Towards Scalable and Generalizable Agentic Large Language Models

by

Wei Fang

B.S., National Taiwan University (2018)
S.M., Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2026

Authored by: Wei Fang
Department of Electrical Engineering and Computer Science
March 6, 2026

Certified by: James R. Glass
Senior Research Scientist, Thesis Supervisor

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

THESIS COMMITTEE

Thesis Supervisor

James R. Glass
Senior Research Scientist
Computer Science and Artificial Intelligence Laboratory

Thesis Readers

Peter Szolovits
Professor of Computer Science and Engineering
Department of Electrical Engineering and Computer Science

Yoon Kim
Associate Professor
Department of Electrical Engineering and Computer Science

Towards Scalable and Generalizable Agentic Large Language Models

by

Wei Fang

Submitted to the Department of Electrical Engineering and Computer Science
on March 6, 2026 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

Large language models (LLMs) have evolved from simple text generators to dynamic agents capable of reasoning, coding, and solving complex problems by interacting with external environments. While the integration of information retrieval and tool use allows these models to overcome the limitations of their parametric knowledge, current approaches face significant bottlenecks when scaling to the complexity of real-world tasks. Standard retrieval methods frequently fail under domain shifts, while existing tool-use paradigms rely on unreliable tool retrieval or manual curation when scaling and generalizing to massive, dynamic tool repositories. This thesis presents three approaches designed to overcome these challenges, enabling the development of more robust, scalable, and generalizable agentic LLMs.

In the first part of the thesis, we propose a generalized passage re-ranking approach, which bridges the gap between discriminative and generative retrieval approaches. To address the limitations of conventional cross-encoders and generative scorers, we introduce a joint inference method based on maximizing the pointwise mutual information between queries and passages. This objective explicitly penalizes generic and uninformative content that often confuse likelihood-based models, enhancing the retrieval towards passages that are specific to the input query. We demonstrate that this approach improves retrieval accuracy for open-domain question answering, and exhibits robust zero-shot generalization across diverse retrieval tasks without requiring extensive domain-specific fine-tuning.

In the second part of this thesis, we tackle the challenge of scalable tool retrieval within massive tool repositories. As tool repositories expand, standard retrieval methods often struggle to bridge the semantic gap between user intent and technical tool documentations, and fail to capture the compositional nature of complex user intents. We introduce a tool query planning framework that reformulates tool retrieval as a sequential decision-making process rather than a static single-shot matching task. By utilizing a lightweight query planner to decompose abstract user intents into logical sub-tasks and leveraging interactive feedback from the retrieval environment, this framework effectively addresses those limitations. We show that this planning-based approach enhances tool retrieval performance significantly and robustly, leading to substantial gains in downstream LLM tool use success rates.

In the final part of this thesis, we present an automated framework for zero-shot prompt optimization that enables LLMs to master new tools without human-labeled data. We propose a "tool-play" strategy where an LLM systematically interacts with a tool through trial-and-error, generating and refining tool usage examples and tool documentations based on

execution feedback and self-reflection, effectively optimizing the tool context for downstream LLMs in a zero-shot manner, circumventing the need for the manual prompt engineering and dataset-building efforts that limit scalability. We demonstrate that this approach yields significant and consistent improvements in tool-use accuracy across various LLMs, providing a robust solution for scaling up tool repositories.

Thesis supervisor: James R. Glass
Title: Senior Research Scientist

Acknowledgments

This thesis has benefited tremendously from the guidance and support of many people.

First and foremost, I am deeply grateful to my doctoral advisor, Jim, for his mentorship and support throughout my time at MIT. He gave me so much flexibility to explore whatever topics I am interested in, while offering constructive advice and perspectives on the topics I chose. He also has great communication skills, and I have learned a lot from him. Outside of research, Jim fosters a lively and collaborative environment in which everyone in our group finds it easy to work with one another, and he takes care of all of us really well. I am extremely lucky to have Jim as my advisor.

I am also very thankful to my thesis committee members, Peter Szolovits and Yoon Kim, who have been very responsive and easy to work with during our committee meetings. Their research insights have been extremely helpful in improving my thesis and research directions. I learned a lot from them in those meetings.

Next, I would like to thank my undergraduate research advisors, Lin-Shan Lee, Hung-Yi Lee, and Yu-Chiang Frank Wang, who sparked my interests in this field. They introduced me to the field and inspired me to pursue a Ph.D. degree, and most importantly, through their guidance I became much more certain of and confident in my long-term career goals. They have continued to support me, and I always feel rejuvenated when I meet them during travel or when I am back in Taiwan.

I would like to thank all the researchers at and outside of MIT that I have had the opportunity to collaborate with over the years. In particular, I would like to thank Hongyin, Yung-Sung, and Mitra for our collaborations that are related to this thesis and my masters work. I also want to thank my internship mentors at MIT-IBM Watson AI Lab, Yada Zhu and Yang Zhang. I really enjoyed the research discussions, and our collaboration resulted in a chapter about agentic LLM tool use in this thesis.

I want to express my gratitude to all the members of the SLS group, whom I've spent the most time with over the years. I am lucky to be a part of this group, and have become friends with these fantastic researchers. I want to especially thank my officemates Alex, Andrew, Yung-Sung, Suwon, Di-Chia, Yonatan, and Kai-Wei, and the NLP subgroup members Hongyin, Philip, and Assaf. We have shared so many research ideas, and more importantly, a lot of moments and memories (and lunches and coffees). Also, I want to thank Marcia for her administrative support of our group and the fun chats we had with her.

I would like to thank my awesome friends during my time here, who have made my everyday life much more colorful and the journey a lot easier. In particular, to Yen-Chen and James, who I spent a lot of time with at Sid-Pac and in Taiwan over the Covid years; and my roommate Brabeeba, who I shared so many unique experiences with (including getting

stranded on Mt. Washington overnight). I also want to express thankfulness to all my friends who I share many memorable moments with: En-Chi, Helen, Clement, Chu-Ya, Chung-Hsun, Tracy, Leyton, Anita, Jeff, Shum, Peixin, Kaixiang, Peter, Scarlett, David, Vivian, Xiaoyang, Jin-Cheng, Wendy, Wei-Chiu, my NTUEE '17 cohorts, and many others. Thank you all for adding so many colors to my Ph.D. life.

Finally, I would like to give special thanks to my beloved family, Li-Hsuan, Chen-Ray, and Chiao and my dear partner, Sophie, who have supported me unconditionally throughout this journey. I'm extremely grateful for their unwavering support, and I dedicate this thesis to them.

This research was sponsored by the Centre for Perceptual and Interactive Intelligence (CPII) Ltd. under the Innovation and Technology Commission's InnoHK Scheme. I would like to thank them for their generous support.

Bibliographic Note

Portions of this thesis are based on peer-reviewed publications. Chapter 3 is based on research published in Fang et al. (2024). Chapter 4 is under review with pre-print available (Fang and Glass, 2026). Chapter 5 follows research published in Fang et al. (2025).

Contents

List of Figures 15
List of Tables 17

1 Introduction **19**
1.1 Motivation . . . 19
1.2 Contributions . . . 23
1.3 Overview . . . 23

2 Background **25**
2.1 Information Retrieval . . . 25
2.1.1 Sparse and Dense Retrieval . . . 26
2.1.2 Query and Document Expansion . . . 28
2.1.3 Re-ranking . . . 29
2.1.4 Retrieval Evaluation . . . 31
2.1.5 Retrieval-augmented Generation . . . 32
2.2 Tool-using for Large Language Models . . . 34
2.2.1 Software Tools . . . 36
2.2.2 Tool-augmented Generation . . . 38
2.2.3 Tool Retrieval . . . 40
2.3 Optimization Techniques for Agentic LLMs . . . 42
2.3.1 Supervised Fine-tuning . . . 42
2.3.2 Reinforcement Learning . . . 43
2.3.3 Automatic Prompt Optimization . . . 44

3 Generalizing Passage Re-ranking by Mutual Information Maximization **47**
3.1 Introduction . . . 47
3.2 Methodology: Joint Passage Re-ranking (JPR) . . . 48
3.2.1 Inference by Maximizing Mutual Information . . . 49
3.2.2 Joint Fine-tuning . . . 51
3.3 Related Work . . . 51
3.3.1 Passage Re-ranking . . . 51
3.3.2 Mutual Information Maximization . . . 52
3.4 Experiments: Open-Domain QA Retrieval . . . 52
3.4.1 Data . . . 52
3.4.2 Setup and Baselines . . . 53

3.4.3 Training and Inference Details . . . 53
3.4.4 Results and Discussion . . . 55
3.5 Experiments: Zero-Shot Retrieval . . . 56
3.5.1 Data . . . 56
3.5.2 Setup and Baselines . . . 56
3.5.3 Training and Inference Details . . . 57
3.5.4 Results and Discussion . . . 58
3.5.5 Ablation Studies and Qualitative Analysis . . . 58
3.6 Chapter Summary . . . 61

4 Retrieving Tools for LLM Tool-use via Query Planning 65
4.1 Introduction . . . 65
4.2 Methodology: Multi-step Tool Retrieval with Query Planning (ToolQP) . . . 67
4.2.1 The Modeling Framework . . . 67
4.2.2 Data Generation & SFT . . . 69
4.2.3 Training – RLVR . . . 71
4.3 Related Work . . . 73
4.3.1 Tool Learning and Retrieval . . . 73
4.3.2 Generative Modeling for Retrieval . . . 74
4.4 Experiments . . . 74
4.4.1 Tool Retrieval . . . 74
4.4.2 Transfer to Unseen Base Retrievers . . . 78
4.4.3 End-to-end Tool Calling . . . 80
4.4.4 Ablation Studies . . . 82
4.4.5 Latency Analysis . . . 84
4.4.6 Qualitative Analysis . . . 84
4.5 Chapter Summary . . . 89

5 Prompt Optimization for LLM Tool Use 91
5.1 Introduction . . . 91
5.2 Methodology: Zero-shot Prompt Optimization via Tool Play (Play2Prompt) 93
5.2.1 Problem Formulation and Notation. . . . 93
5.2.2 Play2Prompt Overview . . . 94
5.2.3 Search Framework . . . 95
5.2.4 Tool-Use Example Generation . . . 96
5.2.5 Tool Documentation Optimization . . . 97
5.3 Related Work . . . 98
5.3.1 LLMs for Tool Use . . . 98
5.3.2 Tool-Use Instructions and Optimization . . . 99
5.4 Experiments . . . 100
5.4.1 Data and Setup . . . 100
5.4.2 Results on BFCL and StableToolbench . . . 102
5.4.3 Robustness of Play2Prompt . . . 104
5.4.4 Ablation Studies . . . 105
5.4.5 Qualitative Analysis . . . 107

5.4.6 Human Evaluation of Generation Quality . . . 111
5.5 Chapter Summary . . . 113

6 Conclusion and Future Work **115**
6.1 Conclusions . . . 115
6.2 Future Work . . . 117

A Additional Details for Chapter 3 (JPR) Experiments **121**
A.1 Dataset Details and Licenses . . . 121

B Additional Details for Chapter 4 (TOOLQP) Experiments **123**
B.1 Dataset Details and Licenses . . . 123
B.2 Additional Baseline Configurations on ToolRet . . . 123
B.3 Detailed Results for the Retrieval Transfer Experiments . . . 126
B.4 Prompts for Baseline Methods . . . 126

C Additional Details for Chapter 5 (PLAY2PROMPT) Experiments **135**
C.1 Dataset Licenses . . . 135
C.2 Detailed Results for the Robustness Experiments . . . 135
C.3 Pseudo-code and Meta-prompts for PLAY2PROMPT . . . 136

List of Figures

1.1 Context construction in the agentic LLM framework. 20

2.1 Example of software tools with different documentation schema and completeness. 37
2.2 Example of LLM performing tool-augmented generation. 39
2.3 Example system prompt that includes tool definitions and in-context tool-use examples. 41

3.1 Example showing a passage that is estimated to have high retrieval probabilities for multiple queries by a conventional re-ranker. Each query asks about different specifics of a movie; however the passage contains mostly general information, and could not be used to answer several top-ranked questions. This motivates our use of a penalization term to discount these high probability passages that are not specific to the input query. 49
3.2 Example of JPR taken from NQ's test set. PMI denotes $\log p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) - \lambda \log p_{\boldsymbol{\phi},\boldsymbol{\theta}}(z)$. 61
3.3 Example of JPR taken from NQ's test set. PMI denotes $\log p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) - \lambda \log p_{\boldsymbol{\phi},\boldsymbol{\theta}}(z)$. 62
3.4 Example of JPR taken from NQ's test set (continued from Figure 3.3). PMI denotes $\log p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) - \lambda \log p_{\boldsymbol{\phi},\boldsymbol{\theta}}(z)$. 63

4.1 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. 68
4.2 Example trajectory constructed via Algorithm 1 for the SFT of TOOLQP. . 72
4.3 Retrieval comparison for example taken from ToolRet's test set, showing that TOOLQP discovers dependencies. 84
4.4 Retrieval comparison for example taken from ToolRet's test set, showing TOOLQP finding two correct tools for a single sub-task. 86
4.5 Retrieval comparison for example taken from ToolRet's test set, showing TOOLQP finding a tool that other methods fail to rank high. 87
4.6 Retrieval comparison for example taken from ToolRet's test set, showing the benefit of planning sub-tasks with TOOLQP. 88
4.7 Retrieval comparison for example taken from ToolRet's test set, which shows fine-grained and targeted queries for each sub-task help retrieve the target tools accurately. 89

5.1 The PLAY2PROMPT framework: Beam search iteratively searches tool-use examples, incorporating tool play into the proposal process. After examples are generated, beam search is once again applied to optimize documentation by incorporating tool-use outputs and errors. Finally, optimized tool-use examples and documentation are used as prompts for $\mathcal{M}_T$ at inference. 94
5.2 Average (weighted) accuracy improvements with PLAY2PROMPT on BFCL, for different parameter description dropout p. 105
5.3 A tool-use example generation search trajectory that shows PLAY2PROMPT in action, with the more difficult and informative example pair being ranked highest. This example is chosen from BFCL's test set. 108
5.4 Another example of PLAY2PROMPT's tool-use example generation with a complex RESTful weather tool, taken from BFCL's test set. 109
5.5 An example of PLAY2PROMPT facing incorrect documentation. We show the beam search trajectory with the highest solve rate on the validation set. At each step, a new documentation is explored based on error feedback. Example chosen from StableToolBench. 110
5.6 A typical example of PLAY2PROMPT assisting LLMs in correcting errors in generating parameter values with optimized documentation. Also from StableToolBench. 111

A.1 Dataset examples from Natural Questions (NQ) and TriviaQA. 122

B.1 Dataset examples from each of the three ToolRet categories. 124
B.2 Dataset examples from API-Bank and StableToolBench. 127
B.3 Prompt for the Q2E/ZS baseline. 129
B.4 Prompt for the Q2E/PRF baseline. 129
B.5 Prompt for the Q2D/ZS and HyDE/ZS baselines. 129
B.6 Prompt for the Q2D/PRF and HyDE/PRF baselines. 129
B.7 Prompt for the D2Q/ZS and Re-Invoke baselines. 130
B.8 Prompt for the Re-Invoke baseline, for extracting user intents. 131
B.9 Prompt for the TOOLQP-PROMPTING ablation study. 132
B.10 Prompts for data generation for TOOLQP training. 133
B.11 Prompts for data generation for TOOLQP training (continued). 134

C.1 Dataset examples from BFCL. 138
C.2 Meta-prompt m_1 for PLAY2PROMPT 139
C.3 Meta-prompt m_2 for PLAY2PROMPT 140
C.4 Meta-prompt m_3 for PLAY2PROMPT 141
C.5 Meta-prompt m_4 for PLAY2PROMPT 142
C.6 Meta-prompt m_5 for PLAY2PROMPT 142
C.7 Meta-prompt m_6 for PLAY2PROMPT 143
C.8 Meta-prompt m_7 for PLAY2PROMPT 143
C.9 Meta-prompt m_8 for PLAY2PROMPT 144

List of Tables

3.1 Training hyperparameters for NQ and TriviaQA selected by performance on the dev set. . . . 54
3.2 Top-K retrieval accuracy (%) on the Natural Questions and TriviaQA test sets. All non-BM25 methods re-rank the top-100 passages retrieved by BM25. Best overall are in **bold** while best non-LLM are underlined. . . . 55
3.3 Zero-shot results on BEIR, scores denote **nDCG@10**. All methods re-rank the top-100 passages retrieved by BM25, except for TREC-DL 2019 to compare to prior work. Best overall are in **bold**. Underlined indicate in-domain performance, and *italicized* are based on Pyserini reproductions, differing from those reported in prior work. . . . 57
3.4 Top-K retrieval accuracy (%) on NQ for different model combinations with the proposed JPR. . . . 59
3.5 Top-K retrieval accuracy (%) on NQ for JPR compared to ensembling and fusion methods. . . . 60

4.1 Details of baselines used in TOOLQP experiments. . . . 76
4.2 Results on TOOLRET, a benchmark of 35 datasets. Web* denotes the zero-shot Web datasets. nDCG@10 is denoted as N@10, and Completeness@10 is denoted as C@10. . . . 78
4.3 Base retriever details for the retriever transfer setting. . . . 79
4.4 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. 80
4.5 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. . . . 81
4.6 Ablation studies for TOOLQP. . . . 82
4.7 Latency Analysis on ToolRet Zero-shot categories. Setting: 1 (or 4 for 30B) A6000 GPU, single query input; Q2E sampling is parallel, cross-enc (`bge-gemma`) uses a batch size of 4. . . . 84

5.1 Results on BFCL Executable. Accuracy scores are shown. . . . 103
5.2 Results on StableToolBench. Solvable pass rates are shown. . . . 103

5.3 Ablation on PLAY2PROMPT (P2P) using generated example demonstrations only (P2P-Demo) and generated descriptions only (P2P-Desc). Scores indicate average accuracy for BFCL and average solvable pass rate for StableToolBench (STB). 106
5.4 Ablation on demonstration difficulty for PLAY2PROMPT. We report average solvable pass rates. 106
5.5 Ablation on search strategies, on the I3-Inst subset of StableToolBench. LLaMA-70B is used for task model $\mathcal{M}_T$. MC denotes Monte Carlo search. . 107
5.6 Human evaluation on tool documentation quality, comparing completeness, conciseness, and accuracy of PLAY2PROMPT versus raw (initial) documentation. 112
5.7 Human evaluation on tool usage example (question-function call) quality, rated on a scale of 1 (low) to 5 (high). 112

A.1 Dataset splits for NQ and TriviaQA. 121

B.1 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. . 125
B.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. Web* denotes the zero-shot Web datasets (excluding the in-domain training sources). 128

C.1 Results on BFCL Executable, with parameter description dropout $p = 0.5$. Accuracy scores are shown. 136
C.2 Results on BFCL Executable, with parameter description dropout $p = 1.0$. Accuracy scores are shown. 137

Chapter 1

Introduction

1.1 Motivation

Large language models (LLMs) have rapidly become a general-purpose interface for interacting with information. A single model used purely as a text generator can draft text, write code, summarize documents, answer questions, and follow natural-language instructions across many domains. At the same time, many real-world tasks require more than generating fluent text—they require grounding in external knowledge and acting through external systems. Questions depend on fast-changing information; answers may need to cite evidence; and tasks often require executing precise operations such as querying databases, calling functions, or running code. These requirements have motivated a shift from treating LLMs as standalone text generators to components in *agentic* systems that retrieve information, select tools, interact with environments, and iteratively refine solutions from feedback.

An agentic LLM system embeds an LLM in a loop, shown in Figure 1.1, that alternates between (1) producing an action from the LLM (e.g. retrieving information or call a function), and (2) observing the outcome of that action (e.g. retrieved documents or function outputs), which then informs subsequent actions. It can be viewed as a system that repeatedly answers the question: *what should be added to the context next to solve this task and*

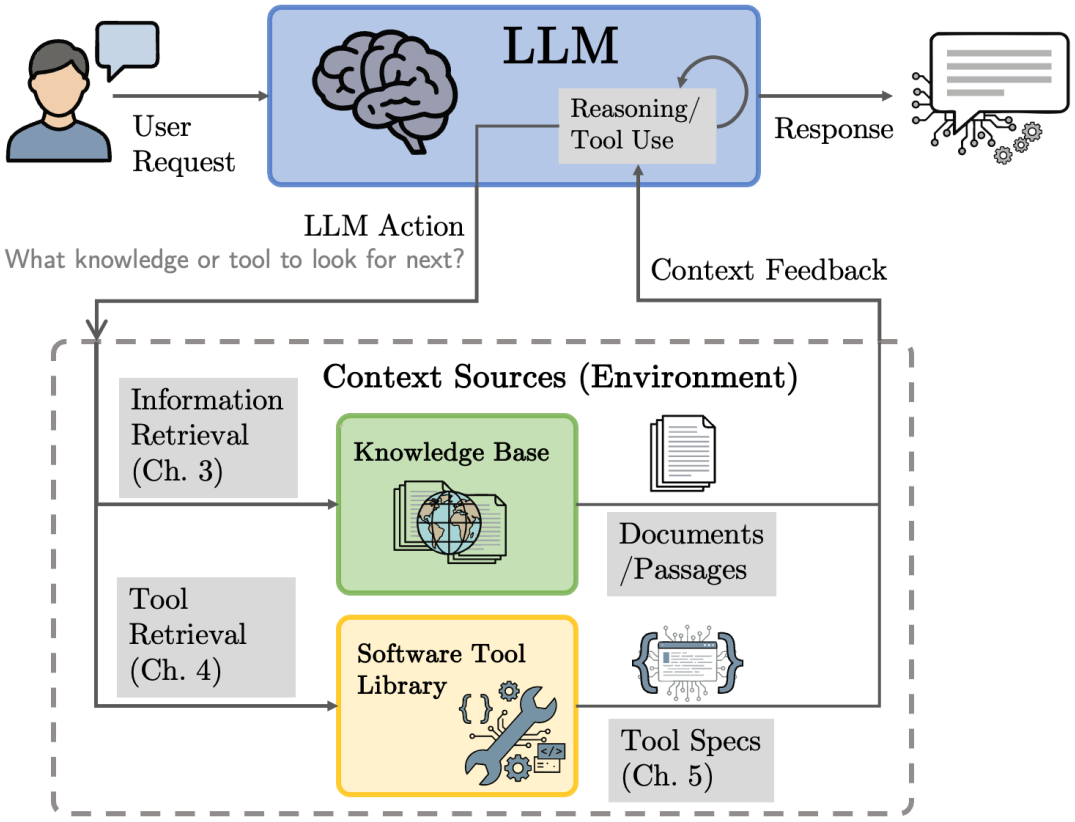

Figure 1.1: Context construction in the agentic LLM framework.

produce a satisfactory response? The answer may be evidence passages from a corpus, software tools from a library, or outputs from running code or tools. This perspective highlights two requirements for deploying agentic systems to real-world settings: *scalability* and *generalizability*. Scalability concerns how the agentic system handles large external resources, such as retrieval over millions of passages or selection from thousands of APIs. Generalizability concerns how the system handles increased task complexity, such as complex multi-step or compositional problems, and how it functions under domain shifts, new tools, noisy and incomplete context, or changes to the environment.

