, learning rate of 2e-5, 150 warmup steps with cosine schedule, batch size of 32, weight decay of 0.01, and a maximum sequence length of 16384. We include the top $k = 5$ retrieved tools as retrieval feedback for teacher forcing.

D.4 RLVR

We used the `verifiers` library to implement GRPO training (Brown, 2025) since it supported multi-step rollout and training. We use a setup of: 500 training steps, learning rate of 1e-6, (effective) batch size of 256, early stopping with dev set, warmup steps of 143, kl regularization of 0.02, group size of 4, μ set to 2, top $k = 5$ feedback per turn with 10 maximum query turns.

For reward shaping, we define $\mathcal{R}_{\text{retrieval}} = \beta_{1,n}\Delta N + \beta_{1,r}\Delta R$, where ΔN is the nDCG@5 difference between the final aggregated retrieval from the sequence and the baseline retrieval with $\text{Concat}(q, \mathcal{P})$, and similarly defined for recall difference ΔR. The format reward $\mathcal{R}_{\text{format}} = \beta_{2,f}R_f + \beta_{2,s}R_s$ consists of R_f, the fraction of responses in correct formatting, and R_s, whether rollout sequence ends in the `<stop_retrieval>` (end of query generation) token. Finally, $\mathcal{R}_{\text{plan}} = \text{Sim}(P, \mathcal{P})$ is the cosine similarity score between the first turn plan prediction P and reference plan $\mathcal{P}$ using an embedding model. We use `gte-Qwen` for convenience. The reward weights are set to: $\beta_{1,n} = 5.0, \beta_{1,r} = 2.5, \beta_{2,f} = 1.5, \beta_{2,s} = 0.6, \beta_3 = 1.0$.

E Details for Retriever Transfer Evaluation

For retriever transfer evaluation, we test on 5 widely-used embedding models previously tested on ToolRet: `bge-large-en-v1.5`, `e5-base-v2`, `e5-mistral-7b-instruct`, `ToolRet-bge-large-en-v1.5` and `ToolRet-e5-base-v2`, with the last two being fine-tuned versions of the first two models using the full 200k training data in Shi et al. (2025) by contrastive training. We also include evaluation for the widely-used lexical retriever BM25 (Robertson and Zaragoza, 2009). Details for each model are provided in Tab. 8. In the main text, we reported the averaged improvements of all 5 models for each baseline method and TOOLQP. Additionally, we provide in Tab. 9 detailed results for each model-method pairing for each category in ToolRet.

F Details for End-to-end Tool Use Evaluation

We use the official evaluation pipelines for both API-Bank and StableToolBench. For both datasets, we report results with the average of 3 runs. `gte-Qwen` is used as the retriever, while all baselines are the same as previous settings. The fine-tuned models, including TOOLQP, are tested directly on the datasets without further adaptation.

For API-Bank, level-2 is used as-is. For level-1,

Model	Reference	#Params	Source
`gte-Qwen2-1.5B-Instruct`	Li et al. (2023c)	1.5B	Alibaba-NLP/gte-Qwen2-1.5B-instruct
`bge-large-en-v1.5`	Xiao et al. (2023)	335M	BAAI/bge-large-en-v1.5
`ToolRet-bge-large-en-v1.5`	Shi et al. (2025)	335M	mangopy/ToolRet-trained-bge-large-en-v1.5
`e5-base-v2`	Wang et al. (2022)	110M	intfloat/e5-base-v2
`ToolRet-e5-base-v2`	Shi et al. (2025)	110M	mangopy/ToolRet-trained-e5-base-v2
`e5-mistral-7b-instruct`	Wang et al. (2024)	7B	intfloat/e5-mistral-7b-instruct
BM25	Robertson and Zaragoza (2009)	-	Pyserini (Lin et al., 2021)

Table 8: Base retriever details.

the standard evaluation is unrealistic since it gives the list of all relevant tools used for a dialogue at the beginning of dialogue, making it much easier for the LLM to choose from. We therefore modify it such that the tool list is replaced with a retrieved top-5 list using the initial user query, which is standard in tool retrieval pipelines, including the STB evaluation below. The tool retrieval step is forced as a required step before the LLM acts in this case. We report the average accuracy based on exact tool-use correctness.