Despite substantial progress, current agentic LLM pipelines still face fundamental bottlenecks that stem from how they represent and retrieve context, and from how they specify tool behavior. In this thesis, we focus on three such problems.

First, retrieval systems should identify context that is not merely correlated with a query, but *specific* to it. Modern retrieval-augmented pipelines typically use a fast first-stage retriever to obtain candidates, followed by a more computationally-expensive re-ranking model to refine the ranking. While this design is effective in many settings, it can fail in a characteristic way when scaled up: some passages are broadly relevant to many queries, for example introductory or summary passages, and therefore receive high scores across different inputs. These *high-prior* passages can dominate rankings even when they are not the passages that contain the answer-specific details that a downstream LLM requires. This issue becomes more pronounced with out-of-domain scenarios. It suggests that ranking solely by conditional relevance could be insufficient, and the retrieval objective should also discourage overly generic content and favor those that are informative given the query.

Next, context-construction problems extend beyond text passages to the tools that LLM agents use. As agentic systems evolve, tool libraries grow from small sets of handpicked functions to large, diverse toolsets. In such scenarios, it is infeasible to place all tools and their documentation in a prompt, making tool retrieval a necessary first step. However, retrieving tools is considerably different from standard text retrieval. Tool documentations are often technical, parameter-driven, and have inconsistent styles across sources, whereas user requests are high-level and goal-oriented, creating a semantic gap. Furthermore, real-world tasks are often compositional, as a single request can require multiple tools, sometimes with dependencies that only become evident after viewing the retrieved tool documentation. Single-shot retrieval struggles in these scenarios due to the semantic mismatches, limitations of embedding models representing compositional concepts, and because they cannot interactively revise its search strategy.

Moreover, even when the correct tools are retrieved, agentic reliability depends largely on how tool behavior is specified in the prompt. In inference-time tool-use settings, the LLM agent is typically given the tool documentation and often a small set of in-context tool-use scenarios for each tool. This approach is brittle in real-world tool libraries, as documentation

can be incomplete, ambiguous, inconsistent across sources, or simply outdated, and newly introduced tools often come with no curated usage demonstrations. Consequently, failures often arise from mismatches between the agent's inferred call specification and the tool's actual functional semantics, leading to tool calls that violate the declared input schema or misinterpretation of its functionality. Humans can often resolve such ambiguity by iteratively probing an unfamiliar tool, and using execution errors and observed outputs as feedback, whereas standard systems lack a principled procedure to interactively identify missing interface semantics.

Together these effects undermine both scalability and generalizability. We tackle these limitations in this thesis in Chapters 3 through 5, each corresponding to a different stage in the agentic pipeline depicted in Figure 1.1.

We first focus on passage re-ranking for retrieval-augmented LLMs. We develop a perspective based on mutual information between queries and passages. The core idea is to rank passages not only by how compatible they are with the query, but also by how *informative* they are relative to what would be retrieved frequently regardless of the query. This introduces a re-ranking mechanism that enables better retrieval for open-domain QA and zero-shot retrieval.

Next, we study the retrieval of tools from large tool libraries for agentic LLMs. We formulate tool retrieval as an interactive multi-step process driven by *query planning*: the system decomposes a user request into sub-tasks and issues targeted retrieval queries for each sub-task, improving tool retrieval correctness and enhances tool-use capabilities for the downstream LLM.

Finally, we investigate prompt construction for agentic tool-use, particularly when new tools arrive with poor documentation and no example demonstrations. We introduce a zero-shot prompt optimization framework that utilizes *tool play* to discover valid invocations and observe tool behavior, resulting in agentic LLMs calling the tool more reliably.

1.2 Contributions

In this thesis, we explore three crucial problems of context construction for scaling and generalizing agentic LLMs.

First we introduce a mutual-information perspective for passage re-ranking that emphasizes query-specific evidence. We develop an inference objective that discourages generic, high-prior passages and favors passages that provide information specific to the input query, addressing a common failure mode of large-scale retrieval. We show how a conventional cross-encoding re-ranker and a conditional generation model can be combined through a conceptually grounded scoring rule.

Next, we propose a query-planning formulation and scalable learning approach of interactive tool retrieval for scaling LLM tool use. We frame tool retrieval as an interactive process that decomposes complex requests into sub-tasks and retrieving tools with targeted sub-queries, directly targeting semantic misalignment and compositionality, two core challenges of reliable tool use at scale. We propose a learning framework in which planning policies can be learned from retrieval feedback and can be used modularly with different underlying retrieval models, enabling robust tool retrieval that is less brittle to tool library or retriever changes.

Finally, we design a zero-shot prompt optimization framework for new tools based on tool play. We introduce a method that does not assume curated tool-use examples, instead interacting with tools to discover valid usage patterns, synthesizes tool-use demonstrations, and refines tool documentation based on observed errors and outputs. This approach makes prompt construction for tool use scalable by eliminating the need for human-curated examples and documentations, and robust by grounding the updates in observed tool behavior.

1.3 Overview

The remainder of this thesis is organized as follows:

- Chapter 2 reviews literature on information retrieval, tool use, and optimization

techniques commonly used in agentic LLM systems.

- Chapter 3 presents Joint Passage Re-ranking (JPR), which improves retrieval robustness by combining discriminative re-ranking and generative modeling through a mutual-information-based inference objective.

- Chapter 4 introduces TOOLQP, a query planning framework for retrieving tools from large libraries by decomposing tasks, iteratively querying retrievers with feedback, and aggregating tools across sub-tasks.

- Chapter 5 presents PLAY2PROMPT, a zero-shot prompt optimization framework that uses tool play to generate tool-use demonstrations and refine tool documentation when curated documentation and examples are unavailable.

- Chapter 6 summarizes the thesis and discusses directions for future work.

Chapter 2

Background

In this chapter, we provide background for several topics that are related to this thesis, including information retrieval, tool-using for LLMs, and LLM-related optimization methods.

2.1 Information Retrieval

Information retrieval (IR) is broadly defined as the task of finding material, usually of an unstructured or semi-structured nature, that satisfies a user's information need, often as a natural language query, from within large collections. It encompasses the identification of information at varying levels of granularity and across different modalities. In the context of natural language processing (NLP) tasks, the unit of retrieval is not necessarily a complete textual document. Instead, systems often operate on specific passages, coherent semantic chunks, or structured object definitions. This distinction is central to the work presented in this thesis: Chapter 3 focuses on the retrieval of textual passages to answer open-domain questions, while Chapter 4 treats software tools, structured as functional definitions, as the unit of retrieval.

The integration of retrieval mechanisms enables the transition from purely generative models to robust agentic systems. While LLMs demonstrate impressive capabilities, they remain closed systems limited by their training data. Retrieval bridges this gap by grounding

the model's generation in external, verifiable evidence, thereby improving factual reliability and enabling access to dynamic information without frequent re-training. This capability shifts retrieval from a supporting utility to an integral system component, supporting agentic behavior through access to accurate, up-to-date, and auditable information. The following subsections introduce key approaches and their roles in end-to-end retrieval pipelines, ranging from sparse and dense retrieval and query/document expansion to re-ranking, retrieval-augmented generation, and evaluation.

2.1.1 Sparse and Dense Retrieval

Sparse Retrieval. The fundamental challenge in retrieval is matching a user's query x to a relevant document z from a large corpus $\mathcal{Z}$. The earliest approach is sparse retrieval, often referred to as *lexical matching*. These methods characterize texts by the discrete words they contain, creating sparse vector representations in a *bag-of-words* fashion of dimensionality V, the vocabulary size. This discards grammar and order, and focuses solely on word occurrence. To quantify relevance, early systems employed *term frequency–inverse document frequency* (TF-IDF) weighting to penalize common words and highlight discriminative terms (Sparck Jones, 1972; Salton et al., 1975). Building upon these principles, BM25 (Robertson et al., 2009) established itself as the robust standard for sparse retrieval, by refining TF-IDF with a parameter to limit the influence of repetitive terms, and another to normalize against document length. To this day, BM25 remains a viable method in modern systems. Its reliance on exact string matching makes it exceptionally precise for queries containing rare entities or specific technical specifiers that learned models might fail to capture. Furthermore, its compatibility with inverted indices allows for extremely fast retrieval over large-scale corpora with minimal computational overhead.

However, sparse retrievers are fundamentally susceptible to *vocabulary mismatch*, where queries and documents use different terms to describe the same concept. Early research explored latent semantic analysis (Deerwester et al., 1990), using singular value decomposition

to the term-document matrix to identify a lower-dimensional subspace that captures semantic co-occurrence, thereby allowing for matching based on latent topics rather than exact keywords. Nonetheless, it is still limited by its linear nature and its inability to capture the syntactic structure or order of words.

Dense Retrieval. Modern dense retrieval methods overcome these limitations by employing deep neural networks to encode text into continuous, low-dimensional vector embeddings. Unlike sparse retrieval methods, which operate on bag-of-words statistics, neural models encode entire sentences or passages, allowing the vector to capture complex syntactic relationships and contextual meaning. Contemporary systems typically utilize a *Bi-Encoder* architecture, such as dense passage retrieval (DPR) (Karpukhin et al., 2020). Two independent neural networks, E_X and E_Z, map the query and document into vectors. Relevance is defined as their dot product similarity $sim(x, z) = E_X(x)^\top E_Z(z)$. Training commonly involves a contrastive loss function, maximizing the similarity between positive pairs while minimizing it against negatives:

$$\mathcal{L}(x, z^+, z^-_{1..k}) = -\log \frac{e^{\text{sim}(x,z^+)}}{e^{\text{sim}(x,z^+)} + \sum_{j=1}^{k} e^{\text{sim}(x,z^-_j)}}$$

While early models like DPR excelled at in-domain tasks, they often suffered from domain shift, underperforming BM25 in zero-shot scenarios. However, recent advancements in unsupervised pretraining and large-scale instruction tuning have closed this gap, demonstrating robust generalization capabilities (Li et al., 2023c; Lee et al., 2025). Ultimately, the choice between sparse and dense retrieval represents a tradeoff between exact lexical matching and semantic understanding, leading many state-of-the-art systems to employ hybrid approaches that combine both paradigms.

2.1.2 Query and Document Expansion

To address the mismatch between query and documents, early work developed expansion techniques that augment the text representation—either the query or the document—with additional, semantically related terms. While initially developed for sparse retrieval to increase term overlap, these expansion methods remain highly effective for modern dense retrievers by enriching the semantic signal available to the neural encoders.

Query Expansion. The concept of refining a query based on relevance feedback was first introduced by Maron and Kuhns (1960), who proposed using probabilistic indexing to adjust the weight of query terms based on user interaction. Rocchio (1971) further proposed pseudo-relevance feedback (PRF), assuming the top k retrieved documents are *pseudo-relevant* and extracts significant terms from them to refine the query. The application of these expansion terms is typically through modification of the query vector in the vector space towards the centroid of relevant document vectors, or in the textual space by explicitly concatenating the expansion terms to the original query text, thereby increasing the likelihood of matching relevant documents in the corpus.

More recently, the rise of LLMs have enabled generative query expansion, which instead of relying on potentially noisy relevance feedback from a first-pass retrieval, these methods leverage the parametric knowledge of LLMs to generate expansion terms. For example, in Wang et al. (2023) and Gao et al. (2023a), hypothetical "documents" that answers the query are generated, which serves as a semantically-rich proxy for the user's intent. These hypothetical documents are incorporated into the query for retrieval, effectively bridging the gap by performing a *semantic translation* into the document space.

Document Expansion. While query expansion modifies the input at inference time, document expansion enriches the corpus itself prior to indexing, which allows for more expensive computational processing. Early approaches expanded document representations

with related terms, while modern studies such as Doc2Query (Nogueira et al., 2019) have leveraged the generative capabilities of LLMs to predict potential queries that a document might answer. These queries are joined to the original document in the textual space or in the vector space, effectively indexing the latent intent of the document, for improved semantic matching.

2.1.3 Re-ranking

To balance the inherent trade-off between retrieval speed and accuracy, modern retrieval systems typically utilize a multi-stage *retrieve-and-rerank* pipeline. The first stage, using efficient sparse or dense retrievers, acts as a high-recall filter to identify a candidate set of potentially relevant documents from the massive corpus. In the second stage, more computationally intensive models are used to *re-rank* the documents into more optimal order.

The methodology of re-ranking has evolved significantly from early learning-to-rank approaches to modern neural networks. Historically, re-ranking was dominated by statistical and machine learning methods that combine hand-crafted features using regression or support vector machines. These methods are categorized into three distinct paradigms: pointwise, pairwise, and listwise approaches.

Pointwise ranking is the most direct approach, where the model predicts a relevance score $s(x, z)$ for a single query-document pair independent from other candidates. This paradigm is exemplified by the *cross-encoder* architecture, which, unlike the bi-encoder used in the initial retrieval stage that processes queries and documents separately to allow for pre-computation, takes as input the query and document concatenated as a single sequence. This fundamentally allows the model to capture more semantic dependencies compared to the vector-based dot products. The model is often trained with noise-contrastive objectives or ranking losses. While there are studies that explore architectures within this paradigm that lie in between fully independent bi-encoders and cross-encoders (that use full cross-attention), in this thesis, we follow the mainstream cross-encoder architecture when re-ranking.

On the other hand, *pairwise* ranking shifts from scoring individual documents to predicting the relative order between pairs of documents, and are usually trained with ranking losses. While effective at capturing relative preferences, they suffer from computationally inefficiency due to the need to compare all combinations from the candidate list, and from inconsistency since they do not consider global rankings. Alternatively, *listwise* ranking attempts to optimize multiple documents in ranked list simultaneously. By directly optimizing retrieval metrics, which we will introduce in detail in the next section, listwise approaches theoretically offer the best alignment with the final retrieval objective. However, they are historically difficult to train due to the complexity of the permutation space and, in the context of LLMs, are constrained by the finite context window which limits the number of documents that can be assessed in a single forward pass.

Finally, we introduce recent advances that re-purposes pre-trained generative models for re-ranking tasks. One prominent approach involves training encoder-decoder models to classify relevance by using the `True/False` token probabilities as scores, effectively applying the pointwise cross-encoder paradigm to a generative model. Alternatively, zero-shot methods like unsupervised passage re-ranking (UPR) (Sachan et al., 2022) leverage auto-regressive LMs by scoring a document's relevance with the likelihood of generating a query x given the document z. This is formally defined as the average log-likelihood of the query tokens:

$$\text{Score}(x, z) = \frac{1}{|x|} \sum_{t=1}^{m} \log P(x_t | x_{<t}, z)$$

This approach, utilized and adapted in Chapter 3, leverages the model's internal knowledge to bridge semantic gaps. Finally, recent advancements have enabled listwise generative re-ranking using state-of-the-art LLMs (Sun et al., 2023), which feed a set of candidate documents into the LLM prompt and instruct the model to output the identifiers of the documents in descending order of relevance. To handle lists larger than the context window, these systems employ sliding window strategies, sorting small batches of documents and

iteratively merging the results to produce a global ranking.

2.1.4 Retrieval Evaluation

The evaluation of retrieval systems requires quantifying how effectively a system separates relevant information from noise. The most fundamental metrics for this task are derived from the confusion matrix of binary classification, treating retrieval as a decision problem of whether to include or exclude a document. **Precision** measures the exactness of the system—specifically, the fraction of retrieved documents that are actually relevant ($P = \frac{|Relevant \cap Retrieved|}{|Retrieved|}$). On the other hand, **Recall** measures the completeness of the system—the fraction of all relevant documents in the corpus that were successfully retrieved ($R = \frac{|Relevant \cap Retrieved|}{|Relevant|}$). Because these two metrics often trade off against one another (retrieving the entire corpus guarantees 100% recall but near-zero precision), the F1 Score is frequently employed as their harmonic mean to provide a balanced single-value metric.

While fundamental, these set-based metrics treat the retrieved list as an unordered set, failing to account for the position of relevant items. In practical applications, a system that places the relevant document at rank 1 is far superior to one that places it at rank 100, even if their precision and recall at $K = 100$ are identical. To capture this, the normalized discounted cumulative gain (**nDCG@K**) was proposed (Wang et al., 2013), which operates on two principles: "gain," which credits the system for retrieving relevant documents (with relevance scores rel_i), and "discounting," which penalizes the system logarithmically if those documents appear lower in the list. The discounted cumulative gain at rank K is defined as:

$$\mathrm{DCG}_K = \sum_{i=1}^{K} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

To enable comparison across queries with varying numbers of relevant documents, this score is normalized by the Ideal DCG (IDCG), which represents the score of a perfect ranking: $\mathrm{nDCG}_K = \frac{\mathrm{DCG}_K}{\mathrm{IDCG}_K}$.

Beyond general ranking quality, specific downstream tasks necessitate specialized metrics. In *open-domain question answering* (Chapter 3), the primary objective is to find at least one passage that contains the answer string, with **Accuracy@K** being defined as whether the answer span appears in the top K retrieved passages or not (Karpukhin et al., 2020). In the distinct context of *tool retrieval* (Chapter 4), the evaluation criteria become stricter. Complex agentic tasks often rely on "compositional intent," requiring a specific set of interdependent tools (e.g., a "Flight Search" tool and a "Hotel Booking" tool) to execute a plan. Retrieving only one of these renders the task unsolvable. To measure this, **Completeness@K** was proposed (Qu et al., 2024), a binary metric that evaluates whether the retrieved set contains all the necessary tools required to solve the user's request. If the set of required tools is T^* and the retrieved set is T_{ret}, Completeness is 1 if $T^* \subseteq T_{ret}$, and 0 otherwise. This metric penalizes partial retrieval heavily, emphasizing the necessity of full functional coverage for agentic workflows.

2.1.5 Retrieval-augmented Generation

Retrieval-augmented generation (RAG) represents an architectural approach that augments language models with explicit access to external knowledge at inference time. This paradigm addresses key limitations of standard LLMs by decoupling the storage of knowledge from the capacity to reason over it. In a standard transformer, factual information must be encoded implicitly within the model's weights—its parametric memory—during the expensive pre-training phase, after which it is typically fixed; incorporating new or corrected facts, for example updating outdated or erroneous information, generally requires additional training, which is itself costly. RAG systems relax this constraint by granting the model access to an external, dynamic knowledge source, typically implemented as a dense vector index over passages or documents.

The mechanics of RAG in modern agentic systems typically follow a *retrieve-then-generate* workflow (Lewis et al., 2020). Given an input query x, the system first utilizes a retriever

to identify a set of relevant passages Z_{ret} from the external corpus. These passages are concatenated with the original query to form an extended context sequence. The generator, conditioned on this enriched prompt, produces the output response y. By making the external context explicitly available to the model, the generation process $P(y|x, Z_{ret})$ is grounded in retrieveable evidence rather than relying solely on the model's internal memorization.

The integration of retrieval is critical for scalable agents due to well-documented deficiencies in purely parametric models. First, parametric memory is static; once a model is trained, its knowledge is frozen in time. RAG allows for the system's knowledge to be updated at inference time by simply modifying the external document index, without incurring the prohibitive cost of re-training the neural network. Moreover, parametric memory is capacity-constrained. Prior work shows that while LLMs efficiently memorize common knowledge effectively, they struggle significantly with *long-tail* knowledge that appears infrequently in pre-training data (Kandpal et al., 2023). By offloading this long-tail information to an external retriever, an agent can access a substantially larger store of details while reserving its limited parametric capacity for learning generalizable reasoning patterns and linguistic syntax. Furthermore, retrieval can improve the reliability of generated outputs. Conditioning generation on retrieved text has been shown to reduce hallucination in retrieval-augmented dialogue settings, as the model is guided to synthesize information present in the provided context rather than producing unsupported completions (Shuster et al., 2021). This grounding can also improve transparency: when outputs are accompanied by the retrieved sources used during generation, users can inspect supporting passages and audit the provenance of model claims, a property emphasized in systems that explicitly collect and present references as part of the generation process (Nakano et al., 2021).

While the primary focus of this thesis is on inference-time retrieval, it is important to note that retrieval can also be integrated into the training objective. For example, one can treat the retrieved document as a latent variable and optimize the marginal likelihood of the target output by summing over possible documents (Lewis et al., 2020; Guu et al.,

2020; Izacard et al., 2023). This end-to-end training aligns the retriever and generator, allowing the system to learn which documents are most useful for generation. However, in the context of general-purpose LLM agents, the dominant paradigm remains the injection of retrieved context at inference time, which supports flexible integration of diverse knowledge sources—from text collections to structured APIs and software tools—without modifying the underlying model weights.

2.2 Tool-using for Large Language Models

Retrieval (described in Section 2.1) and tool use are complementary components in modern LLM agents. Retrieval augments generation with relevant external information, improving factual reliability and enabling access to up-to-date and auditable evidence. Tool use, in contrast, equips the system with interfaces for *action*: it allows an LLM to observe and modify states in external environments and to execute operations with well-defined semantics. Together, retrieval and tools bridge two gaps in purely generative models, and are now standard modules in agentic LLM architectures.

The primary motivation for tool integration stems from the architectural limitations of LLMs itself. At their core, LLMs are trained as next-token predictors, and their outputs are produced via probabilistic decoding rather than by executing symbolic procedures with hard correctness guarantees. In practice, this mismatch becomes significant for tasks requiring rigid, deterministic execution, such as arithmetic and formal logic. Recent mechanistic evidence suggests that LLM arithmetic is implemented not by robust, generalizable algorithms, but by simple memorization or by a collection of heuristic rules that can fail unpredictably outside the regimes where they were learned (Razeghi et al., 2022; Jelassi et al., 2023; Nikankin et al., 2025). These results align with broader findings that Transformer behavior on compositional tasks can degrade sharply as problem complexity increases (Dziri et al., 2023). By offloading such operations to deterministic tools—for example, a Python interpreter or a symbolic solver—

an agent can obtain exact execution without requiring the parametric model to internalize an effectively unbounded space of computations. Program-aided approaches explicitly implement this division of labor by generating executable programs and delegating computation to an external runtime, improving numerical reliability relative to pure text-based generation (Gao et al., 2023b; Chen et al., 2023).

Furthermore, the utility of an agent is defined by its ability to navigate and act in a dynamic environment. Retrieval-augmented generation is fundamentally a *read-only* operation; it provides a snapshot of information but cannot effect change. In contrast, real-world tasks often require *read-write* capabilities, such as modifying a database, sending an email, or booking a reservation. These actions require the model to interface with external services through APIs, and tool-use frameworks study the problem of selecting the correct API and producing valid arguments, where current LLMs can fail by generating incorrect or hallucinated tool calls (Li et al., 2023b; Patil et al., 2023). This capability also extends to inter-agent collaboration, where a generalist LLM can utilize specialized agents or models as tools, modularizing the overall system so that planning and orchestration remain in the LLM while execution is delegated to external components (Shen et al., 2023b).

Finally, the paradigm of tool usage offers significant engineering efficiency compared to pure code generation. While models trained on code can generate programs from natural-language specifications (Chen et al., 2021), this approach is often redundant and error-prone when the software ecosystem already provides optimized and well-tested libraries. It is typically more robust for an agent to invoke a verified function from an established library than to implement equivalent functionality token-by-token. Moreover, in many commercial environments, the underlying logic of a system is proprietary and inaccessible. An agent cannot generate the source code for a closed backend, but it can learn to utilize a public-facing API to perform the required operation. Thus, tool usage represents a shift towards orchestration, where the LLM serves as a controller that leverages existing software infrastructure via tool interfaces.

2.2.1 Software Tools

In the context of agentic LLM systems, a *software tool* is an executable interface that allows a language model to invoke an external computational process and receive its output as an observation. Implementations vary, from local function calls to remote services exposed through REST, but the interaction pattern is similar: the model emits a structured invocation, the tool runs in an external environment, and the result is returned to the model for subsequent reasoning or action selection (Yao et al., 2023). Unlike the model's internal computation, which operates over learned representations, tool use requires producing explicit, structured inputs that conform to a specified interface, such as a typed function signature or a JSON schema.

Figure 2.1 illustrates a tool definition (documentation), which typically includes three components: a unique *tool name* (or identifier) that is used to select the tool; a *description* that states the tool's purpose and the intended usage; and a *parameter schema* that lists required and optional arguments, their types, and descriptions, which enables validation and reduces ambiguity in how the tool should be invoked.

The NLP community has largely converged on JSON-based schemas for representing tool interfaces and tool calls. This choice is pragmatic: JSON is widely used in web applications, and code-trained language models are generally reliable at producing syntactically well-formed JSON objects. Tool-calling APIs therefore commonly specify tools using JSON Schema-like descriptions and require the model to output arguments in a corresponding structured format. In addition to these ad hoc conventions, recent efforts have proposed standardized protocols for exposing tools and context to models such as the model context protocol (MCP). Regardless of the protocol, the central modeling challenge remains the same: mapping natural-language intent to a valid tool selection and a correctly parameterized invocation.

The scope of tools available to modern agents is extensive. Agents may call web APIs to access dynamic data (e.g., weather, prices, schedules) or to trigger state-changing operations in external services. They may also invoke local libraries for numerical computation, data

Tool Documentation #1

```
{
  “name”: “identify_object_type(image_patch: ImagePatch, object_name: str,
object_types: List[str], query: str) -> str”,
  “description”: “def identify_object_type(image_patch: ImagePatch, object_name:
str, object_types: List[str], query: str) -> str:
    ’’’
    Identify the type of a specific object in an image.
    Args:
      image_patch (ImagePatch): The image patch to check.
      object_name (str): The name of the object to identify.
      object_types (List[str]): A list of possible types for the object.
      query (str): The original query to answer.
    Returns:
      str: The type of the object if it exists, otherwise the result of the
simple_query.
    ’’’
    object_patches = image_patch.find(object_name)
    if len(object_patches) == 0:
      # If no object is found, query the image directly with simple_query instead of
returning a long string like “There is no {object_name}.”
      return image_patch.simple_query(query)
    object_patch = object_patches[0]
    return object_patch.best_text_match(object_types)”
}
```

Tool Documentation #2

```
{
  “name”: “BART_Station_information”,
  “description”: “”,
  “category_name”: "Travel",
  “required_parameters”: [
    {
      “name”: “cmd”,
      “type”: “STRING”,
      “description”: “See more examples at http://api.bart.gov/docs/overview/examples
.aspx”,
      “default”: “stns”
    }
  ],
  “optional_parameters”: [],
  “method”: “GET”
}
```

Figure 2.1: Example of software tools with different documentation schema and completeness.

processing, or visualization, leveraging optimized implementations rather than re-generating code for standard routines. In deployed settings, tools often include proprietary internal functions such as database access functions, workflow automation tools, or other application-specific interfaces. Across these cases, the common abstraction is that the tool provides an interface with explicit inputs and outputs that can be composed into a larger agent policy.

2.2.2 Tool-augmented Generation

Integrating tools changes generation from a single, uninterrupted decoding pass into an iterative process that alternates between the language model and an external execution environment. In standard autoregressive generation, the model emits tokens until a termination condition (e.g., an end-of-string (`<eos>`) token) is reached. In tool-augmented settings, generation is instead segmented by explicit signals that indicate a tool invocation, after which control is transferred to an executor that runs the tool and returns the result to the model as additional context.

Figure 2.2 illustrates this interaction loop. The model first encodes the tool name and arguments as context, and during decoding, it generates text that triggers a tool invocation. The environment executes the call and appends the tool output to the context, after which the model continues generation conditioned on the updated history. This pattern of interleaving generation with external actions is widely used in agentic frameworks (Yao et al., 2023).

The mechanism by which a model initiates a tool call varies across implementations but is typically realized through reserved delimiters or structured output formats. For example, some open-source models and inference stacks use explicit tags or control tokens to mark tool-call spans and to serialize arguments, while other systems enforce a structured output channel for tool calls. In all cases, the executor halts free-form decoding upon detecting a tool-call segment, parses the structured payload, executes the tool, and serializes the result back into the prompt so that subsequent tokens can condition on the observation.

At a high level, tool-augmented generation is supported by two paradigms. The first is

Role	Message
System	You are a tool-using AI assistant, and you have access to the following tool(s): 1. {“name”: “get_today_date”, “description”: “Get today’s date.”, “parameters”: {“type”: “object”, “properties”: {}, “required”: []}} 2. {“name”: “weather.fetchDetailedForecast”, “description”: “Provides a detailed weather forecast including atmospheric conditions for a given location and date range.”, “parameters”: {“type”: “dict”, “properties”: {“queryLocation”: {“type”: “string”, “description”: “City or ZIP code to retrieve the forecast for, e.g., 90210 or New York, NY.”}, “dateRange”: {“type”: “dict”, “properties”: {“start_date”: {“type”: “string”, “description”: “Start date for the forecast range, format ’YYYY/MM/DD HH:MM’, e.g., 2023/09/01 00:00.”, “end_date”: {“type”: “string”, “description”: “End date for the forecast range, format ’YYYY/MM/DD HH:MM’, e.g., 2023/09/07 23:59.”}, “required”: [“start_date”, “end_date”]}, “forecastDetails”: {“type”: “array”, “items”: {“type”: “string”, “description”: “Specific details to include in the forecast. Options are ‘precipitation’, ‘cloudCover’, ‘airPressure’.”, “enum”: [“precipitation”, “cloudCover”, “airPressure”]}}}, “required”: [“queryLocation”, “dateRange”]}, “required”: null}
User	What is the chance of raining tomorrow in Cambridge, MA?
Assistant	<think> I need to find today’s date first, and I can use a tool do achieve that. </think> <tool_call>{“tool_calls”: [{“type”: “function”, “function”: {“name”: “get_today_date”, “arguments”:{}}}]}</tool_call>
Tool Execution	<tool_response>2026/01/01</tool_response>
Assistant	<think> Now I can use a weather tool and specify the correct date and location to find precipitation chance. </think> <tool_call>{“tool_calls”: [{“type”: “function”, “function”: {“name”: “weather.fetchDetailedForecast”, “arguments”:{“queryLocation”: “Cambridge, MA”, “dateRange”: {“start_date”: “2026/01/02 00:00”, “end_date”: “2026/01/02 23:59”}, “forecastDetails”: [“precipitation”]}}}]}</tool_call>
Tool Execution	<tool_response>[{“date”: “2026/01/02”, “precipitation”: 0.0}]</tool_response>
Assistant	<think>Now that I have the forecast for tomorrow 1/2, I see that precipitation is 0.0. I can now respond. </think> The chance of raining tomorrow is 0%.

Figure 2.2: Example of LLM performing tool-augmented generation.

native tool use, where the model is trained or fine-tuned on tool-use data so that tool selection and argument generation are learned in the weights. This is the approach adopted by many state-of-the-art LLMs (Achiam et al., 2023; Team et al., 2023; Dubey et al., 2024; Guo et al., 2025a; Yang et al., 2025), which are trained to call a fixed set of built-in tools (e.g., web search, code execution, or image generation) through structured outputs. A representative line of work teaches tool use by augmenting training corpora with tool-call traces, including self-supervised insertion of API calls (Parisi et al., 2022; Patil et al., 2023; Schick et al., 2023; Yang et al., 2023; Lin et al., 2025; Liu et al., 2025). However, native tool use has practical limitations: it requires substantial training compute and curated interaction data, it

typically targets a bounded toolset, and extending coverage to new or rapidly changing tools generally requires additional training, reflecting the broader limitations of purely parametric approaches.

The second paradigm, which is the setting assumed throughout this thesis, is *inference-time prompting*. Here, a general-purpose or native tool-calling model is turned into a general tool-using agent through the construction of the input context rather than weight updates. This setup closely parallels retrieval-augmented generation: just as RAG injects retrieved passages into the prompt to supply external evidence, tool use injects tool specifications and interaction context so the model can select and invoke external capabilities at inference time. In practice, as shown in Figure 2.3, the prompt typically provides the tool definitions, high-level system instructions that constrain the interaction, and in-context tool use example trajectories that illustrate correct tool usage. Prior studies show that tool-use performance is sensitive to the quality of these inputs: insufficient or ambiguous documentation increases hallucinated tool calls and malformed arguments (Hsieh et al., 2023), and removing in-context examples can substantially reduce tool-use success (Xu et al., 2023). This sensitivity motivates the retrieval and prompt construction techniques developed in later chapters.

2.2.3 Tool Retrieval

Because the prompt must carry tool definitions alongside system instructions and in-context examples (Figure 2.3), it grows quickly with the size of the available toolset, and it becomes infeasible to include documentation and in-context examples for all tools within the model's limited context window. As agentic ecosystems scale to hundreds or thousands of APIs, a retrieval stage is therefore required to select a small candidate set of tools to present to the model at inference time.

Adapting standard text retrieval to tools is non-trivial. Tools are specified through rigid schemas, which creates an semantic mismatch between high-level user intents and technical function descriptions; and many tasks require composing multiple tools are hard to capture by

Role	Message
System	You are a tool-using AI assistant, and you have access to the following tool(s): 1. {“name”: “get_today_date”, “description”: “Get today’s date.”, “parameters”: {“type”: “object”, “properties”: {}, “required”: []}} **Example tool use scenarios for this tool** User request: What date is today? Tool call: `get_today_date()` User request: What appointment do I have next Wednesday? Tool call: `get_today_date()` User request: What team won the Super Bowl last year? Tool call: `get_today_date()` 2. {“name”: “weather.fetchDetailedForecast”, “description”: “Provides a detailed weather forecast including atmospheric conditions for a given location and date range.”, “parameters”: {“type”: “dict”, “properties”: {“queryLocation”: {“type”: “string”, “description”: “City or ZIP code to retrieve the forecast for, e.g., 90210 or New York, NY.”}, “dateRange”: {“type”: “dict”, “properties”: {“start_date”: {“type”: “string”, “description”: “Start date for the forecast range, format ’YYYY/MM/DD HH:MM’, e.g., 2023/09/01 00:00.”, “end_date”: {“type”: “string”, “description”: “End date for the forecast range, format ’YYYY/MM/DD HH:MM’, e.g., 2023/09/07 23:59.”}, “required”: [“start_date”, “end_date”]}, “forecastDetails”: {“type”: “array”, “items”: {“type”: “string”, “description”: “Specific details to include in the forecast. Options are ‘precipitation’, ‘cloudCover’, ‘airPressure’.”, “enum”: [“precipitation”, “cloudCover”, “airPressure”]}}}, “required”: [“queryLocation”, “dateRange”]}, “required”: null} **Example tool use scenarios for this tool** User request: What is the weather forecast in Boston, MA on 2026/1/31? Tool call: `weather.fetchDetailedForecast(queryLocation=“Boston, MA”, dateRange={“start_date”=“2026/01/31 00:00”,“end_date”=“2026/01/31 23:59”})` User request: Give me the one week forecast for zip code 02139 starting 2026/02/01. Tool call: `weather.fetchDetailedForecast(queryLocation=“02139”, dateRange={“start_date”=“2026/02/01 00:00”,“end_date”=“2026/02/07 23:59”})` User request: What is the air pressure and cloud cover near Logan airport tomorrow? Tool call: `weather.fetchDetailedForecast(queryLocation= “02128”, dateRange={“start_date”= “2026/02/03 00:00”,“end_date”= “2026/02/03 23:59”}, forecastDetails=[“airPressure”, “cloudCover”])`

Figure 2.3: Example system prompt that includes tool definitions and in-context tool-use examples.

conventional encoders. These properties motivate treating tool retrieval as a distinct retrieval problem, and they underpin the method described in Chapter 4 for improving retrieval over large tool collections.

2.3 Optimization Techniques for Agentic LLMs

The components introduced above—retrieval, tool use, and their integration into agentic workflows—typically require optimization beyond generic pre-training. In practice, models are adapted either by updating parameters with gradient-based learning or by modifying the inputs and interaction context presented at inference time. This section summarizes three families of techniques used throughout this thesis: supervised fine-tuning, reinforcement learning for policy optimization, and prompt optimization.

2.3.1 Supervised Fine-tuning

Supervised fine-tuning (SFT) adapts a pre-trained LLM to a target task using labeled examples. In its simplest form, SFT continues training by maximizing the likelihood of reference outputs under teacher forcing. For an output token sequence $y = (y_1, \ldots, y_T)$ conditioned on an input x, the standard conditional language modeling objective is

$$\mathcal{L}_{\mathrm{NLL}}(\theta) = -\sum_{t=1}^{T} \log P_\theta(y_t \mid y_{<t}, x). \tag{2.1}$$

This objective is natural for generative settings, including instruction following, tool-call formatting, and sequence-to-sequence modeling, where the target behavior is represented directly as text.

For retrieval and ranking models, supervision is often relational rather than purely token-level. Dense retrievers are commonly trained with contrastive objectives that increase similarity between a query and a relevant item while decreasing similarity to negatives, typically constructed via in-batch or mined negative sampling. Cross-encoders and re-rankers are frequently trained with classification or ranking losses defined over query–candidate pairs, again relying on negative sampling to expose the model to near-miss confounders. More generally, a wide range of differentiable objectives can be used to train Transformer models via standard backpropagation (Rumelhart et al., 1986), and the choice of loss follows the

structure of the supervision available for the task.

2.3.2 Reinforcement Learning

SFT is effective when high-quality targets are available, but it is less well-suited to settings where there are many valid solutions, where supervision is naturally defined over outcomes, or where learning requires interaction with an external environment. Reinforcement learning (RL) addresses these cases by optimizing a policy to maximize expected reward, allowing the model to explore alternative action sequences and improve based on feedback from the environment. Let π_θ denote the model policy and let τ denote a sampled trajectory (e.g., an output text sequence or a sequence of tool calls). The objective is

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\big[R(\tau)\big]. \tag{2.2}$$

A basic policy-gradient estimator (REINFORCE) updates the model by increasing the log-probability of trajectories that obtain higher reward (Williams, 1992):

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \underbrace{\big(R(\tau_i) - b\big)}_{\text{advantage}} \underbrace{\nabla_\theta \log \pi_\theta(\tau_i)}_{\text{policy gradient}}, \tag{2.3}$$

where b is a baseline used to reduce variance. In practice, naive policy gradients can be unstable due to high-variance gradient estimates, motivating algorithms that constrain or stabilize updates.

Proximal policy optimization (PPO) is a widely used method that improves stability by limiting how much the policy changes in a single update step (Schulman et al., 2017). PPO optimizes the clipped surrogate objective

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\big(\underbrace{r_t(\theta)}_{\pi_\theta/\pi_{\theta_{\text{old}}}} \underbrace{A_t}_{\text{advantage}}, \underbrace{\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)}_{\text{trust-region proxy}} \underbrace{A_t}_{\text{advantage}}\big)\Big], \tag{2.4}$$

which discourages destructive updates while still allowing improvement. In modern LLM training, RL is used both for preference optimization with learned reward models from human feedback (RLHF) and for settings where rewards can be computed automatically (RLVR). In this thesis, we focus on RL with verifiable rewards and apply it in Chapter 4.

For RLVR, correctness can be validated by execution or other automatic checks, and no separate reward model is required. We adopt group relative policy optimization (GRPO), which simplifies policy optimization by replacing a learned critic with group-relative baselines computed from multiple samples for the same input (Shao et al., 2024). Given G sampled outputs with rewards $\{r_1, \ldots, r_G\}$, GRPO forms a normalized relative advantage

$$A_i = \frac{r_i - \text{mean}(\{r_1, \ldots, r_G\})}{\text{std}(\{r_1, \ldots, r_G\})}, \tag{2.5}$$

encouraging the policy to increase probability mass on higher-reward strategies relative to alternatives generated under the current policy.

2.3.3 Automatic Prompt Optimization

In many practical settings, updating model weights is expensive, constrained by deployment requirements, or impossible when using closed-source models. In these cases, optimization is applied before inference time by modifying the prompt and interaction context. The prompt can be viewed as a controllable input that conditions the model's behavior, so prompt optimization treats the prompt as a tunable object chosen to maximize performance on a task. This perspective underlies a broad family of methods often referred to as automatic prompt engineering or automatic prompt optimization (Shin et al., 2020). In practice, prompt optimization defines a search procedure that scores a candidate prompt on a held-out validation set, then uses this score to guide the search for improved prompts. A common approach is to generate a pool of candidate instructions or prompt variants, evaluate each candidate by running the target model on held-out examples, and select the highest-scoring

prompts for deployment or for further refinement (Pryzant et al., 2023; Wang et al., 2024b). In many recent methods, an LLM is used directly as the optimizer: it proposes revised instructions conditioned on the scores or error patterns observed under previous candidates, forming an iterative generate–evaluate–update loop over prompts (Zhou et al., 2023; Yang et al., 2024). In the tool-use setting emphasized in this thesis, prompt optimization focuses on the pieces of context that govern tool behavior, particularly tool documentation and tool-use examples.

Chapter 3

Generalizing Passage Re-ranking by Mutual Information Maximization

3.1 Introduction

Passage retrieval is a crucial component in open-domain question answering (QA) (Chen and Yih, 2020), a task that requires answering questions from a wide range of domains and could be applied in systems that fulfill user's information needs (Voorhees et al., 1999). Retrieval offers downstream QA systems grounding information, which not only improves accuracy in a lot of cases but also provides transparency to how systems generate answers, similar to how articles provide references and citations, such that model hallucinations can be checked with ease. Furthermore, the set of documents to be retrieved from, or knowledge base, can be quickly updated with new documents and knowledge such that models can adapt to temporal changes, and do not need to be continuously re-trained nor require online training paradigms for continual learning.

As described in Chapter 2.1, early retrieval methods are typically based on term-matching, such as TF-IDF (Sparck Jones, 1972; Salton et al., 1975) or BM25 (Robertson et al., 2009). Such methods, called sparse retrievers, perform keyword matching efficiently with an inverted

index to find relevant contexts. Sparse retrievers often achieve reasonable performance while being computationally efficient and does not require training, but are shown to have limited abilities beyond lexical matching.

Recently, dense retrievers that encode text with continuous embeddings have been heavily studied and utilized in contemporary QA systems, often outperforming their sparse counterparts on high resource evaluation settings (Karpukhin et al., 2020). There are a few drawbacks however, such as higher computational demands during both training and inference, inability to handle large contexts (Luan et al., 2021), and difficulty in generalizing to new domains especially those with limited data (Reddy et al., 2021). Hybrid methods have been explored to get the best of both worlds, generally utilizing an efficient sparse method to retrieve a larger number of possibly relevant contexts, and then perform passage re-ranking with a more computationally-intensive dense model for refined scoring (Nogueira and Cho, 2019).

In this chapter, we focus on passage re-ranking and explore the use of generative models alongside conventional re-rankers. Previous work has explored pre-trained language models as the re-ranking scorer (Sachan et al., 2022), however we find that it underperforms conventional re-rankers for both supervised and zero-shot settings. Starting from maximizing mutual information (MI) for inference, which measures how much more queries and passages co-occur compared to appearing independently, we show how small generative models can be effectively used with conventional cross-encoding re-rankers for improved performance. Experiments on a supervised setting for open-domain QA retrieval and a zero-shot setting across a suite of diverse retrieval benchmarks validate our approach.

3.2 Methodology: Joint Passage Re-ranking (JPR)

Consider the two distributions $p(\boldsymbol{x})$ and $p(\boldsymbol{z})$ over all queries $\boldsymbol{x} \in \mathcal{X}$ and all passages $\boldsymbol{z} \in \mathcal{Z}$. The conditional distributions $p(\boldsymbol{z}|\boldsymbol{x})$ and $p(\boldsymbol{x}|\boldsymbol{z})$ can be used to infer one domain based on the other. The joint distribution $p(\boldsymbol{x}, \boldsymbol{z})$ characterizes the combined structure of both domains,

Passage (z)		
I Can Only Imagine (film) I Can Only Imagine is a 2018 American Christian drama film directed by the Erwin Brothers and written by Alex Cramer, Jon Erwin, and Brent McCorkle, based on the story behind the MercyMe song of the same name, the best-selling Christian single of all time. The film stars J. Michael Finley as Bart Millard, the lead singer who wrote the song about his relationship with his father (Dennis Quaid). Madeline Carroll, Priscilla Shirer, Cloris Leachman, Trace Adkins and Brody Rose also star. "I Can only Imagine" was released in the United States on March 16,		
Query (x)	$\log p_{\boldsymbol{\phi}}(\boldsymbol{z}\|\boldsymbol{x})$	**Relevance**
who produced the movie i can only imagine	-0.882	
who played amy grant in i can only imagine	-0.913	
who wrote the country song i can only imagine	-2.466	✓
who wrote and performed i can only imagine	-2.682	✓
when was i can only imagine the song released	-3.893	
when is i can only imagine coming out	-4.507	

Figure 3.1: Example showing a passage that is estimated to have high retrieval probabilities for multiple queries by a conventional re-ranker. Each query asks about different specifics of a movie; however the passage contains mostly general information, and could not be used to answer several top-ranked questions. This motivates our use of a penalization term to discount these high probability passages that are not specific to the input query.

where $p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x})p(\boldsymbol{z}|\boldsymbol{x}) = p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z})$.