For StableToolBench, we use `Qwen-30B` as the ToolEval LLM judge, and the `stabletoolbench/MirrorAPI-Cache` model as the tool simulation model. We follow the official inference script default and set $k = 5$ to include the 5 top-retrieved tools.

G Additional Qualitative Studies

In the main text, we showed an example of TOOLQP successfully retrieving all relevant tools and discovering implicit tool dependencies compared to other methods. Here, we present four additional examples taken from ToolRet's test set. Figs. 3 and 4 show examples of TOOLQP correctly finding relevant tools by planning a single sub-goal and generating one query. For the former, TOOLQP managed to focus on general books and include crucial keywords such as "publication year" and "genre" that effectively guided the retriever, while other methods mostly retrieved Bible-related tools due to the frequent use of the term "book" in Bible-related tool names and documentations. For the latter, the expansion baselines largely fail to retrieve the correct targets as they focus on the high-level "calculate number of trees planted" and retrieved quite a few plant-related or calculation tools; TOOLQP, on the other hand, accurately finds the target tool by articulating the plan correctly and generating queries that target such a calculation tool specifically for organizations. In Figs. 5 and 6, we analyze and compare TOOLQP to different methods on complex queries that require more fine-grained task decomposition. In the former case, TOOLQP decomposes the user query into two distinct sub-goals, and retrieves all three targets perfectly; most other expansion methods fail, in comparison, and focus on tools related to the topic "exercise" or generic video-related tools, due to the complexity of the user query and the usage of only one single combined query for representation; and while Re-Invoke decomposes the task into three queries, it fails to find relevant tools for two out of the three query attempts. For the final example, all other expansion baselines other than Re-Invoke retrieved only RL-related models endpoints rather than the research paper and repositories the user requested; Re-Invoke manages to find paper-related tools (although not labeled as targets), but fails to rank repository-searching tools high. In comparison, TOOLQP's sub-goals and queries are much more detailed and targeted, and results in accurate retrieval. We also note that TOOLQP spends an extra step for the second sub-goal to confirm, likely due to uncertainty in the feedback, using a slight modification of the query in the previous turn; in this case, our aggregation algorithm does not bias towards tools that appear in more turns, as designed, so that the top-5 are not populated with just GitHub-related tools.

User Query	Can I have a summary of a thriller book by Dan Brown that was published in 2003?
Targets	`BookFinder(genre, author, year), BookSummary(book_name)`

Method	**Query**	**Top-5 Retrieved**

TOOLQP	`<sub_goal> Retrieve a summary of a thriller book by a specified author and publication year. </sub_goal&gt>` tool for retrieving book summary by author and publication year, parameters: author name, publication year, category: book summary, genre : thriller	`BookSummary(book_name)`
		`TopBooks(author_info, num_of_books)`
		`Bible-Memory-Verse-Flashcard-Search-Term-Verse-Address-Summary(term1)`
		`BookFinder(genre, author, year)`
		`BookTitle(author, genre)`

dense	Can I have a summary of a thriller book by Dan Brown that was published in 2003?	`BibleSearch.GetChapterbyBookName(bookName, chapterId)`
		`GetChapter(Book, chapter)`
		`UncoveredTreasureVerse.verse(verse)`
		`GetBookDetails(bookName)`
		`CompleteStudyBible.FullChapterAPI(chapter, translation, book)`

Q2E	summary of a thriller book by Dan Brown published in 2003	`GetBookDetails(bookName)`
		`BibleSearch.GetChapterbyBookName(bookName, chapterId)`
		`UncoveredTreasureVerse.verse(verse)`
		`HolyBible.GetChapter(Book, chapter)`
		`BibleSearch.GetBookByName(bookName)`

Q2D	`{"name": "get_book_summary", "description": "Retrieves a summary of a thriller book by Dan Brown published in 2003.", "parameters": {"book_title": {"description": "The title of the book by Dan Brown published in 2003.", "type": "string"}}}`	`BookSummary(book_name)`
		`GetBookDetails(bookName)`
		`BibleSearch.GetBookByName(bookName)`
		`BookTitle(author, genre)`
		`GetBookInformation(book_Id)`

Q2P	Given a 'book summary' task, retrieve tools that extract summaries based on book title and publication year by processing these inputs to provide the requested information.	`Lead-3(text)`
		`BookSummary(book_name)`
		`GetBookDetails(bookName)`
		`find_page_number()`
		`SearchTerm.ChapterAddressSummary(first_book, second_book, term1)`

Re-Invoke	summarize a thriller book by Dan Brown published in 2003	`BookInspector()`
		`find_page_number()`
		`generateThrillerPlot(protagonist, antagonist)`
		`IdentifyBook(title)`
		`FindBook(title)`

Figure 3: Retrieval comparison for example taken from ToolRet's test set.