Here $p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ defines a passage retrieval model, which we parametrize by $\boldsymbol{\phi}$, generally trained with maximum likelihood estimation (MLE): $\mathcal{L}_{\text{retrieval}}(\phi) \triangleq -\mathbb{E}_{x,z\sim p(\boldsymbol{x},\boldsymbol{z})}\left[\log p_{\boldsymbol{\phi}}(z|x)\right]$. During inference, finding the most probable relevant passage can be written as:

$$\hat{z} = \arg\max_{z} \log p_{\boldsymbol{\phi}}(z|x). \tag{3.1}$$

Since we focus on passage re-ranking, we treat $p_{\boldsymbol{\phi}}(z|x)$ in Eq. 3.1 as re-ranking scores.

3.2.1 Inference by Maximizing Mutual Information

In passage retrieval, documents are commonly chunked into multiple passages of fixed length, some of which containing summaries or general information that are often estimated to have high probabilities by retrieval rankers but do not contain specifics regarding the given query. One of such example is shown in Figure 3.1. In this chapter, we approach inference by finding

the passage that maximizes the *pointwise mutual information* (PMI) between both domains instead of likelihood:

$$\hat{z} = \arg\max_{z} \Big(\log p(z|x) - \log p(z) \Big). \tag{3.2}$$

We see that maximizing PMI adds a penalizing term compared to MLE in Eq. 3.1, which discounts such passages that unconditionally have a higher probability, and biases the model towards those that are specific to the given query. This formulation conceptually parallels the information-theoretic principles of TF-IDF (Sparck Jones, 1972), where the inverse document frequency (IDF) term serves to penalize ubiquitous terms. By explicitly maximizing PMI, our objective formalizes this intuition for dense semantic representations: the regularization term $-\log P(z)$ acts as a "passage-level IDF," discounting generic, high-probability passages to isolate those that share high mutual information with the specific query. A hyperparameter λ is added to control the regularization term. Using Bayes' theorem, we can rewrite Eq. 3.2 as:

$$\begin{aligned} \hat{z} &= \arg\max_{z} \Big(\log p(z|x) - \lambda \log p(z) \Big) \\ &= \arg\max_{z} \Big((1-\lambda) \log p(z|x) + \lambda \log p(x|z) \Big). \end{aligned} \tag{3.3}$$

The PMI objective is equivalent to the convex combination of the terms $\log p(z|x)$ and $\log p(x|z)$. Notice that the latter term can be viewed as a conditional generation model that gives the probability of generating a query given a passage. We denote the generative model by $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ with parameters $\boldsymbol{\theta}$. This term was previously explored as the sole inference objective in Sachan et al. (2022), in which an LLM was used as a question generator for re-scoring. Instead of using either the retrieval model or the generative model only, as explored in prior work, Eq. 3.3 provides a simple way to use both models jointly for inference, which we refer to as *Joint Passage Re-ranking* (JPR).

3.2.2 Joint Fine-tuning

A straighforward way to obtain the two models that can be used for the aforementioned MI-based inference is to train both models using MLE seperately. The retrieval model can be trained with $\mathcal{L}_{\text{retrieval}}(\phi)$, while the generative model can be a trained with a simple LM loss $\mathcal{L}_{\text{generation}}(\boldsymbol{\theta})$.

However, the terms in Eq. 3.3 are derived when the distributions are matched, that is, when $p(\boldsymbol{x})p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = p(\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$. When the two models are optimized independently, we cannot ensure that this holds. We therefore attempt to enforce this constraint with joint fine-tuning. Similar to previous work on dual supervised learning, we approach this by adding a regularization term, defined as the symmetric KL divergence between the two distributions: $\mathcal{L}_{\text{match}}(\boldsymbol{\phi}, \boldsymbol{\theta}) \triangleq D_{\text{sym-KL}}\big(p_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z})||p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\big)$, by enforcing alignment of the marginals multiplied by the conditional probabilities. The joint fine-tuning objective is obtained by combining all three losses: $\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}) \triangleq \mathcal{L}_{\text{retrieval}} + \mathcal{L}_{\text{generation}} + \alpha\mathcal{L}_{\text{match}}$, where α is a regularization hyperparameter. The additional fine-tuning aligns the two conditional distributions such that the conditions for our derivations hold, thereby enhancing the overall performance.

3.3 Related Work

3.3.1 Passage Re-ranking

Passage re-ranking seeks to combine the advantages of sparse retrieval methods, such as efficiency, precise matching, and low-resource generalizability (Reddy et al., 2021; Sciavolino et al., 2021), with the superior performance of dense methods in the presence of extensive annotated data (Guu et al., 2020; Karpukhin et al., 2020). Early work by Nogueira and Cho (2019) examined BERT-based supervised re-rankers, while later research proposed reader prediction based re-ranking (Mao et al., 2021b) and attempted to use LMs as re-

rankers (Sachan et al., 2022), although with limitations. Sequence-to-sequence models have also been investigated to directly generate ranking labels (Nogueira et al., 2020), and further training with explanations can yield improvements under lower-resource scenarios (Ferraretto et al., 2023). More recently, Sun et al. (2023) explored using the proprietary and exceptionally larger ChatGPT models for re-ranking[1]. Departing from existing ensembling techniques for re-ranking such as fusing bi-encoder embeddings (Lu et al., 2021), our method establishes the combination of discriminative and generative re-rankers through PMI maximization.

3.3.2 Mutual Information Maximization

MI-based objectives, originally introduced in speech recognition to measure input-output dependence (Bahl et al., 1986; Woodland and Povey, 2002), have been applied to different tasks such as dialogue (Li et al., 2016), machine translation (Li and Jurafsky, 2016), and QA (Luo et al., 2022). MI-based joint inference and learning have been explored in question answering and generation (Tang et al., 2017), language understanding and generation (Su et al., 2020), and various vision and language tasks (Xia et al., 2017).

3.4 Experiments: Open-Domain QA Retrieval

3.4.1 Data

First, we evaluate on two standard open-domain QA retrieval benchmark datasets: Natural Questions (NQ; Kwiatkowski et al., 2019) and TriviaQA (Joshi et al., 2017). Wikipedia passages used in DPR (Karpukhin et al., 2020) were used in these experiments, which consists of 21M 100-word passages from the English Wikipedia dump of Dec. 20, 2018 (Lee et al., 2019). Additional dataset information can be found in Appendix A.

[1] Sun et al. (2023) reported results only on a subset of BEIR and uses BM25 "flat" (*cf.* "multifield").

3.4.2 Setup and Baselines

We adopt the setting from prior work using standard dataset splits, retrieving the top 100 passages for re-ranking. We use Pyserini (Lin et al., 2021) for BM25 as the initial retriever, with default Lucene parameters of $k = 0.9$ and $b = 0.4$. We report top-K retrieval accuracy, the standard metric.

We compare JPR against several baselines: 1) cross-encoding re-ranker (BERT-FT), a fine-tuned BERT-based (Devlin et al., 2019) re-ranker, running inference with Eq. 3.1; 2) generative re-ranker (T5-FT), a fine-tuned T5 conditional generation model (Raffel et al., 2020) with the second term of Eq. 3.3 as inference objective; and 3) UPR (Sachan et al., 2022), a generative re-ranker using the larger pre-trained T0-3B model (Sanh et al., 2022).

For our approach, we report one setting with joint inference (JPR), and another with joint fine-tuning followed by the MI-based inference (JPR-FT). Joint inference uses the separately fine-tuned retrieval re-ranker and generative re-ranker described above directly. For joint fine-tuning, we bootstrap with the two models, and further fine-tune with our proposed objective to match the discriminative and generative distributions. λ and α are chosen by performance on the development set.

Furthermore, we aim to explore the effects of scaling generative re-rankers up. We experiment with a large language model (LLM), the 33B-parameter LLaMA (Touvron et al., 2023), as our generative re-ranker for both UPR and JPR.

3.4.3 Training and Inference Details

Training. Generally, conventional cross-encoders are trained to minimize the negative likelihood $\mathcal{L}_{\text{retrieval}}(\phi) \triangleq -\mathbb{E}_{x,z\sim p(\boldsymbol{x},\boldsymbol{z})}\left[\log p_{\boldsymbol{\phi}}(z|x)\right]$, where $p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is usually calculated from the retrieval score of question-passage pairs, with the partition function approximated by a noise contrastive approach trained either with a classification or a ranking objective (Ma and Collins, 2018). We choose to fine-tune our cross-encoder, BERT-FT, using a 6-layer transformer model (Vaswani et al., 2017), which takes the concatenated input of a query

Hyperparameter	NQ		TriviaQA	
	BERT-FT	T5-FT	BERT-FT	T5-FT
learning rate	1e-5	2e-5	1e-5	1.5e-5
batch size	96	64	64	64
α	0.0005	0.0005	0.005	0.005

Table 3.1: Training hyperparameters for NQ and TriviaQA selected by performance on the dev set.

and a passage, with a noise contrastive objective (Mikolov et al., 2013). The 6-layer SBERT model `MiniLM-L-6-v2` we use was previously pre-trained on MS MARCO, which we fine-tune for 2 epochs using the top 32 passages from BM25 on the NQ/TriviaQA training set. We train with a batch size of 128, learning rate of 5e-5, linear warmup and decay with ratio of 0.1.

For training of T5-FT, we fine-tune with $\mathcal{L}_{\text{generation}}(\boldsymbol{\theta})$ using the `t5-base-lm-adapt` model, a 12-layer encoder-decoder configuration with 220M parameters initialized from T5-base v1.1 and trained for an additional 100k steps with an LM objective. It takes a ground truth passage as input with its corresponding query as the decoder target. Ground truth query-passage pairs from the training set was used to fine-tune the model for 2 epochs. We use a batch size of 64, learning rate of 5e-5, and linear warmup and decay ratio of 0.1. Hyperparameters were chosen by performance on the dev set.

UPR uses the pre-trained T0-3B directly without any fine-tuning.

JPR uses BERT-FT and T5-FT, described earlier, directly during inference. JPR-FT requires further fine-tuning, which we train for another epoch. Training hyperparameters were searched with the dev set, with one run for each hyperparameter setting, shown in Table 3.1. We report results for the model with the best-performing run on the dev set.

All models were trained with HuggingFace's `Transformers` library (Wolf et al., 2020), using the AdamW optimizer (Loshchilov and Hutter, 2018) with default parameters. The maximum sequence lengths for queries and passages were set to 128 and 512, respectively, for generative models. For the cross-encoding BERT-FT, we set the maximum concatenated

Re-ranking Method	Cross-Enc? $\log p_{\phi}(z\|x)$	Generative? $\log p_{\theta}(x\|z)$	NQ			TriviaQA		
			Top-1	Top-5	Top-10	Top-1	Top-5	Top-10
BM25	✗	✗	22.1	43.8	54.5	46.3	66.3	71.7
BERT-FT	✓	✗	49.4	66.4	71.4	66.7	77.6	80.2
T5-FT	✗	✓	34.3	59.6	66.7	56.8	74.1	78.0
UPR (T0-3B)	✗	✓	36.8	61.6	68.2	57.7	75.4	78.5
JPR	✓	✓	51.1	**67.5**	**72.1**	68.3	78.3	**80.5**
JPR-FT	✓	✓	**51.5**	67.4	71.8	**69.2**	**78.5**	**80.5**
UPR (LLaMA-33B)	✗	✓	35.0	61.5	69.0	57.2	76.7	79.5
JPR (LLaMA-33B)	✓	✓	48.2	66.9	71.5	70.1	79.3	80.8

Table 3.2: Top-K retrieval accuracy (%) on the Natural Questions and TriviaQA test sets. All non-BM25 methods re-rank the top-100 passages retrieved by BM25. Best overall are in **bold** while best non-LLM are underlined.

length to be 512. Training was done with four Nvidia A6000 GPUs, with around 2.5 GPU hours per epoch, equating to around 250 GPU-hours in total.

Inference. For the conventional cross-encoding re-ranker (BERT-FT), we re-rank with Eq. 3.1 by directly ranking the retrieval scores. When using BERT-FT in JPR, we approximate $\log p_{\phi}(z|x)$ by taking SoftMax over the scores for the 100 retrieved passages. For generative re-rankers T5-FT and UPR, we follow Sachan et al. (2022) and estimate $\log p_{\theta}(x|z)$ with length-normalized conditional likelihood of the output sequence followed by taking SoftMax over the passages. For JPR, the preceding two terms are weight-averaged according to Eq. 3.3.

3.4.4 Results and Discussion

Open-domain QA retrieval results are shown in Table 3.2. Using the conventional cross-encoder BERT-FT on initial BM25 results yields decent improvements. UPR, not fine-tuned but being much larger, significantly underperforms BERT-FT. The fine-tuned generative model T5-FT, 15× smaller than the T0-3B model in UPR, nearly matches the performance of UPR. When using JPR, which corresponds to scoring with Eq. 3.3 using the re-ranker BERT-FT and the generative model T5-FT, surpasses all baselines. The generative model,

although used by itself underperforms BERT-FT, boosts performance especially for the top retrieved passages. Matching distributions (JPR-FT) by fine-tuning for a small amount of steps further improves performance, albeit more modestly. For LLM generative re-ranking, despite being multitudes larger, LLaMA-33B surprisingly underperforms against T5-FT and T0-3B on NQ for both UPR and JPR, however on TriviaQA JPR with LLaMA-33B achieves best overall results.

3.5 Experiments: Zero-Shot Retrieval

3.5.1 Data

We further evaluate in a transfer learning setting on BEIR (Thakur et al., 2021), a commonly used benchmark consisting of a suite of information retrieval datasets that span multiple tasks and domains. The benchmark contains 18 datasets across a variety of text retrieval tasks and domains, containing queries and passages that have significantly different styles and lengths. Moreover, no training data is provided, making it considerably difficult for models to perform well across all datasets. In this chapter, we evaluated on the 14 publicly available ones. Futher details can be found in Appendix A.

3.5.2 Setup and Baselines

We follow BEIR's zero-shot evaluation on all tasks, using MS MARCO (Nguyen et al., 2017) as training data. Pyserini is used for BM25 to retrieve 100 passages, with default parameters and indexing title and passage as separate fields[2,3]. The Normalized Cumulative Discount Gain (nDCG@K) (Wang et al., 2013) is used for evaluation, with $K = 10$, computed by the official TREC evaluation tool (Van Gysel and de Rijke, 2018).

We compare against the three baselines used previously with slight differences: 1) con-

[2]Pyserini reproductions for BEIR can be found at https://castorini.github.io/pyserini/2cr/beir.html.
[3]We follow BEIR and retrieve 100, which is more practical.

Dataset	BM25	Re-ranking Method					
		BERT-FT	T5-FT	UPR	JPR	UPR (LLM)	JPR (LLM)
TREC-DL 2019	50.8	74.9	65.6	-	75.0	-	-
TREC-COVID	65.6	75.7	75.7	76.5	78.2	76.5	77.2
NFCorpus	32.6	35.0	33.2	34.8	35.3	33.5	35.7
NQ	32.9	53.3	43.8	44.5	52.1	45.3	54.0
HotpotQA	60.3	70.7	68.5	70.9	72.4	72.3	72.1
FiQA-2018	23.6	34.7	35.7	42.0	38.5	40.3	36.6
ArguAna	*41.4*	*41.8*	50.2	*50.9*	49.3	28.5	43.3
Touché-2020	36.7	27.1	25.0	21.0	26.8	18.5	25.7
CQADupStack	29.9	37.1	37.7	40.2	39.7	42.9	39.0
Quora	78.9	82.5	81.2	83.6	84.8	84.4	84.1
DBPedia	31.3	40.9	34.6	35.5	40.5	35.1	41.6
SCIDOCS	15.8	16.6	16.9	17.6	18.3	18.1	17.1
FEVER	75.3	81.8	75.7	61.3	82.5	62.5	79.7
Climate-FEVER	21.3	25.3	18.4	14.6	25.2	11.2	24.9
SciFact	66.5	68.8	69.3	70.4	72.7	65.7	70.3
Average	43.7	49.4	47.6	47.4	**51.2**	45.3	50.1

Table 3.3: Zero-shot results on BEIR, scores denote **nDCG@10**. All methods re-rank the top-100 passages retrieved by BM25, except for TREC-DL 2019 to compare to prior work. Best overall are in **bold**. Underlined indicate in-domain performance, and *italicized* are based on Pyserini reproductions, differing from those reported in prior work.

ventional discriminative re-ranker (BERT-FT), using a BERT-based re-ranker pre-trained on MS MARCO with the same configuration (Reimers and Gurevych, 2019); 2) generative re-ranker (T5-FT), using the same `t5-base-lm-adapt` but fine-tuned on MS MARCO; and 3) UPR, but re-ranked over 100 instead of 1000. For our proposed approach, we only evaluate the joint inference method (JPR), as the MS MARCO pre-trained re-ranker from SBERT[4] is already at a saddle point, and using it to bootstrap leads to degraded performance.

3.5.3 Training and Inference Details

For BEIR, since the SBERT model was already pre-trained on MS MARCO, we directly use it for BERT-FT. On the other hand, T5-FT stills requires fine-tuning, which we train for 3 epochs on query-passage pairs in the training set, with batch size of 16 and learning rate of 5e-5 with no warmup. The inference process is the same as open-domain QA retrieval,

[4] https://www.sbert.net/docs/pretrained_cross-encoders.html

described earlier in Sec. 3.4.3, except for λ which we set to 0.5 for all tasks as the BEIR tasks are zero-shot and we do not have access to the validation sets.

3.5.4 Results and Discussion

Zero-shot results on BEIR are presented in Table 3.3. JPR attains roughly 2% absolute gain on average simply by utilizing both discriminative and generative models for inference, which is more prominent when compared against in-domain performances in Sec. 3.4 and on TREC-DL 2019. JPR surpasses BERT-FT on 10 out of the 14 tasks and is roughly equal on the other 4, and eclipses T5-FT on 13 of 14. Notably, for two tasks, FEVER and Climate-FEVER, generative re-rankers struggle and exhibit degraded performance, whereas JPR avoids this issue and outperforms BERT-FT. When using the comparatively huge LLaMA, we see that UPR worsens on average, mostly due to major underperformance on tasks such as ArguAna, Touché-2020, FEVER, and Climate-FEVER. On most other tasks it outperforms UPR, suggesting that larger models' effects may scale both ways, positively on familiar tasks, such as CQADupStack which LLaMA had exposure during LM training, and negatively on a few out-of-domain ones. JPR (LLM) can mitigate the worst cases, however it mostly does not outperform JPR that uses the considerably smaller generative model.

3.5.5 Ablation Studies and Qualitative Analysis

Different Cross-encoding and Generative Model Pairs. We show the efficacy of JPR on NQ by conducting additional evaluations on NQ with various model combinations. We experiment with BERT models of different sizes for the cross-encoders, and for generative models we chose T5 models of multiple models sizes. All cross-encoding models were previously pre-trained on MS MARCO, which we fine-tune on NQ, and the T5 models were fine-tuned on NQ, all following training procedures reported in Sec. 3.4.3. For inference, we follow the inference steps outlined in Sec. 3.4.3. The results are shown in Table 3.4.

From the results, notice that when T5-small is paired with `MiniLM-L-6` for JPR, it aligns

Cross-encoder	Generative Model	#params	Top-1	Top-5	Top-10
TinyBERT	✗	4.4M	37.8	60.3	67.0
`MiniLM-L-4`	✗	19.2M	47.5	65.9	70.9
`MiniLM-L-6` (BERT-FT)	✗	22.7M	49.4	66.4	71.4
BERT-base	✗	109.5M	49.2	66.0	70.8
BERT-large	✗	335.1M	49.8	67.5	71.7
✗	T5-tiny	15.6M	25.7	51.4	62.0
✗	T5-small	77.0M	30.7	57.1	65.2
✗	T5-base (T5-FT)	247.6M	34.4	59.7	66.9
TinyBERT	T5-tiny	20.0M	41.3	62.0	68.8
TinyBERT	T5-small	81.4M	42.6	63.4	69.7
TinyBERT	T5-base	252.0M	43.4	64.0	70.1
`MiniLM-L-4`	T5-tiny	34.7M	48.7	66.1	71.2
`MiniLM-L-4`	T5-small	96.2M	48.3	66.6	71.6
`MiniLM-L-4`	T5-base	266.8M	49.2	66.9	71.7
`MiniLM-L-6`	T5-tiny	38.3M	49.3	67.2	71.6
`MiniLM-L-6`	T5-small	99.7M	50.3	67.3	72.1
`MiniLM-L-6`	T5-base	270.3M	51.1	67.5	72.1

Table 3.4: Top-K retrieval accuracy (%) on NQ for different model combinations with the proposed JPR.

with the performance of T5-base paired with `MiniLM-L-6`. This observation underscores that the additional parameters of T5-base may be superfluous in our application. When comparing JPR (`MiniLM-L-6` & T5-small) with the standalone BERT-base, which is in the same parameter ballpark, and the larger BERT-large, it's evident that the gains from JPR are not solely attributable to model size.[5]

Comparison to Ensembling and Fusion. As JPR utilizes outputs from multiple models, we further compare it to similar methods such as ensembling outputs of different models and with different fusion strategies. In Table 3.5 we show such comparisons. Against the ensemble of `MiniLM-L-6` and BERT-base, where we simply replaced the generative scorer in JPR with a fine-tuned BERT-base, JPR with both T5-base and T5-small outperforms, suggesting that the improvement is not just from ensembling. We further compare against two alternative

[5]We also note that generative scorers are not generally costlier computationally than their cross-encoding counterparts of similar size due to their teacher-forcing nature rather than sampling

Method	#params	Top-1	Top-5	Top-10
BERT-base	109.5M	49.2	66.0	70.8
JPR	270.3M	51.1	67.5	72.1
JPR (w/ T5-small)	99.7M	50.3	67.3	72.1
`MiniLM-L-6 & BERT-base ensemble`	132.2M	49.7	66.5	71.1
`MiniLM-L-6 & T5-base w/ prob-avg`	270.3M	48.6	66.5	70.7
`MiniLM-L-6 & T5-base w/ rank-avg`	270.3M	46.1	65.8	70.6

Table 3.5: Top-K retrieval accuracy (%) on NQ for JPR compared to ensembling and fusion methods.

fusion strategies, with the same underlying models as JPR. *prob-avg* denotes averaging in the probability space rather than the log-space as we derived, and *rank-avg* denotes averaging the ranks. The performance for both strategies are inferior, which suggests that maximizing PMI as JPR does in the log-space is fundamentally more sound.