User Query	An environmental agency examined how many trees were planted by different organizations. In all, how many trees were planted by Let it Grow and Heal the Earth?
Targets	`calculate_total_items_for_organizations(df, organization_col, item_count_col, organization_list)`

Method	**Query**	**Top-5 Retrieved**

TOOLQP	`<sub_goal> Calculate the total number of trees planted by specific organizations by processing the organization names as inputs. </sub_goal&gt>` tool to calculate total number of trees planted by specific organizations, input parameters: organization names, output: total tree count, category: environmental data, tree planting statistics	`calculate_total_items_for_organizations(df, organization_col, item_count_col, organization_list)`
		`calculate_num_trees(yard_length, distance_between_trees)`
		`count_trees(leaves, threshold)`
		`count_gardens(plants)`
		`count_leaves_1_to_7(plot)`

dense	An environmental agency examined how many trees were planted by different organizations. In all, how many trees were planted by Let it Grow and Heal the Earth?	`IndoorPlants()`
		`PlantRecommender()`
		`calculate_num_trees(yard_length, distance_between_trees)`
		`GetAllPlants()`
		`count_gardens(plants)`

Q2E	Total number of trees planted by Let it Grow and Heal the Earth combined	`calculate_num_trees(yard_length, distance_between_trees)`
		`IndoorPlants()`
		`CarbonFootprint_TreeEquivalent(weight, unit)`
		`PlantRecommender()`
		`count_gardens(plants)`

Q2D	`{"name": "sum_trees_planted", "description": "Calculates the total number of trees planted by two specified organizations.", "parameters": {"organization1": {"description": "The name of the first organization.", "type": "string"}, "organization2": {"description": "The name of the second organization.", "type": "string"}}}`	`calculate_num_trees(yard_length, distance_between_trees)`
		`calculate_sum(num1, num2)`
		`CarbonFootprint_TreeEquivalent(weight, unit)`
		`calculateSum(num1, num2)`
		`treeequivalent(weight, unit)`

Q2P	Given a 'tree planting query', retrieve tools that can provide the number of trees planted by specific organizations by processing the organization name as input.	`CharityTool()`
		`PlantRecommender()`
		`GetAllPlants()`
		`calculate_total_items_for_organizations(df, organization_col, item_count_col, organization_list)`
		`calculate_num_trees(yard_length, distance_between_trees)`

Re-Invoke	determine the total number of trees planted by Let it Grow
	determine the total number of trees planted by Heal the Earth
	calculate the combined total number of trees planted by Let it Grow and Heal the Earth
		`calculate_num_trees(yard_length, distance_between_trees)`
		`CarbonFootprint_TreeEquivalent(weight, unit)`
		`count_gardens(plants)`
		`calculate_total_germination_rate(seeds_1, seeds_2, germination_rate_1, germination_rate_2)`
		`case1_count()`

Figure 4: Retrieval comparison for example taken from ToolRet's test set.

User Query	I have an article about the benefits of daily exercise. I want to rewrite the article to make it unique, expand the content to include more detailed descriptions, and find a related video. Here's the original article text: 'Regular exercise is essential for maintaining physical and mental health. It can improve mood, increase energy levels, and help prevent various diseases.'
Targets	`ArticleSpinner(text)`, `TextExpander(text)`, `VideoSearch(text)`

Method	**Query**	**Top-5 Retrieved**

TOOLQP	`<sub_goal> Rewrite and expand article content based on the original article text</sub_goal&gt>`
	tool for rewriting and expanding article content, input: article text string, output: rewritten and expanded unique article text, content rewriting, article expansion, text processing
	`<sub_goal> Find related videos based on the article content. </sub_goal&gt>`
	tool for finding related videos based on article content, input: article text string, output: related video list, video retrieval, content-based video search, article content video linking
		`ArticleSpinner(text)`
		`VideoSummarizeTool(link)`
		`VideoSearch(text)`
		`TextExpander(text)`
		`TextParaphraser(text)`

dense	I have an article about the benefits of daily exercise. I want to rewrite the article to make it unique, expand the content to include more detailed descriptions, and find a related video. Here's the original article text: 'Regular exercise is essential for maintaining physical and mental health. It can improve mood, increase energy levels, and help prevent various diseases.'	`exercise.video_library(category, difficulty_level)`
		`Planfit()`
		`getAllExercises()`
		`getExerciseDetails(id)`
		`v1_exercises(offset, muscle, type, difficulty, name)`

Q2E	Rewrite and expand the article on the benefits of daily exercise with detailed descriptions, and find a relevant educational video on the topic.	`exercise.video_library(category, difficulty_level)`
		`Planfit()`
		`VideoSummarizeTool(link)`
		`video_highlight(link)`
		`getAllExercises()`

Q2D	`{"name": "rewrite_and_enhance_content", "description": "Rewrites the provided article to make it unique, expands the content with detailed descriptions, and includes a related video recommendation.", "parameters": {"original_text": {"description": "The original article text about the benefits of daily exercise.", "type": "string"}}}`	`exercise.video_library(category, difficulty_level)`
		`Planfit()`
		`video_highlight(link)`
		`VideoSpeedChanger(video, text)`
		`getExerciseDetails(id)`

Q2P	Given a 'content rewriting and video search' task, retrieve tools that can rewrite and expand text content while also finding related videos based on the given topic and context. The tools should process the original article text and return the rewritten content and a list of related videos.	`exercise.video_library(category, difficulty_level)`
		`ArticleSpinner(text)`
		`video_highlight(link)`
		`Planfit()`
		`VideoSummarizeTool(link)`

Re-Invoke	rewrite the article about the benefits of daily exercise to make it unique
	expand the article with more detailed descriptions
	find a related video to accompany the article
		`DescribeEnvironmentAndActivityBenefits(image_patch)`
		`YouTube_v3.v3_Search(part)`
		`TextExpander(text)`
		`inquire_benefits_of_activity(activity, group)`
		`play_youtube_video(video_id)`

Figure 5: Retrieval comparison for example taken from ToolRet's test set.

User Query	I need resources for understanding "reinforcement learning" techniques, can you suggest some research papers and GitHub repositories?
Targets	`ResearchFinder, RepoTool`

Method	**Query**	**Top-5 Retrieved**

TOOLQP	`<sub_goal> Retrieve research papers related to a specific topic by processing the topic as input.</sub_goal&gt>`
	tool for retrieving research papers related to a specific topic, input topic string, output research papers, academic, paper database, paper search, literature retrieval
	`<sub_goal> Suggest GitHub repositories related to a specific topic by processing the topic as input. </sub_goal&gt>`
	tool for suggesting GitHub repositories related to a specific topic, input topic string, output GitHub repositories, repository search, code repository, GitHub API, research resources
	tool for suggesting GitHub repositories related to a specific topic, input topic string, output GitHub repositories, GitHub API, repository search, code repository, research resources
		`sb3/dqn-Acrobot-v1`
		`RepoTool`
		`ppo-BreakoutNoFrameskip-v4`
		`GitHub`
		`ResearchFinder`

dense	I need resources for understanding "reinforcement learning" techniques, can you suggest some research papers and GitHub repositories?	`ppo-BreakoutNoFrameskip-v4`
		`ppo-PongNoFrameskip-v4`
		`ppo-Pendulum-v1`
		`sb3/ppo-CartPole-v1`
		`sb3/dqn-MountainCar-v0`

Q2E	research papers on reinforcement learning techniques, GitHub repositories for reinforcement learning, open-source reinforcement learnin g projects, academic papers on RL algorithms, best reinforcement learning implementations on GitHub	`ppo-PongNoFrameskip-v4`
		`ppo-BreakoutNoFrameskip-v4`
		`ppo-Pendulum-v1`
		`sb3/ppo-CartPole-v1`
		`sb3/dqn-Acrobot-v1`