Qualitative Analysis We use two queries from the example shown earlier in Figure 3.1 to demonstrate how maximizing PMI aids in retrieval re-ranking. In Figure 3.2, the passage for the film "I Can Only Imagine" that contains the title and introduction is too generic and uninformative with respect to the query asking about the producer. It is ranked the highest by both the initial sparse retrieval and subsequent cross-encoding re-ranking. The actual relevant passage about the producer is ranked third, with log probabilities far below the incorrect passage. JPR instead penalizes it, and the new PMI-based scoring re-ranks the target passage to the top.

In Figures 3.3 and 3.4, the related song with the same name which the movie was based on is being queried. Although the top-3 retrieved passages are relevant originally (by the cross-encoder), a number of relevant passages are ranked much lower. Instead, in ranks 4 and 5 are two high $p(z)$ passages, likely due to the high correlation between the terms "imagine" and "song" and the singer John Lennon, and the previously mentioned uninformative film passage. Using PMI to re-rank can greatly help out, successfully recovering relevant passages in ranks 4-8.

User Query who produced the movie i can only imagine

Passage	$\log p_{\phi}(\boldsymbol{z}\|\boldsymbol{x})$	Old rank	PMI	New rank	Relevance
I Can Only Imagine (film) I Can Only Imagine is a 2018 American Christian drama film directed by the Erwin Brothers and written by Alex Cramer, Jon Erwin, and Brent McCorkle, based on the story behind the MercyMe song of the same name, the best-selling Christian single of all time. The film stars J. Michael Finley as Bart Millard, the lead singer who wrote the song about his relationship with his father (Dennis Quaid). Madeline Carroll, Priscilla Shirer, Cloris Leachman, Trace Adkins and Brody Rose also star. "I Can Only Imagine" was released in the United States on March 16,	-0.88	1	3.29	2	
1 film in DVD sales and rentals for the week ending June 16, 2018. I Can Only Imagine (film) I Can Only Imagine is a 2018 American Christian drama film directed by the Erwin Brothers and written by Alex Cramer, Jon Erwin, and Brent McCorkle, based on the story behind the MercyMe song of the same name, the best-selling Christian single of all time. The film stars J. Michael Finley as Bart Millard, the lead singer who wrote the song about his relationship with his father (Dennis Quaid). Madeline Carroll, Priscilla Shirer, Cloris Leachman, Trace Adkins and Brody Rose also	-0.96	2	3.25	3	
Erwin Brothers Andrew and Jon Erwin, known as the Erwin Brothers, are American Christian film directors, screenwriters and film producers known for such films as "Woodlawn", "October Baby", "Moms' Night Out" and "I Can Only Imagine". Their most successful movie to date is "I Can Only Imagine", which grossed a worldwide total of $83.5 million against a production budget of $7 million, with only 1629 theaters for its opening week. The Erwin brothers are the grandsons of Henry Erwin a Medal Of Honor recipient from World War II. In 2017 the brothers and their filmmaking partner Kevin Downes announced the	-2.33	3	3.35	1	✓
100, based on 8 critics, indicating "generally unfavorable reviews". Audiences polled by CinemaScore gave the film an average grade of "A+" on an A+ to F scale, one of fewer than 80 films in the history of the service to earn such a score. "The Arizona Republic"s James Ward gave the film 4/5 stars and wrote, "Too often faith-based films — say anything with Kirk Cameron or the terrible "God's Not Dead" series — tend to preach to the choir or hector their audience. The Erwins' films — "I Can Only Imagine" definitely among them — are more inclusive, charitable	-3.43	4	2.99	4	
Dennis Quaid. The film was released on March 16, 2018. I Can Only Imagine (MercyMe song) "I Can Only Imagine" (sometimes shortened to "Imagine") is a single recorded by Christian rock band MercyMe. Written and composed by lead vocalist Bart Millard, the song, based around a main piano track, was inspired by the death of Millard's father and considers what it would be like in Heaven and to be standing before God. The song was first issued as a track on MercyMe's 1999 album "The Worship Project", which was released on an independent record label. The song was re-recorded and	-4.07	5	2.43	5	

Figure 3.2: Example of JPR taken from NQ's test set. PMI denotes $\log p_{\phi}(\boldsymbol{z}|\boldsymbol{x}) - \lambda \log p_{\phi,\theta}(z)$.

3.6 Chapter Summary

In this chapter, we introduce a simple and effective approach to enhance re-ranking for passage retrieval. We propose *Joint Passage Re-ranking* (JPR), a method utilizing both a cross-encoder and a generative model in the retrieval re-ranking process, optimizing the

User Query when was i can only imagine the song released

Passage	$\log p_{\phi}(\boldsymbol{z}\|\boldsymbol{x})$	**Old rank**	**PMI**	**New rank**	**Relevance**
Dennis Quaid. The film was released on March 16, 2018. I Can Only Imagine (MercyMe song) "I Can Only Imagine" (sometimes shortened to "Imagine") is a single recorded by Christian rock band MercyMe. Written and composed by lead vocalist Bart Millard, the song, based around a main piano track, was inspired by the death of Millard's father and considers what it would be like in Heaven and to be standing before God. The song was first issued as a track on MercyMe's 1999 album "The Worship Project", which was released on an independent record label. The song was re-recorded and	-1.11	1	3.09	1	✓
I Can Only Imagine (MercyMe song) "I Can Only Imagine" (sometimes shortened to "Imagine") is a single recorded by Christian rock band MercyMe. Written and composed by lead vocalist Bart Millard, the song, based around a main piano track, was inspired by the death of Millard's father and considers what it would be like in Heaven and to be standing before God. The song was first issued as a track on MercyMe's 1999 album "The Worship Project", which was released on an independent record label. The song was re-recorded and included on their 2001 major-label debut album "Almost There" as	-1.34	2	3.04	2	✓
as a wake-up call for Barry E. Wilmore during STS-129. The original version of "I Can Only Imagine" was a track on MercyMe's 1999 independent release "The Worship Project". In August 2006, both an acoustic and live form (as well as the original 1999 version) were included in the 'Platinum edition' of "Almost There". MercyMe recorded a version of the song for their "iTunes Originals" album. In 2009, two further variants were included on their compilation album "10"; a 'symphony version' featuring the London Symphony Orchestra, and a live version. "I Can Only Imagine" has also been covered by several	-2.52	3	2.69	3	✓
the sessions for "Imagine" three years later. The melody was inspired by Lonnie Donegan's "Lost John", a song Lennon enjoyed and played often. The song was recorded on 25 May 1971 at Ascot Sound Studios. Robert Christgau called it "an instant folk song worthy of Rosie & the Originals". EMI wanted to release the song as a single but Lennon refused. The only single issued from "Imagine" was the title track in the United States; none was issued in the United Kingdom. Oh Yoko! "Oh Yoko!" is a 1971 song, written and performed by John Lennon, that can be found	-3.76	4	1.66	30	
I Can Only Imagine (film) I Can Only Imagine is a 2018 American Christian drama film directed by the Erwin Brothers and written by Alex Cramer, Jon Erwin, and Brent McCorkle, based on the story behind the MercyMe song of the same name, the best-selling Christian single of all time. The film stars J. Michael Finley as Bart Millard, the lead singer who wrote the song about his relationship with his father (Dennis Quaid). Madeline Carroll, Priscilla Shirer, Cloris Leachman, Trace Adkins and Brody Rose also star. "I Can Only Imagine" was released in the United States on March 16,	-3.89	5	1.97	15	
be standing before God. Regarding this theme, Millard explained to Fox News that "I was always told that if he could choose, he would rather be in Heaven than here with me. As a Christian I believed that, but as an 18-year-old it was a little hard to swallow. So the questions in the song came from me asking God what was so great about Him that my dad would rather be there." "I Can Only Imagine" was re-recorded for their major-label debut record "Almost There" and released as its lead single in 2001. The album was recorded in various	-3.90	6	2.40	4	✓

Figure 3.3: Example of JPR taken from NQ's test set. PMI denotes $\log p_{\phi}(\boldsymbol{z}|\boldsymbol{x}) - \lambda \log p_{\phi,\theta}(z)$.

User Query	when was i can only imagine the song released					
Passage	$\log p_{\phi}(z\|x)$	**Old rank**	**PMI**	**New rank**	**Relevance**
at the time. Millard began writing the words "I can only imagine" on items when he was thinking about his father. During the recording of the band's 1999 independent album "The Worship Project", MercyMe needed one more song to fill out the album. Millard, alone on a bus in the middle of the night, finally wrote the lyrics to the song by drawing on his thoughts and personal faith about what one would experience standing before God in Heaven. Millard attests that "['I Can Only Imagine'] is one of the only songs I have ever written where there wasn't any	-3.90	7	2.19	8	✓
United States. "I Can Only Imagine" grossed $83.4 million in the United States, and Canada and $1.8 million in other territories, for a worldwide total of $85.2 million, against a production budget of $7 million. It is the fourth-highest grossing music biopic of all-time in the United States, behind "Bohemian Rhapsody", "Straight Outta Compton" and "Walk the Line". It is also the highest-grossing independent film of 2018. "I Can Only Imagine" was released on March 16, 2018, alongside "Tomb Raider" and "Love, Simon", and was originally projected to gross $2–4 million from 1,620 theaters in its opening weekend. However, after	-3.98	8	1.96	17	
band would release one more independent album, 2000's "Look", before signing with INO Records and releasing their 2001 album "Almost There". Two songs from "The Worship Project" were re–recorded and included on "Almost There" – "I Can Only Imagine" and "Cannot Say Enough". "I Can Only Imagine" was released as the album's second single and became the band's breakthrough hit, topping the US Christian radio charts and receiving a GMA Dove Award for "Song of the Year" before becoming a hit on US mainstream radio as well. It became the first Christian song to be certified double platinum by the	-4.12	9	2.33	5	✓
edition, CCM Magazine listed Almost There as one of '100 Albums You Need to Own'. In the following year, the previous magazine, ranked "I Can Only Imagine" as the fourth-greatest song in Christian music. At the 33rd GMA Dove Awards, "I Can Only Imagine" won the awards for Song of the Year and Pop/Contemporary Recorded Song of the Year. Almost There (album) Almost There is the first major-label studio album by the American Christian rock band MercyMe. Produced by Pete Kipley, the album was released on August 14, 2001 by INO Records. After releasing six independent records, the band decided	-4.20	10	2.30	6	✓
of the song at radio led to initial album sales that were lower than anticipated, although the album would later be certified double platinum by the Recording Industry Association of America (RIAA) following the success of the album's second single, "I Can Only Imagine". Credits from the album liner notes) Bless Me Indeed (Jabez's Song) "Bless Me Indeed (Jabez's Song)" (sometimes called "Bless Me Indeed") is a song by Christian rock band MercyMe. Written by the band and produced by Pete Kipley, it was released as the lead single from the band's 2001 album "Almost There". The song was written	-4.53	14	2.04	13	✓
the fifth song on the album. "I Can Only Imagine" was released in 2001 as the album's lead single. It gained significant airplay on Christian radio formats before crossing over to mainstream radio formats such as adult contemporary and Top 40 in late 2003 and into 2004; to aid in promotion to these markets, a double A-side physical single (combined with "Word of God Speak") was released in 2003. It charted on several formats, including the "Billboard" Adult Contemporary (where it peaked at No. 5) and the Hot 100 (where it peaked at No. 71). In 2002, "I Can Only	-4.54	15	2.30	7	✓

Figure 3.4: Example of JPR taken from NQ's test set (continued from Figure 3.3). PMI denotes $\log p_{\phi}(z|x) - \lambda \log p_{\phi,\theta}(z)$.

pointwise mutual information directly between query and passage distributions to retrieve passages that are more informative and specific to a given query. We demonstrate that JPR outperforms conventional re-rankers and generative scorers in open-domain QA retrieval evaluation and diverse zero-shot retrieval datasets, and corrects certain failures of those models.

Chapter 4

Retrieving Tools for LLM Tool-use via Query Planning

4.1 Introduction

Large language models 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 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, 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.

4.2 Methodology: Multi-step Tool Retrieval with Query Planning (ToolQP)

4.2.1 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

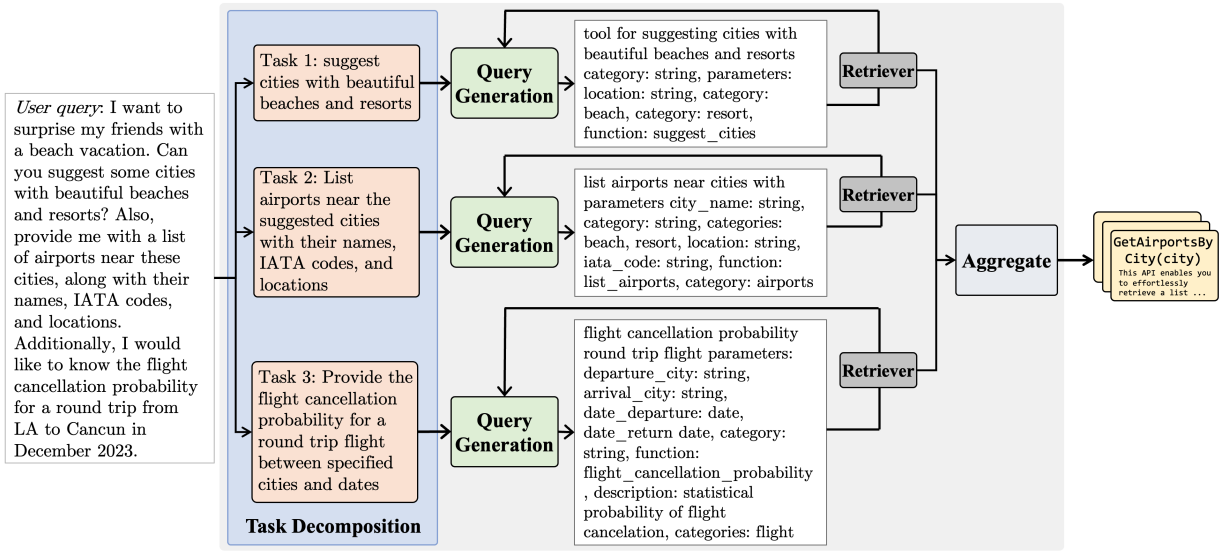

Figure 4.1: Overview of the ToolQP framework. The Planner decomposes a complex user query (e.g., travel planning) into sequential sub-tasks. For each sub-task, it interactively generates queries, processes feedback from the dense retriever, and self-corrects if necessary, before aggregating the final set of relevant tools.

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 for 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 Figure 4.1.

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 has explored reciprocal rank fusion 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-tasks.

4.2.2 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 threshold 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

Algorithm 1 TRAJECTORYGENERATION

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

$\{(s_n, \mathcal{T}_n^\star)\}_{n=1}^{K} \leftarrow \textsc{Parse}_{\mathcal{M}}(q, \mathcal{P})$ ▷ Parse sub-tasks
$\mathcal{H} \leftarrow [\mathcal{P}, s_1, \ldots, s_K]$
for $n \leftarrow 1$ **to** K **do**
  $\mathcal{A} \leftarrow [\,]$
  while $\textsc{AvgRank}(\mathcal{T}_n^\star, O_t) > r_\tau$ **do**
    $c \leftarrow \textsc{AddMoreInfo}(c, \mathcal{T}_n^\star)$ ▷ c: context
    $Q_{\text{Cand}} \leftarrow \{\textsc{Generate}_{\mathcal{M}}(q, s_{1:n-1}, c)\}_{i=1}^{N}$
    $\mathcal{O}_{\text{Cand}} \leftarrow \{\textsc{Retrieve}(Q_{\text{Cand}}^{(i)})\}_{i=1}^{N}$
    $i^\star \leftarrow \text{Best}_{i \in \{1,\ldots,N\}}\ \textsc{AvgRank}(\mathcal{T}_n^\star, \mathcal{O}_{\text{Cand}}^{(i)})$
    $q_t, O_t \leftarrow Q_{\text{Cand}}^{(i^\star)}, \mathcal{O}_{\text{Cand}}^{(i^\star)}$
    $\textsc{Append}(\mathcal{A}, (q_t, O_t))$
  end while
  if $\textsc{Bern}(p)$ **then** ▷ Keep *failed* attempts with prob. p
    $\textsc{Extend}(\mathcal{H}, \mathcal{A})$
  else
    $\textsc{Append}(\mathcal{H}, (q_t, O_t))$
  end if
end for
return $\mathcal{H}$

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. We show an example query trajectory sequence in Figure 4.2.

4.2.3 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

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 4.2: Example trajectory constructed via Algorithm 1 for the SFT of ToolQP.

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.

4.3 Related Work

4.3.1 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 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.

4.3.2 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., 2023a; Jagerman et al., 2023; Mackie et al., 2023; Shen et al., 2023a; Wang et al., 2023) or expansion term selection (Mao et al., 2021a; 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.

4.4 Experiments

4.4.1 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 (Qin et al., 2024b), ToolACE (Liu et al., 2025), and APIGen Liu et al. (2024b) training sets. The original training set contains roughly 200k instances, which we subsample and select 10k for data generation and training. 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.

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., 2023a); (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: A `Qwen3-1.7B` model fine-tuned to predict initial plans as expansions; and (b) ContrastiveFT: `gte-Qwen` fine-tuned via contrastive learning.

We include the implementation details for baselines used in our evaluation in Table 4.1. In Appendix B, we include all prompts used. 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.

Implementation of ToolQP. We implemented two versions of ToolQP, both fine-tuned with the lightweight `Qwen3-1.7B`. The proposed method is denoted as ToolQP, where

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. B.3, B.4
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. B.5, B.6
HyDE (Gao et al., 2023a)	ZS/PRF Prompting	`Qwen3-30B`	Dense fusion of $N = 5$ hypothetical documents *and* user query.	See Figs. B.5, B.6
D2Q (Nogueira et al., 2019)	ZS Prompting	`Qwen3-30B`	Averaged embeddings of $N = 10$ hypothetical queries concatenated to tool documentation.	See Fig. B.7
Re-Invoke (Chen et al., 2024b)	FS Prompting	`Qwen3-30B`	Same embeddings as D2Q.	See Figs. B.7, B.8
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. B.9

Table 4.1: Details of baselines used in ToolQP experiments.

we trained on 10k synthetic trajectories generated by `gpt-4.1-mini` (Achiam et al., 2023) based on ToolRet's training set. The probability p of keeping failed attempts is set to 0.4, and the rank threshold r_τ is set to 5. Prompts for the teacher model used for trajectory synthesis are provided in Appendix B. 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.

For 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.

For 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: $\beta_{1,n} = 5.0, \beta_{1,r} = 2.5, \beta_{2,f} = 1.5, \beta_{2,s} = 0.6, \beta_3 = 1.0$.

Results. Table 4.2 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

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 4.2: Results on ToolRet, a benchmark of 35 datasets. Web⋆ denotes the zero-shot Web datasets. nDCG@10 is denoted as N@10, and Completeness@10 is denoted as C@10.

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.

4.4.2 Transfer to Unseen Base Retrievers

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

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. (2024a)	7B	intfloat/e5-mistral-7b-instruct
BM25	Robertson et al. (2009)	-	Pyserini (Lin et al., 2021)

Table 4.3: Base retriever details for the retriever transfer setting.

`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 overfit 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: `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 et al., 2009). Details for each model are provided in Table 4.3. We reported the averaged improvements (of nDCG@10 and Completeness@10) of all 5 models for each baseline method and ToolQP. Detailed breakdowns are provided in Appendix B.

Results. As shown in Table 4.4, ToolQP and ToolQP-Format consistently outperform baseline expansion methods, even when the inference environment differs from training. ToolQP-Format achieves ∼10% gains in-domain, while ToolQP obtains ∼6-7% 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.

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 4.4: 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.

4.4.3 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. 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 instead of providing a set of ground truth tools. For level-2, we follow the official evaluation setting, where the downstream LLM only has access to `ToolSearch`, the tool retriever wrapped as a tool. For level-1, 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 quite common 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.

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 requiring multiple steps of reasoning and tool call execution. We evaluated with ReAct (Yao et al., 2023) as the baseline agentic

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 4.5: 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.

inference method, and report the solvable pass rate[1]. `Qwen-30B` is utilized as the ToolEval LLM judge, and the `stabletoolbench/MirrorAPI-Cache` model as the tool simulation model. Other settings follow the official inference script default, and we set $k = 5$ to including the 5 top-retrieved tools.

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.

Results. Table 4.5 highlights the downstream impact of improved retrieval. On the smaller API-Bank, ToolQP achieves the highest accuracy with a ∼3-6% accuracy gain. For level-1, the gain is larger as the top-5 retrieved tools, retrieved with the initial user query, are provided to the downstream LLM at the beginning of each dialogue, which is likely due to smaller training-testing setting discrepancy. For level-2, the downstream LLM is given the `ToolSearch` tool, the tool retriever wrapper; the queries in this case are generated by the downstream LLM and not the initial user query, resulting in a larger query distribution shift and comparatively lower gains. Overall, the performance improvements on API-Bank is

[1]Solvable pass rate (SoPR), defined in Guo et al. (2024), uses an LLM evaluator that categorizes answers as *Solved*, *Unsure*, or *Unsolved*, which respectively contribute scores of 1, 0.5, and 0 to the average SoPR calculation.

Method	In-Domain		Zero-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**

Table 4.6: Ablation studies for ToolQP.

notable as ToolQP is designed to operate and excel for large toolsets and not specifically 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 again.

4.4.4 Ablation Studies

We conduct the following ablation studies to validate our design choices, with results summarized in Table 4.6.

Prompting vs SFT vs RLVR. We first investigate whether explicit training is necessary by comparing ToolQP against ToolQP-Prompting (see Table 4.1 for details), 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 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 complicated 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. Finally, we test whether we can rely solely on the queries for sub-tasks. 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.

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

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 4.3: Retrieval comparison for example taken from ToolRet's test set, showing that ToolQP discovers dependencies.

4.4.5 Latency Analysis

4.4.6 Qualitative Analysis

Figure 4.3 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.

Figures 4.4 and 4.5 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 generates queries that target such a calculation tool specifically for organizations.

In Figures 4.6 and 4.7, 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 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

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 4.4: Retrieval comparison for example taken from ToolRet's test set, showing ToolQP finding two correct tools for a single sub-task.

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	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 4.5: Retrieval comparison for example taken from ToolRet's test set, showing ToolQP finding a tool that other methods fail to rank high.

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>`
	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>`
	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 4.6: Retrieval comparison for example taken from ToolRet's test set, showing the benefit of planning sub-tasks with ToolQP.