Q2D	`{"name": "research_and_code_resources", "description": "A tool that suggests key research papers and GitHub repositories for understanding reinforcement learning techniques.", "parameters": {"topic": {"description": "The specific topic within reinforcement learning to focus on, such as 'Q-learning', 'deep reinforcement learning', 'policy gradients', etc.","type": "string"}}}`	`ppo-Pendulum-v1`
		`ppo-PongNoFrameskip-v4`
		`ppo-BreakoutNoFrameskip-v4`
		`sb3/ppo-CartPole-v1`
		`sb3/dqn-Acrobot-v1`

Q2P	Given a 'research resource retrieval' task, retrieve tools that can provide research papers and GitHub repositories related to a specific topic, such as "reinforcement learning," by processing the topic input and returning relevant resources.	`ppo-Pendulum-v1`
		`ppo-PongNoFrameskip-v4`
		`ppo-BreakoutNoFrameskip-v4`
		`sb3/ppo-CartPole-v1`
		`sb3/dqn-MountainCar-v0`

Re-Invoke	find research papers on reinforcement learning techniques
	find GitHub repositories related to reinforcement learning techniques
		`ppo-Pendulum-v1`
		`ArxivSearch.get_arxiv_article_information(query)`
		`ppo-BreakoutNoFrameskip-v4`
		`PaperAnalyzer`
		`ppo-seals-CartPole-v0`

Figure 6: Retrieval comparison for example taken from ToolRet's test set.

Method	In-Domain		Zero-Shot Transfer							
	Avg		Web*		Code		Custom		Macro-Avg	
	N@10	C@10	N@10	C@10	N@10	C@10	N@10	C@10	N@10	C@10
BM25	39.2	40.9	19.8	12.4	20.7	26.6	25.5	23.7	22.0	20.9
Q2E/ZS	42.0	44.6	20.9	13.9	23.6	31.4	30.2	30.9	24.9	25.4
Q2P/SFT	41.8	44.7	19.7	13.1	22.5	30.4	30.5	29.0	24.2	24.2
D2Q	40.9	42.2	20.0	12.7	23.7	29.6	20.5	18.1	21.4	20.1
Re-Invoke	44.1	47.1	21.0	13.9	23.7	30.5	27.5	26.3	24.1	23.6
TOOLQP-FORMAT	**53.2**	**55.3**	17.8	13.1	22.7	32.2	32.0	31.4	24.2	25.5
TOOLQP	47.5	52.0	**22.3**	**16.1**	**25.3**	**34.2**	**41.0**	**38.6**	**29.5**	**29.6**
`bge-large`	42.5	45.5	19.7	13.5	20.6	26.0	24.0	23.4	21.4	21.0
Q2E/ZS	47.0	50.5	21.2	14.0	24.1	31.1	28.5	27.8	24.6	24.3
Q2P/SFT	45.9	49.1	17.6	13.0	22.5	29.0	34.7	35.9	24.9	25.9
D2Q	47.1	49.5	22.5	16.7	23.1	28.8	27.5	25.8	24.3	23.8
Re-Invoke	52.1	56.8	25.4	19.6	25.0	32.5	34.2	33.0	28.2	28.4
TOOLQP-FORMAT	**56.7**	**59.5**	19.6	14.6	23.8	31.8	30.5	30.2	24.6	25.5
TOOLQP	52.4	56.3	**21.9**	**16.3**	**26.6**	**34.1**	**38.0**	**37.2**	**28.9**	**29.2**
`ToolRet-bge-large`	52.8	54.3	25.4	18.8	21.0	26.8	39.5	36.4	28.7	27.3
Q2E/ZS	54.1	56.4	25.6	19.0	24.7	29.9	**40.6**	37.1	**30.3**	28.7
Q2P/SFT	52.8	56.1	18.6	13.9	22.5	29.7	38.6	36.4	26.6	26.6
D2Q	**58.7**	61.6	28.0	18.6	27.6	33.9	38.2	34.8	31.3	29.1
Re-Invoke	58.4	**63.5**	**28.3**	**21.0**	**28.3**	**37.2**	39.1	34.9	31.9	31.0
TOOLQP-FORMAT	57.7	62.1	19.8	16.9	25.4	34.3	35.1	35.2	26.8	28.8
TOOLQP	56.4	60.8	21.8	17.4	28.0	36.0	40.5	**39.1**	30.1	**30.8**
`e5-mistral-7b`	45.9	49.9	22.1	14.9	17.9	24.3	32.0	29.8	24.0	23.0
Q2E/ZS	47.1	51.2	22.1	15.3	21.0	28.3	35.7	35.6	26.3	26.4
Q2P/SFT	50.8	54.5	23.4	17.4	29.3	38.1	41.3	37.5	31.3	31.0
D2Q	52.9	55.9	23.8	16.2	27.0	33.9	29.2	27.0	26.7	25.7
Re-Invoke	54.5	60.1	**26.2**	18.4	28.1	35.9	37.8	33.2	30.7	29.1
TOOLQP-FORMAT	**59.1**	**63.3**	21.5	17.8	31.5	42.3	36.2	35.0	29.7	31.7
TOOLQP	55.6	60.5	24.4	**19.9**	**33.8**	**44.8**	**43.3**	**39.2**	**33.9**	**34.6**
`e5-base`	46.1	48.7	14.7	10.1	14.6	17.9	23.0	22.0	17.4	16.7
Q2E/ZS	47.0	49.8	15.4	10.9	17.3	21.4	24.6	24.7	19.1	19.0
Q2P/SFT	48.2	51.0	14.5	11.3	17.5	22.4	28.8	29.8	20.3	21.2
D2Q	48.4	50.3	19.7	13.8	**22.5**	**28.3**	23.7	23.0	22.0	21.7
Re-Invoke	51.4	56.4	**22.0**	**16.1**	21.4	27.3	**32.5**	30.2	**25.3**	**24.5**
TOOLQP-FORMAT	**60.1**	**60.6**	13.9	12.7	18.0	23.2	24.0	25.9	18.6	20.6
TOOLQP	52.2	55.1	17.0	13.2	21.5	27.5	31.9	**32.4**	23.5	24.4
`ToolRet-e5-base`	60.4	63.5	24.5	17.6	18.3	22.9	38.2	37.0	27.0	25.8
Q2E/ZS	60.5	63.6	25.3	19.3	22.5	30.1	39.4	**38.4**	29.1	29.3
Q2P/SFT	57.2	60.7	22.1	16.4	20.2	27.9	37.0	**38.4**	26.4	27.6
D2Q	57.6	61.2	25.2	17.8	**26.1**	32.5	33.4	32.3	28.2	27.5
Re-Invoke	57.6	62.7	**25.6**	18.5	25.1	32.0	35.1	33.0	28.6	27.8
TOOLQP-FORMAT	**61.3**	**64.4**	21.2	17.2	23.9	31.9	34.6	34.8	26.5	28.0
TOOLQP	59.3	63.1	23.0	**19.5**	25.2	**33.4**	**39.9**	38.0	**29.4**	**30.3**

Table 9: Retriever transfer results on TOOLRET. TOOLQP is trained on `gte-qwen`-generated data, and used out-of-the-box directly with various retrievers at inference time. Web* denotes the zero-shot Web datasets (excluding the in-domain training sources).

Role	Message
System	You are a Multi-Turn Tool Retrieval Planner, an expert AI assistant that deconstructs a user's query into a logical plan and then executes that plan to find the best tools. You must perform one of the following: - To create the initial plan, you must output a <task_breakdown> block followed by a <sub_goals> block on the next line. - To execute your plan, you must output a single functional search query in a <query>block. - To end the retrieval process, you must output the <stop_retrieval> tag when you have found tools for all sub-goals.
User	User query: I'm planning a beach vacation with my friends and I need information about hotels in Cancun. Can you suggest some hotels in Cancun with a beachfront location, all-inclusive meal plan, and water sports facilities? Also, provide me with the hotel details and guest reviews. Additionally, I would like to know the cancellation probability of a flight from Los Angeles to Cancun on June 30, 2024, operated by American Airlines, flight number AA567.
Assistant	<task_breakdown> Given a 'travel accommodation and flight information' task, retrieve tools that can suggest hotels based on location, amenities, meal plans, and guest reviews, as well as tools that provide the cancellation probability for a specific flight by processing parameters such as region, check-in/out dates, number of adults, desired amenities, flight details, and dates. The tools should return hotel suggestions with detailed information and reviews, as well as the cancellation probability for the specified flight. </task_breakdown> <sub_goals> ["Suggest hotels based on location, amenities, meal plans, and guest reviews.", "Retrieve hotel suggestions with detailed information and guest reviews based on location, amenities, meal plans, and check-in/out dates.", "Provide the cancellation probability for a specific flight based on flight details and dates."] </sub_goals>
User	Begin retrieval.
Assistant	hotels by filter domain string region_id int check_in date check_out date adults int children int amenities array meal_plan beachfront water_sports all-inclusive accessibility boolean available only
User	System retrieved tools for previous query: {formatted_retrieval_results}
Assistant	hotel suggestion tool with detailed hotel information and guest reviews filters region date_range amenities beachfront all-inclusive meal_plan water_sports
User	System retrieved tools for previous query: {formatted_retrieval_results}
Assistant	flight cancellation probability flight_number string airline string origin string destination string flight_date date
User	System retrieved tools for previous query: {formatted_retrieval_results}
Assistant	<stop_retrieval>