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>`
	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>`
	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 learning 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 4.7: Retrieval comparison for example taken from ToolRet's test set, which shows fine-grained and targeted queries for each sub-task help retrieve the target tools accurately.

4.5 Chapter Summary

In this chapter, we introduce ToolQP, a framework that formulates tool retrieval as an iterative planning process. By decomposing instructions and leveraging interactive feedback,

ToolQP facilitates the resolution of complex, compositional queries and facilitate the discovery of inter-tool dependencies that single-shot dense retrievers are inherently limited in addressing. This is enabled through a robust training pipeline using synthetic trajectories with SFT and further refinement with 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.

Chapter 5

Prompt Optimization for LLM Tool Use

5.1 Introduction

Recently, there has been growing research interest in agentic large language model frameworks, where, rather than having LLMs answer requests and queries from their own knowledge, LLMs can call a set of external tools with specialized capabilities. This allows LLMs to address more complex tasks and produce responses more accurately (Mialon et al., 2023; Qin et al., 2024a). One key challenge in developing agentic LLM frameworks is how to *dynamically learn to use new, user-defined tools*, which is crucial because having a fixed, general-purpose tool set is often insufficient for real-world scenarios requiring domain-specific functionalities.

The existing mainstream paradigm for dynamically incorporating new tools is by supplementing user-defined tools at inference time in a zero- or few-shot manner via prompting (Lu et al., 2023; Shen et al., 2023b), leveraging zero-shot tool-calling capabilities of current LLMs that have been tuned with tool-use instructions. The success of this paradigm depends on providing sufficient information about the new tool in the prompt. Specifically, existing approaches generally rely on two types of information: 1) *Comprehensive tool documentation* detailing the tool's functionalities and input/output formats, and 2) *In-context demonstrations* that include example queries and corresponding tool calls (Hsieh et al., 2023; Patil et al.,

2023). Inadequate documentation can lead to failures in tool usage, such as syntax errors in both zero-shot and fine-tuned models (Zhang et al., 2023), hallucinations due to incomplete or incorrect tool documentation (Hsieh et al., 2023), and diminished performance resulting from inadequate demonstrations (Xu et al., 2023).

However, in many practical scenarios, it is often not realistic to rely on users, who are often non-experts in AI, to provide adequate documentation for their tools, nor to craft tool-use examples. When users do provide documentation, it may lack crucial details needed for LLMs to call the tools correctly. While automatic prompt optimization techniques (Wang et al., 2024b) could enhance tool documentation, they still require sufficient tool-use examples, which are unavailable in true zero-shot settings. In short, without tool-use examples, neither polished documentation nor in-context demonstrations can be supplied, leading to significant performance degradation in integrating new tools.

To address these challenges in zero-shot tool utilization, we introduce PLAY2PROMPT, an automated framework that generates both high-quality tool documentation and tool-use demonstrations, as illustrated in Figure 5.1. Unlike prior work, PLAY2PROMPT does not rely on any external labeled examples. Instead, it systematically interacts with the new tools—mimicking human trial-and-error—and observes both successful and failed attempts to gather evidence about each tool's correct usage. Using insights from this "tool-play", PLAY2PROMPT creates example demonstrations and refines the tool documentation that better guide LLMs in subsequent inference.

PLAY2PROMPT consists of two steps. In Step 1, a set of tool-use examples are generated via a trial-and-error process, where an LLM agent iteratively calls the target tool with different invocation parameters until a correct invocation is found. Then, for each correct tool invocation instance, a query is generated such that it can be answered by the tool invocation, forming a question-answer pair as a tool-use example. In Step 2, the tool documentation is refined, using the generated tool-use examples as a validation set. In both steps, we employ self-reflection (Madaan et al., 2023; Pryzant et al., 2023; Shinn et al., 2023) to generate error

feedback, thereby directing the search algorithm towards progressively improved outputs. Because PLAY2PROMPT operates entirely in a zero-shot manner and is inherently task-agnostic, it offers a practical and scalable solution for enhancing LLM tool utilization without additional labeled data or human intervention.

We evaluate PLAY2PROMPT with benchmark on real-world scenarios. On the Berkeley Function-Calling Leaderboard (Yan et al., 2024) and the StableToolBench benchmark (Guo et al., 2024), our approach consistently surpasses baseline methods for both open (Dubey et al., 2024; Liu et al., 2025; Lin et al., 2025) and closed models (Achiam et al., 2023). Extensive experiments and analyses further underscore the robustness of our approach.

5.2 Methodology: Zero-shot Prompt Optimization via Tool Play (PLAY2PROMPT)

5.2.1 Problem Formulation and Notation.

A typical agentic LLM framework contains two components: 1) A task LLM, denoted as $\mathcal{M}_T$, and 2) a set of tools, $\mathcal{F} = \{F_{1:K}\}$, where K represents the number of tools, and $1:K$ represents a set of indices running from 1 to K. Given an input query x, rather than directly answering it based on its own knowledge, the task LLM $\mathcal{M}_T$ first selects a sequence of N tools, $(F_{k_{1:N}})$, generates the appropriate input parameters, I_{k_n}, to call each tool, *i.e.*, $F_{k_n}(I_{k_n})$, and finally produces the answer y based on the tool outputs.

We consider the setting where the tool list $\mathcal{F}$ is ad-hoc and dynamic. In order for the task LLM to learn to use the tools in $\mathcal{F}$ at inference time, we follow the conventional prompting-based pipeline (Lu et al., 2023; Hsieh et al., 2023), where the prompt, in addition to an instruction, contains the following tool-specific information:

- **Tool Documentation**, denoted as $\mathcal{D} = \{D_{1:K}\}$;
- **In-context examples of tool-use**, denoted as $\mathcal{E} = \{E_{1:M}\}$, where each example E_m

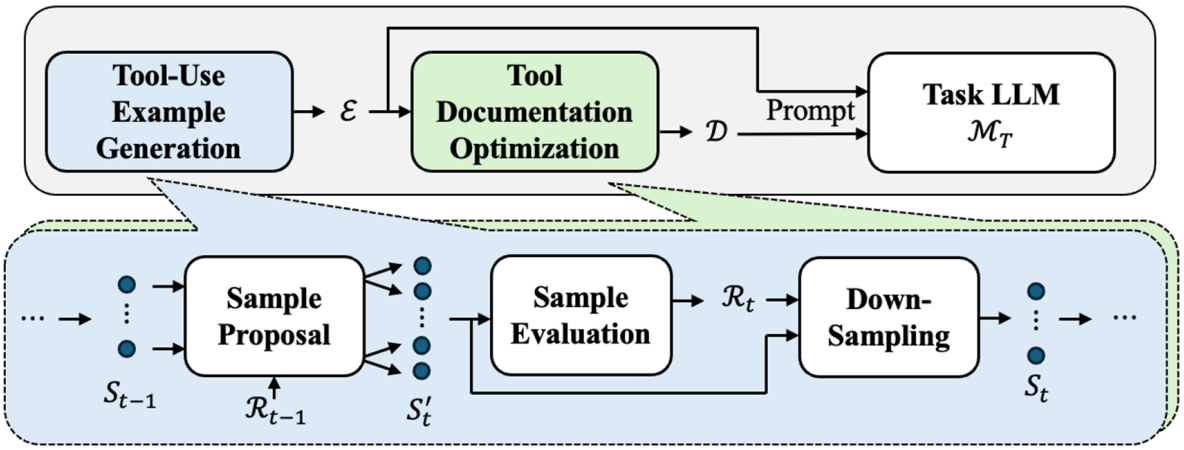

Figure 5.1: The PLAY2PROMPT framework: Beam search iteratively searches tool-use examples, incorporating tool play into the proposal process. After examples are generated, beam search is once again applied to optimize documentation by incorporating tool-use outputs and errors. Finally, optimized tool-use examples and documentation are used as prompts for $\mathcal{M}_T$ at inference.

contains an input query, x, the tool invocation details, $(F_{k_{1:N}}, I_{k_{1:N}})$, and the answer, y.

In many real-world scenarios where users supply their own tool list, it is unrealistic to require them to supply high-quality documentation or tool-use examples. To model these scenarios, we adopt a challenging zero-shot setting with the following constraints: 1) The initial documentation $\mathcal{D}_0$ is sub-optimal and may lack important details, and 2) No tool-use examples $\mathcal{E}$ are available. Given these constraints, our objective is to generate tool-use examples and refine the tool documentation to enhance the task LLM's tool-use performance.

5.2.2 PLAY2PROMPT Overview

The primary goal of PLAY2PROMPT is to utilize knowledge gained from tool interactions to optimize tool documentation and example demonstrations. PLAY2PROMPT consists of the following two steps. *First*, a tool-use example set, $\mathcal{E}$, is generated. *Second*, using $\mathcal{E}$ as the validation set, a refined tool documentation $\mathcal{D}$ is generated based on the initial one $\mathcal{D}_0$. After the two steps are accomplished, $\mathcal{D}$ and a subset of $\mathcal{E}$ will be fed as the prompt to the task LLM during inference.

In both steps, we adopt a tree search framework similar to Wang et al. (2024b) to generate

tool-use examples or tool documentation. Therefore, we will first briefly introduce the beam search framework (see Figure 4.1) in Section 5.2.3, and then elaborate on the two steps in Sections 5.2.4 and 5.2.5, respectively.

5.2.3 Search Framework

The beam search framework is an iterative algorithm to generate high-quality samples, denoted $\mathcal{S} = \{S^{(i)}\}$, which correspond to $\mathcal{E}$ in step 1, and $\mathcal{D}$ in step 2. At the start of iteration t, the algorithm has access to $\mathcal{S}_{t-1} = \{S_{t-1}^{(i)}\}$, which are samples generated in the previous iteration, as well as their corresponding reward $\{R_{t-1}^{(i)}\}$, which depicts the quality of each sample. Then iteration t involves the following procedure to generate a set of improved samples:

• **Sample Proposal.** For each old sample, $S_{t-1}^{(i)}$, generate L new samples from the proposal distribution $p(S_t^{(i)}|S_{t-1}^{(i)}, R_{t-1}^{(i)})$, based on the reward of the old sample. This is accomplished by prompting a generator LLM, denoted as $\mathcal{G}$, to perform self-reflection on why the old sample is imperfect, how to improve the sample quality, and finally generate the improved samples. All the new samples form a new sample set, denoted as $\mathcal{S}'_t$, whose size is L times the size of $\mathcal{S}_{t-1}$.

• **Sample Evaluation.** For each new sample $S_t^{(i)}$ in $\mathcal{S}'_t$, compute its reward $R_t^{(i)}$.

• **Subsampling.** Trim $\mathcal{S}'_t$ down to the size of $\mathcal{S}_{t-1}$ by keeping the samples with the highest reward. Denote the trimmed sample set as $\mathcal{S}_t$.

The iterations terminate when the pre-set maximum number of iterations is reached. With the beam search framework, the algorithm design boils down to designing 1) the sample proposal distribution, and 2) the reward and feedback of each sample. In the following sections, we detail how these design choices are set in tool-use example generation and tool documentation optimization.

5.2.4 Tool-Use Example Generation

In this chapter, we only consider generating examples where only a single tool is used. We will show that (Section 5.4.2) the task LLM can still learn to solve queries that require multiple tools with the examples of single tool use. In this way, examples for different tools can be generated separately. Specifically, examples of using tool F_k take the form of $E = (x, F_k, I_k, y)$, which can be generated via the beam search framework in Section 5.2.3, with the design choices detailed below.

Sample Proposal. The sample proposal is performed by prompting an example generator LLM, denoted as $\mathcal{G}_E$. Since the tool is fixed to F_k, it only needs to propose new samples for the query x, the invocation parameters I_k, and the final answer y. However, the challenge lies in the limited information available about F_k—only an (incomplete) initial documentation $\mathcal{D}_0$ and no user-supplied examples—making it likely that the generated samples fail to invoke the tool correctly.

To address this, we generate samples in **reverse order**: first, we explore valid invocations I_k, observe the tool's output, and then construct a corresponding query x and answer y. This approach effectively "*plays with*" the tool to understand its behavior before defining its use cases.

Legitimate samples of I_k are generated in a rejection-sampling process, where $\mathcal{G}_E$ first generates a tentative sample invocation given $\mathcal{D}_0$, observes the tool outputs and error messages, performs self-reflection, and generates the next one. Note that this inner loop is nested in the outer loop of the beam search framework, so the rejection sampling is also conditional on the tool-use examples generated in the previous outer iteration $t-1$, along with their reward (see Section 5.2.3), except for in outer iteration 0. This inner loop terminates at a fixed number of steps, after which L legitimate invocations are selected.[1]

For each sampled legitimate invocation, we feed the invocation and the tool output to

[1] If the number of legitimate invocations is smaller than L, we will limit the number of proposed samples accordingly.

$\mathcal{G}_E$, which is then prompted to generate a query and an answer by performing N_E steps of self-reflecting refinement, which concludes the sample proposal procedure.

Reward Design. For each generated sample $E^{(i)} = (x^{(i)}, F_k, I_k^{(i)}, y^{(i)})$, the corresponding reward consists of two terms,

$$R^{(i)} = R_q^{(i)} + \lambda R_e^{(i)}. \tag{5.1}$$

$R_q^{(i)}$ evaluates the quality of the generated example, including clarity and coherence between the input query and tool use. This is evaluated by prompting $\mathcal{G}_E$ to output a score of 1-3 given the grading criteria. $R_e^{(i)}$ evaluates the performance of the task LLM $\mathcal{M}_T$ in answering the query in this example:

$$R_e^{(i)} = -\mathcal{P}\{\mathcal{M}_T(x^{(i)}; \mathcal{D}_0, \varnothing); y^{(i)}, F_k, I_k^{(i)}\}, \tag{5.2}$$

where $\mathcal{P}\{\hat{M}; y, F_k, I_k, \}$ denotes the task performance metric (the higher the better, *e.g.*, accuracy) of the model output $\hat{M}$ against the ground-truth answer y and tool invocation F_k, I_k; $\mathcal{M}_T(x^{(i)}; \mathcal{D}_0, \varnothing)$ denotes the output of the task LLM in answering the query $x^{(i)}$ given the initial documentation $\mathcal{D}_0$ and *no in-context examples* (because we don't have any yet at this stage).

The negative sign in Eq. 5.2 indicates that we encourage *difficult examples* – examples that the task LLM cannot get right, because difficult examples bring more surprise to the task LLM and thus are more effective in shaping the LLM's behavior.

5.2.5 Tool Documentation Optimization

The goal of tool documentation optimization is to generate improved tool documentation $\mathcal{D}$ based on the initial one $\mathcal{D}_0$, which is again achieved by the beam search framework, where the documentation sample with the highest reward will be chosen as the final tool documentation.

Sample Proposal. The sample proposal process primarily follows the approach described in Section 5.2.3. We provide the generator LLM $\mathcal{G}_D$, which differs from $\mathcal{G}_E$ used in the

previous step, with the current documentation $\mathcal{D}^{(i)}$ along with tool use errors from $\mathcal{M}_T$. By conditioning the proposal distribution on these errors, we inform $\mathcal{G}_D$ of the documentation's deficiencies or ambiguities, prompting more effective revisions.

Reward Design. The reward is computed on the tool-use example set $\mathcal{E}$ generated in the previous step, essentially treating $\mathcal{E}$ as the validation set for documentation optimization. For each tool documentation sample, $\mathcal{D}^{(i)}$, the corresponding reward is the tool use performance given $\mathcal{D}^{(i)}$ on a small batch in the validation set:

$$R^{(i)} = \mathbb{E}_{(x,F,I,y)\in\mathcal{E}}\big[\mathcal{P}\{\mathcal{M}_T(x;\mathcal{D}^{(i)},\varnothing);y,F,I\}\big]. \tag{5.3}$$

By comparing Eqs. 5.2 and 5.3, it can be observed that tool-use example generation and tool documentation optimization have adversarial objectives, the former seeking to reduce the tool use performance (while maintaining quality and alignment), and the latter to improve it. This resembles the active learning strategy of choosing high-loss examples (Yoo and Kweon, 2019), which reveals that maximizing the performance on the most difficult examples leads to high learning efficiency.

The detailed pseudo-code for both example demonstration optimization and documentation optimization are presented in Appendix C. The meta-prompts are also provided.

5.3 Related Work

5.3.1 LLMs for Tool Use

Recent years have seen notable advances in employing large language models as agents to master tool use for solving complex tasks (Mialon et al., 2023; Qin et al., 2024a), thereby enhancing LLMs' capabilities in multi-modal understanding (Gupta and Kembhavi, 2023; Surís et al., 2023; Wu et al., 2023), programming tools (Gao et al., 2023b; Paranjape et al.,

2023; Team et al., 2023; Zhang et al., 2024; Cai et al., 2024), and other domain-specific functionalities. The conventional approach involves training base models with tool-use data (Thoppilan et al., 2022; Dubey et al., 2024) or fine-tuning LLMs (Parisi et al., 2022; Patil et al., 2023; Schick et al., 2023; Yang et al., 2023; Lin et al., 2025; Liu et al., 2025), but may require continual learning as new tools are added. Hao et al. (2023) addressed scalability by training tool embeddings plug-and-play usage, though still requiring labeled data. Alternatively, LLMs can be augmented with meta-prompts or tool-use instructions at inference time (Lu et al., 2023; Shen et al., 2023b; Song et al., 2023; Qin et al., 2024b; Zhuang et al., 2024). As the range of applications and tools expands, enhancing LLMs' capacity to handle new tools remain pivotal, which we address through PLAY2PROMPT.

5.3.2 Tool-Use Instructions and Optimization

Tool documentation and example demonstrations are key to prompting LLMs for effective tool use. Hsieh et al. (2023) highlighted the risk of hallucinations when documentation is lacking, and Xu et al. (2023) showed performance declines without in-context examples. To automate generating tool-use instances, Shen et al. (2024) leveraged a graph of tool relations for back-instructing queries, relying on the availability of external tool graphs. Yuan et al. (2024) proposed direct prompting to rewrite tool documentation, which relies on labeled documentation examples and lacks systematic optimization and the ability to measure optimization quality. Qu et al. (2025) explored LLMs interacting with tools, focusing on tool documentation only with single-thread iterative refinement. Automatic prompt tuning (Pryzant et al., 2023; Wang et al., 2024b) adapts prompts to domain-specific tasks but require held-out test sets, rendering it unsuitable for zero-shot tool instruction rewriting (Wu et al., 2024). These constraints underscore the need for approaches that optimize tool instructions and demonstrations without labeled data or manual effort, which PLAY2PROMPT achieves by interacting directly with the tool itself.

5.4 Experiments

5.4.1 Data and Setup

Dataset: Berkeley Function-Calling Leaderboard (BFCL). We evaluate on the Berkeley Function-Calling Leaderboard (Yan et al., 2024), a benchmark of real-world data that assesses LLMs' tool-use abilities. Its data includes a non-executable subset evaluated by abstract syntax trees and an executable subset assessed by running the functions. We use the executable subset (referred to as Executable), because actual *tool-play* is central to our approach, reflecting realistic scenarios in which most real-world APIs are callable. This subset has four categories of Python functions (single-tool, multi-tool, parallel tool-calling, and multi-tool parallel tool-calling) and plus one category of REST functions, yielding 310 test queries.

Dataset: StableToolBench. We also evaluate on StableToolBench (Guo et al., 2024), an updated version of ToolBench (Qin et al., 2024b), one of the most widely-used tool-use datasets, which addressed RapidAPIs' instability through a fallback system with caching and API simulation. Our experiments use all six of its testing subsets, including single-tool (I1), multi-tool (I2-same category and I3-different category) queries. Individual APIs, grouped under "tools", correspond to F_k, so I1 test queries usually require calls to multiple APIs within a tool and are not single-tool under our definition. The original dataset's categorization based on tool overlap with training data are thus less relevant for our strictly zero-shot setting. In total, these six subsets cover 790 queries, spanning 2479 unique APIs with an average of 5.3 APIs per query.

Inference and Evaluation. We adhere to official inference settings of each benchmark, where a set of tools is provided for each test query. For task LLM $\mathcal{M}_T$, we tested the 8B and 70B LLaMA models (Dubey et al., 2024), and GPT-3.5 and GPT-4o were used for GPT (Achiam et al., 2023). We also tested on BFCL two state-of-the-art LLMs trained

specifically for tool-calling, namely ToolACE (Liu et al., 2025) and Hammer (Lin et al., 2025), with both models achieving high rankings on the BFCL leaderboard. For BFCL, single-turn prompting with official prompts is used as the baseline inference method for LLaMA models, while direct function-calling mode is used for the GPT models, ToolACE, and Hammer. We do not provide tool-use examples or additional documentation in these baseline runs, complying with a zero-shot setting. For PLAY2PROMPT, optimized in-context examples and documentation are supplied as prompts and runs the baseline inference methods. It supports multi-tool queries by independently optimizing examples and documentation for each tool and then performing inference. We use the official evaluation metric of accuracy, with exact or structural matches depending on categories to determine correctness. We report the average across five categories, and, following the official setup, the weighted average ("Simple" categories weighted by 0.5).

StableToolBench employs a more complex chain-of-thought inference method, ReAct (Yao et al., 2023). We compare against EasyTool (Yuan et al., 2024), which uses direct prompting to optimize tool documentation and generate usage scenarios, but requires documentation and labeled in-context examples from additional tools and is thus not entirely zero-shot[2]. Additionally, we compare against DRAFT (Qu et al., 2025), a concurrent work, which optimizes tool documentation only in an iterative manner [3]. An evaluation LLM is used to judge whether a response adequately answers a user query, due the dataset's free-form output design. We follow the official pipeline, using official ReAct prompts and report solvable pass rate, which measures the percentage of queries deemed solvable by the evaluation LLM.

Additional Implementation Details. For task LLM $\mathcal{M}_T$, `llama-3.1-8b-instruct` is used for LLaMA-8B, `llama-3.3-70b-instruct` for LLaMA-70B, `gpt-3.5-turbo-0125` for GPT-3.5, `gpt-4o-2024-11-20` for GPT-4o, `toolace-2-llama-3.1-8b` for ToolACE-8B, and

[2]We use the optimized tool documentation and usage scenarios provided at https://github.com/microsoft/JARVIS/tree/main/easytool. The inference setting of Yuan et al. (2024) differs from ours as we follow the official inference prompts provided with StableToolBench.

[3]We use the optimized documentation open-sourced at https://github.com/quchangle1/DRAFT. Note that Qu et al. (2025) tested only on the I2-Cat and I3-Inst subsets.

`hammer2.1-7b` for Hammer-7B. On StableToolBench, we use `llama-3.3-70b-instruct` as the evaluation LLM, which produces stable assessments.