Figure 7: Training (SFT) example for TOOLQP.

You are an expert AI assistant. Your task is to search for a specific set of 'Tools' that can be used to successfully solve the user's query. Given a user query, you should rewrite it into a search query for a dense retrieval system that covers all possible tools that can be used to solve this query. Your output should contain the rewritten search query only without any intermediate output.

User Query: {query}

Figure 8: Prompt for the Q2E/ZS baseline.

You are an expert AI assistant. Your task is to search for a specific set of 'Tools' that can be used to successfully solve the user's query. Given a user query and possibly relevant context, you should rewrite it into a search query for a dense retrieval system that covers all possible tools that can be used to solve this query. Your output should contain the rewritten search query only without any intermediate output.

Context:
{prf_context}
User Query: {query}

Figure 9: Prompt for the Q2E/PRF baseline.

You are an expert AI assistant. Your task is to create a 'Tool' that can be used to successfully solve the user's query. Given a user query, you should consider all tools that are needed to solve this query, and output one of them. Your output should contain a single tool in JSON format. An example tool is in this format: `{"name": "tool name", "description": "tool description.", "parameters": {"parameter a": {"description": "parameter a description.", "type": "parameter a type"}}}`

User Query: `{query}`

Figure 10: Prompt for the Q2D/ZS and HyDE/ZS baselines.

You are an expert AI assistant. Your task is to create a 'Tool' that can be used to successfully solve the user's query. Given a user query and possibly relevant context, you should consider a ll tools that are needed to solve this query, and output one of them. Your output should contain a single tool in JSON format.

Context:
`{prf_context}`
User Query: `{query}`

Figure 11: Prompt for the Q2D/PRF and HyDE/PRF baselines.

Suppose you are an assistant and you have access to the following API to answer user's queries. You are provided with a tool and its available API function including the description and parameters.

Your task is to generate a possible user query that can be handled by the API.

You must include the input parameters required in the API call. Please be creative and generate random but specific information for the required parameters. Now you are given the API documentation below:

`{tool_documentation}`

Please generate a user query that you will need to call this tool. Note the generated query should be complex enough to describe the scenarios that you will need to call the provided API to address them.

The relevant query is:

Figure 12: Prompt for the D2Q/ZS and Re-Invoke baselines.

Instructions
Suppose you are a query analyzer and your task is to extract the underlying user intents from the input query. You should preserve all the underlying user request and the extracted user intents should be easily understood without extra context information. You should carefully read the given user query to understand its different intents. Then identify what are the specific intents. Each individual intent should be separated by a newline.

Here are some examples of how you should solve the task.

Example
Query: I'm planning to travel to Paris next weekend to visit my family, could you help me book a round trip flight ticket? I want to fly in economy class.
Intent:
book a round-trip flight ticket in economy class to Paris next weekend

Query: I'm a potential buyer looking for a condominium in the city of Miami. I am specifically interested in properties that have a minimum of two bathrooms. It should have walkable distance to the grocery stores.
Intent:
buy a real estate in Miami with a minimum of two bathrooms and walkable distance to the grocery stores

Query: I want to learn Spanish by talking to the native speakers at any time. Additionally, can you recommend some interesting books, preferably fictions, so that I can learn by reading? Also include the websites that I can buy them.
Intent:
learn Spanish by talking to the native speakers
recommend fictions to learn Spanish by reading
suggest the websites to buy Spanish fictions

Begin!
Query: `{query}`
Intent:

Figure 13: Prompt for the Re-Invoke baseline, for extracting user intents.