For GPT models, we keep the exact same inference setting as used in the original benchmark, except for requiring it to return a single action at each step from the provided toolset by supplying a flag to the OpenAI API. For LLaMA models, on BFCL we follow the exact official inference prompts, and on StableToolBench we adapt the ReAct prompts into LLaMA-3's prompt format but keep everything else fixed as much as possible. For ToolACE and Hammer on BFCL, we also follow the exact official inference prompts.

During inference, for PLAY2PROMPT we set the number of tool-use example demonstrations to 5 per tool for BFCL and 1 per tool for StableToolBench, as StableToolBench averages more tools per query. The sampling temperature is set to 0.001. We report scores averaged over 3 independent runs.

PLAY2PROMPT Optimization Details. PLAY2PROMPT first uses beam search to optimize tool-use examples. We set $\lambda = 1$, $N_E = 3$, limit depth to 3, and explore $L = 3$ beams per node. We use beam width $W = 10$ for BFCL and $W = 3$ for StableToolBench to generate and select the top W examples for each tool, which then are passed to the documentation optimization procedure. Beam search is again applied to select the best tool documentation. We employ `llama-3.1-8b-instruct` as both $\mathcal{G}_E$ and $\mathcal{G}_D$.

5.4.2 Results on BFCL and StableToolbench

In Table 5.1, we summarize the results on BFCL with LLaMA and GPT task models, both with and without tool-use examples and tool documentation produced by PLAY2PROMPT. With PLAY2PROMPT, absolute gains of 4-7% are observed for the open models and GPT-3.5, while GPT-4o achieves 3% increase in average accuracy despite its already high baseline. Notably, PLAY2PROMPT addresses challenging categories such as REST for LLaMA-8B and Hammer-7B, and Multiple-Parallel for all models, yielding improvements of 10-17%. These

Base Model	Method	Simple-Python	Simple-REST	Multiple	Parallel	Multiple Parallel	Weight Avg	Avg
LLaMA-8B	Prompting	96.0	70.0	96.0	90.0	77.5	86.6	85.9
	+Play2Prompt	97.0	87.1	96.0	92.0	92.5	**93.1**	**92.9**
LLaMA-70B	Prompting	100.0	91.4	96.0	84.0	82.5	89.6	90.8
	+Play2Prompt	100.0	91.4	98.0	88.0	95.0	**94.2**	**94.5**
GPT-3.5	Function-calling	97.0	94.3	90.0	86.0	67.5	84.8	87.0
	+Play2Prompt	98.0	95.7	94.0	90.0	85.0	**91.5**	**92.5**
GPT-4o	Function-calling	98.0	98.6	94.0	94.0	77.5	91.0	92.4
	+Play2Prompt	99.0	95.7	98.0	92.0	90.0	**94.3**	**94.9**
ToolACE-8B	Function-calling	96.0	92.9	92.0	86.0	70.0	85.6	87.4
	+Play2Prompt	95.0	90.0	94.0	90.0	82.5	**89.8**	**90.3**
Hammer-7B	Function-calling	96.0	72.9	88.0	86.0	70.0	82.1	82.6
	+Play2Prompt	95.0	82.8	92.0	86.0	80.0	**86.7**	**87.2**

Table 5.1: Results on BFCL Executable. Accuracy scores are shown.

Base Model	Method	I1-Inst	I1-Cat	I1-Tool	I2-Inst	I2-Cat	I3-Inst	Avg
LLaMA-8B	ReAct	50.6	59.3	53.2	58.0	61.3	52.3	55.8
	ReAct+EasyTool	54.7	58.9	57.9	56.1	64.2	48.1	56.7
	ReAct+DRAFT	-	-	-	-	62.6	57.9	-
	ReAct+Play2Prompt	56.6	65.7	60.9	62.5	63.5	63.8	**62.2**
LLaMA-70B	ReAct	58.4	68.3	61.1	63.1	63.2	64.3	63.1
	ReAct+EasyTool	59.0	70.3	63.3	67.6	70.6	63.0	65.6
	ReAct+DRAFT	-	-	-	-	68.5	62.3	-
	ReAct+Play2Prompt	67.5	73.6	64.1	66.5	72.8	70.7	**69.2**
GPT-3.5	ReAct	56.0	64.6	67.4	56.0	63.4	57.4	60.7
	ReAct+EasyTool	57.3	61.4	69.5	61.5	68.3	60.7	63.1
	ReAct+DRAFT	-	-	-	-	68.3	56.1	-
	ReAct+Play2Prompt	61.1	66.6	66.2	67.0	68.3	66.3	**65.9**
GPT-4o	ReAct	54.0	69.7	66.1	59.8	65.3	61.7	62.8
	ReAct+EasyTool	49.9	69.0	65.2	61.3	65.3	52.5	60.5
	ReAct+DRAFT	-	-	-	-	62.1	63.9	-
	ReAct+Play2Prompt	60.7	68.7	71.9	58.8	63.7	65.6	**64.9**

Table 5.2: Results on StableToolBench. Solvable pass rates are shown.

gains suggest that optimized in-context examples and documentation can correct specific shortcomings in tool usage. Moreover, for the more difficult multi-tool queries in Multiple

and Multiple-Parallel, we see larger gains compared to single-tool, even when we optimize each tool independently.

In Table 5.2, we show the solvable pass rates on StableToolBench for LLaMA and GPT models with and without PLAY2PROMPT. Our approach surpasses all baselines with average gains of 5-7% across LLaMA models and GPT-3.5, and outperforms the specifically designed EasyTool, which is not fully zero-shot as it requires in-context demonstrations during optimization. PLAY2PROMPT is also consistent across subsets, avoiding large performance drops in multiple subsets when using EasyTool, highlighting the benefits of real-time "tool-play" and evaluation of candidate examples and documentation during optimization. Moreover, as GPT-4o exhibits more sensitivity to documentation and in-context demonstration changes on this dataset, EasyTool degrades GPT-4o on all subsets, whereas PLAY2PROMPT achieves a 2% improvement on average. Overall, PLAY2PROMPT essentially boosts performance of models up to the baseline performance of the larger models. Compared to DRAFT, which optimizes tool documentation only, PLAY2PROMPT outperforms for all four models on both I2-Cat and I3-Inst. This suggests that optimizing not only documentation, as DRAFT does, but also generating tool-use examples, can synergize and achieve even larger improvements. Additionally, our optimization model of LLaMA-8B is much smaller than the GPT-4o model used in DRAFT, further confirming the effectiveness of PLAY2PROMPT. It is noteworthy that most test samples for all 6 subsets in StableToolBench, as well as the Multiple and Multiple-Parallel subsets in BFCL, are multi-tool queries (see Section 5.4.1), underscoring the effectiveness of PLAY2PROMPT, which operates on single tools independently during optimization.

5.4.3 Robustness of PLAY2PROMPT

To evaluate the robustness of PLAY2PROMPT under incomplete documentation, we simulate noisy tool descriptions on the BFCL Executable dataset by randomly dropping parameter descriptions with increasing probabilities $p \in \{0.0, 0.5, 1.0\}$, while retaining overall tool

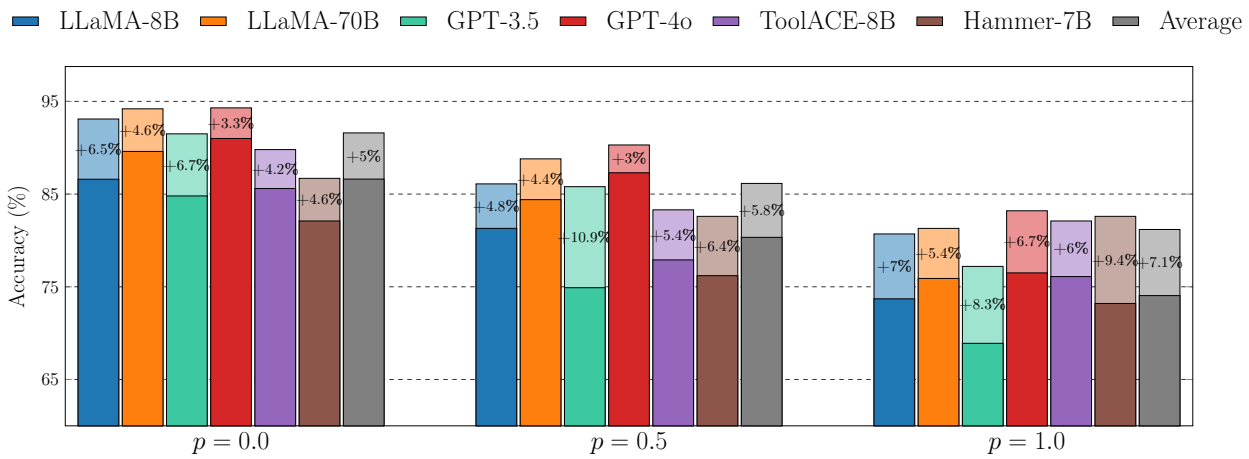

Figure 5.2: Average (weighted) accuracy improvements with PLAY2PROMPT on BFCL, for different parameter description dropout p.

descriptions and parameter names. This setup reflects varying degrees of real-world documentation sparsity. As expected, the degradation of documentation leads to reduced baseline accuracy across all models.

Despite these challenges, PLAY2PROMPT consistently improves tool-use performance across all models and noise levels. As shown in Figure 5.2, PLAY2PROMPT achieves steady accuracy gains, with average improvements growing from 5.0% at $p = 0.0$ and reaching 7.1% at $p = 1.0$. Notably, the benefit of PLAY2PROMPT becomes even more pronounced as documentation quality deteriorates, demonstrating its ability to compensate for missing parameter details and its strong robustness in low-resource settings. Detailed per-model results can be found in Appendix C.

5.4.4 Ablation Studies

Comparing Demonstrations and Documentation. We investigate how much the optimized examples contribute to PLAY2PROMPT's performance gains compared documentation. Using the same optimization procedure, we test two inference settings: one employs new in-context demonstrations alongside original documentation, and the other uses updated documentation without demonstrations. As shown in Table 5.3, employing only optimized

Base Model	Method	BFCL	STB
LLaMA-8B	Baseline	85.9	55.8
	+P2P-Desc	89.9	57.9
	+P2P-Demo	90.8	59.5
	+P2P	**92.9**	**62.2**
LLaMA-70B	Baseline	90.8	63.1
	+P2P-Desc	93.6	64.4
	+P2P-Demo	92.7	67.8
	+P2P	**94.5**	**69.2**
GPT-3.5	Baseline	87.0	60.7
	+P2P-Desc	89.0	63.5
	+P2P-Demo	91.9	65.3
	+P2P	**92.5**	**65.9**

Table 5.3: Ablation on PLAY2PROMPT (P2P) using generated example demonstrations only (P2P-Demo) and generated descriptions only (P2P-Desc). Scores indicate average accuracy for BFCL and average solvable pass rate for StableToolBench (STB).

Base Model	Method	STB
LLaMA-8B	ReAct	55.8
	+PLAY2PROMPT-Easy	60.7
	+PLAY2PROMPT	**62.2**
LLaMA-70B	ReAct	63.1
	+PLAY2PROMPT-Easy	68.2
	+PLAY2PROMPT	**69.2**

Table 5.4: Ablation on demonstration difficulty for PLAY2PROMPT. We report average solvable pass rates.

demonstrations yields larger improvements than solely relying on optimized documentation, and combining both consistently achieves the best results. These findings suggest that demonstrations and documentation complement each other in guiding LLMs' tool use, confirming the effectiveness of our two-stage approach.

Tool-use Example Difficulty. We further examine how challenging examples influence performance by removing the R_e term during example generation and only using R_q. Without R_e, the model no longer targets difficult examples, yielding what we denote as P2P-Easy. Under the same inference configuration, we substitute these examples in place of the original

Search strategy	Pass Rate	# Proposals Explored
ReAct	64.3	-
+P2P-MC (depth= 1)	65.2	3
+P2P-MC (depth= 5)	66.7	15
+P2P-MC (depth= 5) & N_E= 3	68.9	45
+P2P-Beam (depth= 5) & N_E= 3	**70.7**	135

Table 5.5: Ablation on search strategies, on the I3-Inst subset of StableToolBench. LLaMA-70B is used for task model $\mathcal{M}_T$. MC denotes Monte Carlo search.

demonstrations and report results for LLaMA on StableToolBench in Table 5.4. Ignoring difficult examples reduces final performance by an average of 1-2% across all subsets, confirming that generating in-context examples with higher difficulty enhance tool usage.

Search Strategy. We investigate alternative strategies for exploring the example-documentation space by comparing beam search in PLAY2PROMPT to Monte Carlo search (MC) at various depths, sharing the same sample proposal scheme but with different number of proposals and sub-sampling approach. We also study the role of N_E self-reflection steps during example generation. Results are shown in Table 5.5, focusing on I3-Inst, which combines multiple tools from different categories and is thus more challenging. The results show that MC underperforms beam search due to its limited exploration and is more subject to randomness, while deepening the search and adding self-reflection actions significantly improves performance, highlighting the importance of systematic search in optimizing examples and documentation. Given the tradeoff between performance and efficiency, a suitable search method can be chosen when using PLAY2PROMPT based on the user's computational budget.

5.4.5 Qualitative Analysis

Tool-use Examples. Here, we show two examples of how PLAY2PROMPT generates tool-use examples that aids both documentation refinement and downstream LLM inference. In Figure 5.3, we generate tool-use examples for a derivative estimation python function zero-shot

Tool Signature	`estimate_derivative(function: str, x: int)`
Documentation	Estimate the derivative of a function at a given point.

State	Trial	Tool Call	Error Feedback	Generated Question	Test with $\mathcal{M}_T$ (Difficulty)
s_1	1	`estimate_derivative( function= "lambda x: x**3 + 2*x", x=2)`	*Errors*: None	Given a function $f(x) = x^3 + 2x$ and a point $x = 2$, estimate the rate of change of this function at $x = 2$ using the correct computational tools and APIs.	Pass (easy ✗)
s_2	1	`estimate_derivative( function= "lambda x: sin(x)", x=3.14)`	*Errors*: name 'sin' is not defined	–	–
s_2	2	`estimate_derivative( function= "lambda x: math.sin(x)", x=0.5)`	*Errors*: None	Estimate the derivative of the sine function at $x = 0.5$ using the math library.	Fail (hard ✓)

Figure 5.3: A tool-use example generation search trajectory that shows PLAY2PROMPT in action, with the more difficult and informative example pair being ranked highest. This example is chosen from BFCL's test set.

from scatch. In the first search state, the first generated tool call passed the tool-play stage with no errors, and the corresponding question is generated. However, when tested with the task model $\mathcal{M}_T$, the model passed successfully, suggesting that the generated example pair is too trivial and uninformative. The search then reaches the second state, where an initial generated tool call resulted in error and needed a second attempt, which PLAY2PROMPT corrects through feedback, and subsequently generates a corresponding question. In this case, $\mathcal{M}_T$ fails to correctly call the tool, and we deem this (tool-call, question) pair to be representative, ending the current search. Note that in practice for each tool, we generate multiple example pairs. The generated questions are also scored on quality R_e through prompting; in this case, the question shown in the example was the top-scoring one.

In the second example, shown in Figure 5.4, a much more complex API weather endpoint tool is optimized with PLAY2PROMPT. It requires sending a `GET` request to a URL with specific payload format and arguments. We see from the example that the optimization

Tool Signature `requests.get(url: str, params: {'latitude', 'longitude', ...}, ...)`

Documentation Sends a GET request to the specified URL. The API endpoint for fetching weather data from the Open-Meteo API for the given latitude and longitude.

State	Trial	Tool Call	Error Feedback	Generated Question	Test with $\mathcal{M}_T$ (Difficulty)
s_1	1	`requests.get( latitude="37.7749", longitude= "-122.4194", hourly= "temperature_2m, relative_humidity_2m, dew_point_2m, cloud_cover, wind_speed_10m", temperature_unit= "celsius", wind_speed_unit= "kmh",...,api_key= "<api_key>")`	*Errors*: missing 1 required positional argument: 'url'	–	–
s_1	2	`requests.get( url="https://api.open-meteo.com/v1/forecast?", latitude="40.7128", longitude= "-74.0060", hourly= "temperature_2m, relative_humidity_2m, dew_point_2m, cloud_cover, wind_speed_10m", temperature_unit= "celsius", wind_speed_unit= "kmh",...,api_key= "<api_key>")`	*Errors*: got an unexpected keyword argument 'latitude'	–	–
s_1	3	`requests.get( url="https://api.open-meteo.com/v1/forecast?", params={"latitude": "37.7749", "longitude": "-122.4194", "hourly": "temperature_2m, relative_humidity_2m, dew_point_2m, cloud_cover, wind_speed_10m", "temperature_unit": "celsius", "wind_speed_unit": "kmh",...,"api_key": "<api_key>"})`	*Errors*: The supplied API key is invalid.	–	–
s_1	4	`requests.get(url="https://api.open-meteo.com/v1/forecast?", params={"latitude": "40.7128", "longitude": "-74.0060", "hourly": "temperature_2m, relative_humidity_2m, dew_point_2m, cloud_cover, wind_speed_10m", "temperature_unit": "celsius", "wind_speed_unit": "kmh",...}, headers= {"X-RapidAPI-Key": "<api_key>"})`	*Errors*: None	I need to get the current and future weather forecast for New York City, and I want to know the hourly temperature, relative humidity, dew point, cloud cover, and wind speed for the next 10 days, starting from now, with data points every hour for 12 hours, all in Celsius and km/h. Please use the API associated with the X-RapidAPI-Key `<api_key>` which I have.	Fail (hard ✓)

Figure 5.4: Another example of PLAY2PROMPT's tool-use example generation with a complex RESTful weather tool, taken from BFCL's test set.

Tool Signature `calendar_events_for_yh_finance_complete(end_date: str, start_date: str, symbol: str, period: str)`

Validation question I'm planning to invest in GOOGL stock for a yearly perspective. Can you provide me with the calendar events, such as earning announcements, ex-dividend dates, and dividend payments, from September 1st, 2024, to September 20th, 2024?

State	Documentation	Error Feedback	Test with $\mathcal{M}_T$ on question
s_0	(*Original*) Calendar Events of a particular stock.	*Errors*: "start_date" not a valid parameter	*Solve Rate*: 33%
s_1	This function retrieves calendar events of a particular stock, including earnings announcements, ex-dividend dates, and dividend payments, between a specified start and end date. It requires the stock symbol (a string), start date (yyyy-mm-dd), end date (yyyy-mm-dd), and period(daily, yearly, etc.).	*Errors*: "start_date" not a valid parameter	*Solve Rate*: 66%
s_2	This function retrieves calendar events of a particular stock, including earnings announcements, ex-dividend dates, and dividend payments, between a specified start date ('yyyy-mm-dd') and end date ('yyyy-mm-dd'). It requires the stock symbol (a string), start date ('yyyy-mm-dd'), end date ('yyyy-mm-dd'), and an optional period (daily, yearly, etc.). Clearly specify these required parameters to ensure accurate function calls.	*Errors*: "start_date" not a valid parameter	*Solve Rate*: 66%
s_3	This function retrieves calendar events of a particular stock, including earnings announcements, ex-dividend dates, and dividend payments, between a specified 'from' and 'to' date. It requires the stock symbol (a string), the 'from' date in the format 'yyyy-mm-dd', the 'to' date in the format 'yyyy-mm-dd', and an optional period (daily, yearly, etc.). Clearly specify these required parameters to ensure accurate function calls. This function is designed for YH Finance Complete tool.	*Errors*: None	*Solve Rate*: 100%

Figure 5.5: An example of PLAY2PROMPT facing incorrect documentation. We show the beam search trajectory with the highest solve rate on the validation set. At each step, a new documentation is explored based on error feedback. Example chosen from StableToolBench.

goes through several trial-and-error attempts, each time refining the tool call from error feedback and eventually learns to output a functional tool call. The resulting example pair is informative (w.r.t. $\mathcal{M}_T$) and the search ends.

Tool Documentations. Additionally, we analyze two examples of how PLAY2PROMPT leverages tool play errors to refine tool documentation. In Figure 5.5, we show an optimal

Tool Signature	`constructor_standings_for_formula_1_standings()`		
Validation question	Find the current season constructor standings for the 2023 Formula 1 season, including the team names, their positions, and the points they have accumulated so far.		
State	**Documentation**	**Error Feedback**	**Test with $\mathcal{M}_T$ on question**
s_0	(*Original*) Get the current season constructor standings.	*Errors*: "season" not a valid parameter	*Solve Rate*: 33%
s_1	Retrieve the current constructor standings in Formula 1.	*Errors*: "season" not a valid parameter	*Solve Rate*: 66%
s_2	Retrieve the current constructor standings in Formula 1 without specifying a season, using the function `constructor_standings_for_formula_1_standings`	*Errors*: None	*Solve Rate*: 100%

Figure 5.6: A typical example of PLAY2PROMPT assisting LLMs in correcting errors in generating parameter values with optimized documentation. Also from StableToolBench.

search path for optimizing the finance API tool, with one of the questions from the validation set shown. In this example, the documentation is outdated, specifying `start_date` and `end_date` instead of `from` and `to`, and wrongly labeling them as required. Initially, PLAY2PROMPT is confused by contradictory information but gradually incorporates more precise details, solves queries without needing those parameters, and eventually after 4 search steps it identifies the correct parameter names, leading to better performance..

We show another example in Figure 5.6 to illustrate how PLAY2PROMPT commonly aids LLMs' tool usage. The task model $\mathcal{M}_T$ (LLaMA-8B) gets confused by the query specifying the year, which PLAY2PROMPT first attempts to remove "year" from the description, and further explicitly prompts $\mathcal{M}_T$ to not use the parameter, which finally results in the task model answering the question correctly and consistently.

5.4.6 Human Evaluation of Generation Quality

Since our primary objective is to improve LLMs' tool-use capabilities, our main evaluation focuses on tool-use accuracy, which directly measures the effectiveness of our approach. While the quality of documentation and examples may not directly reflect tool-use improvements, they can provide complementary insights—particularly in enhancing clarity for human users

Dataset	Completeness			Conciseness			Accuracy		
STB	P2P	Raw	Equal	P2P	Raw	Equal	P2P	Raw	Equal
	85%	2%	13%	35%	**47%**	18%	**43%**	19%	37%
BFCL	P2P	Raw	Equal	P2P	Raw	Equal	P2P	Raw	Equal
	79%	5%	17%	15%	**70%**	15%	18%	19%	**63%**

Table 5.6: Human evaluation on tool documentation quality, comparing completeness, conciseness, and accuracy of PLAY2PROMPT versus raw (initial) documentation.