1st Turn

You are an expert AI Planner. Your goal is to find the best tools to solve a user's query by interacting with a tool retrieval system. You will be given the user's query.

For the first turn, you should deconstruct the user's query into a logical plan that you can follow to retrieve the correct tools by extracting the underlying user intents from the input query. You should preserve all the underlying user request and the extracted user intents should be easily understood without extra context information. You should carefully read the given user query to understand its different intents. Then identify what the specific intents are.

From the second turn on, you must generate a search query that targets an individual intent from the intents you extracted. Do not use any tags or formatting, any text you output will be used as the search query. A good tool query is a descriptive 'functional query' that captures the tool's purpose, possibly including likely parameter names (e.g., 'city: string'), categories, or library types to aid retrieval, but not specific filled-in parameter values (e.g. 'city: Chicago') that may harm retrieval. To end the retrieval process, you must output the`<stop_retrieval>` tag without any other text when you have found the best tools for all intents.

User Query: `{query}`. Breakdown of user's intents:

2nd Turn

Next search query (or `<stop_retrieval>` if best tools found for all intents):

3rd Turn+ (Retrieval Feedback)

System retrieved tools for previous query:
`{formatted_retrieved_results}`

Figure 14: Prompt for the TOOLQP-PROMPTING ablation study.

Sub-task Extraction Prompt

You are an AI assistant that extracts key tasks. Given a high-level task breakdown and a specific target tool, extract the single, concise semantic goal from the breakdown that corresponds to that tool. Do not invent new goals that do not exist in the breakdown. Your output should be one single sentence or phrase, and not just keywords.
Task breakdown: {natural_language_plan}.
Target tool name: {tool_name}.

Plan Alignment Prompt

You are an expert planner. Your task is to analyze a user query, a task breakdown, and a list of available tools, then determine the correct sequential order to call the tools to solve the user's request.

Instructions:
1. Read the User Query and Task Breakdown to understand the user's multi-step goal.
2. Examine the 'Unordered Tools with Descriptions' to understand what each tool does.
3. Create a logical, step-by-step plan by ordering the provided tool names. The order should reflect the sequence of operations needed to fulfill the user's query from start to finish.
4. Your output MUST be a single, valid JSON list of tool names, and nothing else. The output list length must be equal to the number of tools in the given unordered list, with each tool appearing once only.

[USER INPUT]
User Query: {query}
Task Breakdown: {task_breakdown}
Unordered Tools with Descriptions: {tools}

[YOUR CORRECT OUTPUT]

Query Generation Prompt

You are an expert AI Planner. Your goal is to find the best tools to solve a user's query by interacting with a tool retrieval system. You will be given the user's query and an initial Task Breakdown that serves as your high-level plan. For each step, you must generate a search query in a <query> tag. Do not use any other tags such as <tool_call> or <reasoning> or <think>. A good tool query is a descriptive 'functional query' that captures the tool's purpose, possibly including likely parameter names (e.g., 'city: string'), categories, or library types to aid retrieval, but not specific filled-in parameter values (e.g. 'city: Chicago') that may harm retrieval.

[Overall User Goal]: {query}
[Your Overall Plan]: {natural_language_plan}
[Previously Completed Steps]:
[Step {i}]
Goal: "{goal}"
Successful Query: "{search_query}"
...
—
{tool_context}
—
[Instruction]: Write the functional query within a <query></query> tag (e.g., '<query>your functional query</query>').

AddMoreInfo (1st attempt)

[Goal For Your Current Step]: Find a tool for "{semantic_goal}"
Task to focus on: craft a functional query to retrieve a tool that can address the current step goal. You can include information or keywords from the goal in your functional query

AddMoreInfo (2nd attempt)

We should look for a tool with this description: "{tool_description}"

AddMoreInfo (3rd attempt)

Previously retrieved with query "{search_query}", here's the retrieved tools:
{retrieval_results}

AddMoreInfo (4th attempt)

Previously retrieved with query "{search_query}", tools: {retrieval_results}
Previously retrieved with query "{search_query+natural_language_plan}", tools: {retrieval_results}

AddMoreInfo (5th attempt)

Hint: Full Target Tool Info: {json.dumps(tool_documentation_without_toolname)}.
Include information from 'description' and relevant parameter names in the functional query.

Figure 15: Prompts for data generation for training TOOLQP.