Examples	Naturalness	Alignment
BFCL Test Set	4.39	4.68
PLAY2PROMPT	4.17	4.52

Table 5.7: Human evaluation on tool usage example (question-function call) quality, rated on a scale of 1 (low) to 5 (high).

and serving as secondary evidence of optimization quality. To assess these aspects, we conducted human evaluations on 50 randomly selected documentation outputs and 30 tool-use examples using six annotators. For documentation, we follow Qu et al. (2025) and assess completeness, conciseness, and accuracy relative to the original dataset-provided descriptions. For tool-use examples, we adopt the evaluation criteria from Shen et al. (2024), scoring the naturalness of the query and its alignment with the function call[4], each on a scale of 1 to 5. As shown in Table 5.6, our optimized documentation achieves significantly higher completeness and comparable or better accuracy, though with reduced conciseness due to increased verbosity. As shown in Table 5.7, the zero-shot-generated examples are rated similarly to manually curated ones in terms of both naturalness and alignment, demonstrating their practical utility.

[4]We omit the complexity metric used in Shen et al. (2024), as PLAY2PROMPT generates single-tool usage examples.

5.5 Chapter Summary

In this chapter, we propose PLAY2PROMPT, an automated framework that iteratively refines tool documentation and generates usage examples, enabling more effective zero-shot tool utilization without any labeled data. PLAY2PROMPT integrates a search-based trial-and-error process augmented with self-reflection, allowing LLMs to explore and "play" with tools to refine both tool documentation and demonstrations for enhanced performance. Tool-use examples are first generated with this process, with more difficult examples prioritized so they are more informative. The examples are then used as validation when optimizing tool documentation in the second stage. PLAY2PROMPT is entirely zero-shot, scalable, and task-agnostic, making it broadly applicable across diverse tools and domains, and practical for large-scale enhancement of LLM tool use without additional manual effort.

Chapter 6

Conclusion and Future Work

6.1 Conclusions

This thesis studies context construction for agentic LLM frameworks, where an LLM responds to user requests by grounding its reasoning on external resources such as retrieved information and tool outputs in order to generalize outside of its parametric knowledge. While modern LLMs exhibit strong instruction-following and tool-calling capabilities, agentic systems are limited by the need to select and present the correct and relevant context under compute and context-window constraints. These challenges are amplified as knowledge sources and tool libraries scale from small, curated collections to large, diverse, and continuously-changing libraries. Motivated by these scalability and generalizability issues, this thesis focuses on three core problems of agentic context construction: retrieving query-specific textual information from large corpora, retrieving relevant tools from large tool repositories for compositional and abstract user intents, and constructing effective tool instructions for newly introduced tools that lack high-quality documentation or example demonstrations.

Chapter 3 addresses information retrieval by proposing Joint Passage Re-ranking (JPR), an approach that improves re-ranking effectiveness by directly maximizing the pointwise mutual information between queries and passages during inference. Instead of ranking passages solely

by a discriminative cross-encoder score, JPR incorporates a term that penalizes passages with high unconditional probability, thereby reducing the tendency to retrieve generic, broadly relevant passages that are less informative for the given query. This yields a joint re-ranking method that combines a discriminative cross-encoder and a conditional generation, leveraging complementary signals from both paradigms. Empirically, JPR improves open-domain QA retrieval and also generalizes effectively in zero-shot retrieval across diverse tasks, consistently outperforming conventional cross-encoders and generative scorers and mitigating their failure cases.

Chapter 4 shifts the focus to tool retrieval, where the objective is to select a small, relevant subset of tools from a large tool library so that a downstream LLM agent can use them to answer user requests better. We propose tool query planning (TOOLQP), which formulates tool retrieval as an iterative planning problem instead of a single-shot retrieval task. TOOLQP decomposes a complex user request into a sequence of sub-goals, interactively generates targeted retrieval queries for each sub-goal, and uses retrieval feedback to self-correct and uncover tool dependencies before aggregating results into a final ranking. To learn this planner, we develop a training pipeline based on sythesizing multi-step trajectories for supervised fine-tuning followed by further refinement by reinforcement learning optimization directly with retrieval outcomes. Experiments show strong in-domain retrieval gains and substantial improvements in zero-shot transfer settings, while remaining robust when deployed with unseen retrievers, indicating a query planning policy that is generalizable. This translates to downstream end-to-end tool-use performance, significantly increasing tool-use success rates.

Chapter 5 studies prompt construction for tool use in real-world settings where newly introduced tools are accompanied by incomplete or low-quality documentation with no curated in-context demonstrations available. We introduce PLAY2PROMPT, a fully zero-shot and task-agnostic prompt optimization framework that improves tool use by generating both tool-use examples and refined tool documentation through tool interaction. PLAY2PROMPT utilizes

tool-play within a search-based process, by executing candidate tool calls and using success and error signals to guide the refinement steps to synthesize tool-use examples. Using the generated examples as validation, tool documentation is then optimized with a similar search framework that incorporates tool-use errors for better specification or constraints, usage, and failure cases. Evaluation show that this interaction-driven optimization consistently improves tool-use performance across models, with especially strong robustness when initial documentation is poor.

Together, the thesis supports the view that scalability and generalizability in agentic LLMs depend not only on the strengths of the underlying model but also significantly on how the context is constructed. Across text retrieval, tool retrieval, and tool instruction construction, the unifying theme is to replace fragile single-shot augmentation with objectives and procedures that better reflect the structure of the data and agentic setup. Beyond direct empirical improvements, these results suggest practical design principles for future agentic systems, including modular context-construction layers that can be built independently yet incorporated easily with downstream LLM agents, and motivate future research directions that are critical for deploying agentic LLMs reliably at scale.

6.2 Future Work

Based on the ideas and results presented in this thesis, the following lists several research directions that can be pursued in the future:

Mutual Information Maximization for Retrieval. In Chapter 3, we tackled passage re-ranking for retrieval, focusing on the second stage re-ranking scores using dense cross-encoders and generative models. We have not explored approaching the retrieval process without passage re-ranking, that is, directly applying the PMI objective to train a dense retrieval model, which could potentially lead to larger improvements but comes with much higher computational costs. Furthermore, this idea could be explored for other retrieval

tasks such as document and tool retrieval, as retrieval models may be biased by the *general* documents or tools that appear too frequently in training.

Multi-step Query Planning for Tool Retrieval. In Chapter 4, we designed a query planning method, ToolQP, that retrieves tools in multiple steps; however our synthetic data generation utilized a single base retriever due to compute constraints, implicitly biasing the planner to a specific embedding space; future work could explore multi-retriever synthesis to foster a more robust, agnostic policy. 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. Finally, developing an adaptive planning mechanism that dynamically bypasses or simplifies decomposition for simple queries would further optimize the latency-performance trade-off.

Tool Instruction Tuning. In Chapter 5, we described a prompt optimization method, Play2Prompt, for tool documentation and in-context examples. Play2Prompt optimizes a single tool, but it's still applicable to queries that require multiple tools by optimizing each tool independently, and then using the optimized examples and documentation from multiple tools together at inference. Although results (on Multiple and Multiple-Parallel in BFCL and the entirety of StableToolBench) on multi-tool queries show sizeable performance gains, scaling the optimization process itself from single-tool scenarios to multiple tools can likely enhance LLM's tool use effectiveness even more. Next, for example demonstrations, we use rejection sampling to generate tool invocations first, which do not work for functions whose parameter space is too large, for instance parameters that take long ID string inputs or authentication tokens that require calls to other tools beforehand. Exploring multi-tool dependencies could potentially resolve this issue and improve tool play. Furthermore, in the study we relied on prompt optimization via LLM feedback, resulting in reliance and potentially bias from the LLM optimizer we chose. Training a specialized model via RLVR could potentially be more effective and efficient. Finally, in our study we focus on tool

documentation and demonstrations only, relegating other information as meta-information, which could be potential next steps to explore.

Appendix A

Additional Details for Chapter 3 (JPR) Experiments

A.1 Dataset Details and Licenses

Natural Questions and TriviaQA. We show the number of train/dev/test examples in NQ and TriviaQA in Table A.1. Please refer to Kwiatkowski et al. (2019) and Joshi et al. (2017) for more details. Note that NQ is licensed under Apache License 2.0, which we follow, and TriviaQA does not provide a dataset license. We also show one testing example from each dataset in Figure A.1.

Dataset	Train	Dev	Test
Natural Questions	58,880	8,757	3,610
TriviaQA	60,413	8,837	11,313

Table A.1: Dataset splits for NQ and TriviaQA.

The BEIR Benchmark. We evaluate baselines and our approach on the publicly available datasets in BEIR: TREC-COVID (Voorhees et al., 2021), NFCorpus (Boteva et al., 2016), NQ (Kwiatkowski et al., 2019), HotpotQA (Yang et al., 2018), FiQA-2018 (Maia et al.,

Dataset	Query	Relevant Passage(s)
NQ	When was "as you like it" first performed?	*As You Like It* As You Like It is a pastoral comedy by William Shakespeare believed to have been written in 1599 and first published in the First Folio in 1623. The play's first performance is uncertain, though a performance at Wilton House in 1603 has been suggested as a possibility. "As You Like It" follows its heroine Rosalind as she flees persecution in her uncle's court, accompanied by her cousin Celia to find safety and, eventually, love, in the Forest of Arden. In the forest, they encounter a variety of memorable characters, notably the melancholy traveller Jaques who speaks
TriviaQA	In what outdoor sport, sanctioned by the NHPA, do you score 3 points for a ringer, 2 for a leaner, and the closest scores a point?	second player throws both of their horseshoes–again, one at a time–at their end. After scoring, the next round is done in reverse order, or by throwing back at the original stake. Play continues until one player has at least 15 points at the end of a round. NHPA sanctioned games are generally played to 40 points, or a shoe limit of 40 or 50 shoes. The horseshoes can be made of either plastic or metal. In horseshoes, there are two ways to score: by throwing "ringers" or by throwing the horseshoe nearest to the stake. A ringer is a horseshoe

Figure A.1: Dataset examples from Natural Questions (NQ) and TriviaQA.

2018), ArguAna (Wachsmuth et al., 2018), Touché-2020 (Bondarenko et al., 2020), CQADupStack (Hoogeveen et al., 2015), Quora[1], DBPedia (Hasibi et al., 2017), SCIDOCS (Cohan et al., 2020), FEVER (Thorne et al., 2018), Climate-FEVER (Diggelmann et al., 2020), and SciFact (Wadden et al., 2020). For details on dataset statistics, links, and licenses please refer to BEIR (Thakur et al., 2021). Note that datasets in BEIR that are under copyright were not used in this study, and 4 out of the 14 publicly available datasets do not report dataset licenses. We follow the intended uses for each dataset license.

[1] https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

Appendix B

Additional Details for Chapter 4 (ToolQP) Experiments

B.1 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)[1]. 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. We also show one testing example from each dataset in Figures B.1 and B.2.

B.2 Additional Baseline Configurations on ToolRet

In additional to the main experimental results shown in Chapter 4, 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

[1]See Table 6 (Appendix) in Shi et al. (2025) for the complete list of component datasets.

Dataset	Query	Relevant Passage(s)
ToolRet (Web*)	I heard about a new tool called 'AutoAGI' that can help me automate some tasks. Can you install it for me? I could not wait to try it out!	[{"id": "toolemu_tool_0", "doc": {"toolkit": "Terminal", "category": "Programming, Development, and Scientific Tools", "tools": [{"name": "Execute", "summary": "Execute a terminal command and return the output. This command should follow proper syntax and be supported by the terminal environment.", "parameters": [{"name": "command", "type": "string", "description": "The command to execute in the terminal.", "required": true}], "returns": [{"name": "output", "type": "string", "description": "The output generated by the executed terminal command, including both standard output and standard error streams."}, {"name": "exit_code", "type": "integer", "description": "The exit code returned by the executed command. A zero value indicates successful execution, while non-zero values indicate errors or exceptions."}], "exceptions": [{"name": "InvalidRequestException", "description": "The 'command' parameter contains an invalid or malformed command, which results in a failed execution attempt."}]}], "name": "Terminal", "description": "Executes commands in a terminal on the user's local system. Use it to run valid terminal commands for tasks such as file management, system control, and more"}, "relevance": 1}, {"id": "toolemu_tool_15", ...}...
ToolRet (Code)	Our company receives invoices in different formats. We need to extract specific information from these documents to process payments and keep records.	[{"doc": {"domain": "Multimodal Document Question Answer", "framework": "Hugging Face Transformers", "functionality": "Document Question Answering", "api_call": "AutoModelForDocumentQuestionAnswering.from_pretrained('tiennvcs/layoutlmv2-base-uncased-finetuned-docvqa')", "api_arguments": [], "python_environment_requirements": ["transformers==4.12.2", "torch==1.8.0+cu101", "datasets==1.14.0", "tokenizers==0.10.3"], "example_code": "", "performance": {"dataset": "unknown", "accuracy": {"Loss": 1.194}}, "description": "This model is a fine-tuned version of microsoft/layoutlmv2-base-uncased on an unknown dataset.", "id": "huggingface_api_115", "name": "layoutlmv2-base-uncased-finetuned-docvqa"}}]
ToolRet (Custom)	I have a video 'example.mp4' and a voiceover 'example.wav'. Can you synchronize the voiceover with the visuals of the video?	[{"id": "taskbench_multimedia_tool_38", "doc": {"input-type": ["video", "audio"], "output-type": ["video"], "name": "Video Synchronization", "description": "Synchronizes the timing of an existing voiceover or audio file with the visuals of a given video."}, "relevance": 1}]

Figure B.1: Dataset examples from each of the three ToolRet categories.

the full ToolRet training set, with 200k instances. The results are shown in Table B.1.

Comparing re-ranking top-20 against top-100, we observe that while in-domain retrieval

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 B.1: 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.

performance are improved with the additional 5x compute, for zero-shot generalization it instead harmed performance, and significantly outperforms 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 that ToolQP-Format, and also much more compute-heavy with contrastive losses instread of standard negative likelihood training. For generalizability, it still underperforms ToolQP trained on much less data, further showcasing the effectiveness and efficiency of ToolQP.

B.3 Detailed Results for the Retrieval Transfer Experiments

We provide in Table B.2 detailed results for each model-method pairing for each category in ToolRet.

B.4 Prompts for Baseline Methods

We include the prompts for baselines and TOOLQP described in Chapter 4 below in Figures B.3-B.11.

Dataset	Role	Messages
API-Bank	User	Can you help me delete my account?
	Assistant	Sure, I can definitely help with that. Before proceeding with the deletion, can you please provide me your authentication details?
	User	username - user1, password - user1pass
	Assistant	For authentication, I need to get your user token. Please provide me with your email address and current password.
	API	`{“api_name”: “GetUserToken”, “param_dict”: {“username”: “user1”, “password”: “user1pass”}, “result”: {“api_name”: “GetUserToken”, “input”: {“username”: “user1”, “password”: “user1pass”}, “output”: {“token”: “n9m8k7j6h5g4f3d2s1a0”}, “exception”: null}}`
	Assistant	Great! Now that I have your user token, I can proceed with deleting your account. Are you sure you want to proceed with the deletion?
	User	Yes, I am sure.
	API	`{“api_name”: “DeleteAccount”, “param_dict”: {“token”: “n9m8k7j6h5g4f3d2s1a0”}, “result”: {“input”: {“token”: “n9m8k7j6h5g4f3d2s1a0”}, “output”: {“status”: “success”}, “exception”: null}}`
	Assistant	Your account has been successfully deleted. Is there anything else I can help you with?
StableToolBench	System	You are AutoGPT, you can use many tools(functions) to do the following task. First I will give you the task description, and your task start. At each step, you need to give your thought to analyze the status now and what to do next, with a function call to actually execute your step. After the call, you will get the call result, and you are now in a new state. Then you will analyze your status now, then decide what to do next... After many (Thought-call) pairs, you finally perform the task, then you can give your finial answer. Remember: 1. the state change is irreversible, you can’t go back to one of the former state, if you want to restart the task, say “I give up and restart”. 2. All the thought is short, at most in 5 sentence. 3. You can do more then one tries, so if your plan is to continuosly try some conditions, you can do one of the conditions per try. Let’s Begin! Task description: You should use functions to help handle the real time user queries. Remember: 1. ALWAYS call “Finish” function at the end of the task. And the final answer should contain enough information to show to the user,If you can’t handle the task, or you find that function calls always fail(the function is not valid now), use function Finish->`give_up_and_restart`. 2. Do not use origin tool names, use only subfunctions’ names. You have access of the following tools: {`retrieved_tools`}
	User	I’m a phone enthusiast and I want to compare the specifications and prices of different phone models. Can you provide me with a list of phone models along with their specifications and prices? Additionally, I would like to know if there are any security vulnerabilities associated with the firmware of these phones.

Figure B.2: Dataset examples from API-Bank and StableToolBench.

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 B.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. Web* denotes the zero-shot Web datasets (excluding the in-domain training sources).

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 B.3: 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 B.4: 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 B.5: 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 B.6: 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 B.7: 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 B.8: 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 B.9: 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>').

Figure B.10: Prompts for data generation for ToolQP training.

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 B.11: Prompts for data generation for ToolQP training (continued).

Appendix C

Additional Details for Chapter 5 (PLAY2PROMPT) Experiments

C.1 Dataset Licenses

BFCL (Yan et al., 2024) and StableToolBench(Guo et al., 2024) are both licensed under Apache License 2.0. We adhere to intended uses in the license. We also show testing examples from BFCL in Figure C.1.

C.2 Detailed Results for the Robustness Experiments

To assess the robustness of PLAY2PROMPT, we artificially introduce noise to tool documentation in BFCL Executable by dropping each parameter description with a probability p, leaving general tool descriptions and signatures intact. As shown in Table C.1, with $p = 0.5$, this degradation reduces baseline performance by about 5% for the LLaMA models and GPT-4o, and by 10% for GPT-3.5, especially for the most challenging Multiple-Parallel category (c.f. Table 5.1). With PLAY2PROMPT, performance consistently exceeds baselines across all models, improving by 3-4% for the larger LLaMA-70B and GPT-4o models, 5% for LLaMA-8B, and most significantly, recovering the full 10% loss for GPT-3.5.

Base Model	Method	Simple-Python	Simple-REST	Multiple	Parallel	Multiple Parallel	Weight Avg	Avg
LLaMA-8B	Prompting	91.0	67.1	92.0	84.0	70.0	81.3	80.8
	+Play2Prompt	94.0	81.4	92.0	82.0	82.5	**86.1**	**86.4**
LLaMA-70B	Prompting	96.0	90.0	92.0	80.0	72.5	84.4	86.1
	+Play2Prompt	95.0	88.6	92.0	84.0	87.5	**88.8**	**89.4**
GPT-3.5	Function-calling	92.0	87.1	84.0	76.0	50.0	74.9	77.8
	+Play2Prompt	98.0	88.6	86.0	84.0	80.0	**85.8**	**87.3**
GPT-4o	Function-calling	93.0	95.7	92.0	88.0	75.0	87.3	88.7
	+Play2Prompt	98.0	94.3	94.0	88.0	82.5	**90.3**	**91.6**
ToolACE-8B	Function-calling	92.0	84.3	86.0	80.0	57.5	77.9	80.0
	+Play2Prompt	93.0	84.3	90.0	82.0	72.5	**83.3**	**84.4**
Hammer-7B	Function-calling	93.0	71.4	78.0	82.0	62.5	76.2	77.4
	+Play2Prompt	92.0	82.9	86.0	82.0	75.0	**82.6**	**83.6**

Table C.1: Results on BFCL Executable, with parameter description dropout $p = 0.5$. Accuracy scores are shown.

In Table C.2, we further include experiments on BFCL with an even higher parameter description dropout with $p = 1.0$, that is, dropping out all parameter descriptions. The documentation still contains information from the overall tool description and parameter name in this scenario. These results illustrate Play2Prompt's robustness to incomplete documentation, and especially shines when initial documentation is poor.

C.3 Pseudo-code and Meta-prompts for Play2Prompt

The full procedures for both example demonstration optimization and documentation optimization are presented in Algorithm 2 and 3, respectively. Please refer to Section 5.2.1 for notations. In the algorithms, m_i represents meta-prompts, which are included below.

Base Model	Method	Simple-Python	Simple-REST	Multiple	Parallel	Multiple Parallel	Weight Avg	Avg
LLaMA-8B	Prompting	85.0	62.9	84.0	82.0	55.0	73.7	73.8
	+Play2Prompt	88.0	78.6	90.0	82.0	67.5	**80.7**	**81.2**
LLaMA-70B	Prompting	86.0	87.1	86.0	76.0	55.0	75.9	78.0
	+Play2Prompt	89.0	87.1	92.0	80.0	65.0	**81.3**	**82.6**
GPT-3.5	Function-calling	84.0	87.1	78.0	72.0	40.0	68.9	72.2
	+Play2Prompt	92.0	84.3	82.0	86.0	52.5	**77.2**	**79.4**
GPT-4o	Function-calling	85.0	95.7	84.0	84.0	47.5	76.5	79.2
	+Play2Prompt	91.0	94.3	92.0	88.0	60.0	**83.2**	**85.0**
ToolACE-8B	Function-calling	88.0	80.0	86.0	82.0	52.5	76.1	77.7
	+Play2Prompt	89.0	85.7	90.0	86.0	65.0	**82.1**	**83.1**
Hammer-7B	Function-calling	85.0	75.7	76.0	84.0	52.5	73.2	74.6
	+Play2Prompt	90.0	82.9	88.0	86.0	70.0	**82.6**	**83.4**

Table C.2: Results on BFCL Executable, with parameter description dropout $p = 1.0$. Accuracy scores are shown.

Algorithm 2 ExampleOptimizationStep

Input: $\mathcal{S}_t = E_t = (x_t, F, I_t, y_t)$: tool-use example, $\mathcal{D}_0$: documentation, a_t: reflection

Output: $\mathcal{S}_{t+1} = E_{t+1} = (x_{t+1}, F, I_{t+1}, y_{t+1})$: updated example, r_{t+1}: score, a_{t+1}: reflection

$c \leftarrow$ false
while $\neg c$ **do** ▷ Rejection Sampling of tool invocation i
  $I_{t+1} \sim p_{\mathcal{G}_E}(I|I_t, c, \mathcal{D}_0, a_t, m_1)$ ▷ Sample candidate tool invocation, incorporating reflection a_t
  $o_{t+1} \leftarrow F(I_{t+1})$ ▷ Execute tool function
  $c \sim p_{\mathcal{G}_E}(c|I_{t+1}, o_{t+1}, \mathcal{D}_0, m_2)$ ▷ Verify validity of tool call
end while
for $n \leftarrow 1$ to N_E **do** ▷ Rollout for with self-reflection policy
  $x_{t+1} \sim p_{\mathcal{G}_E}(x|I_{t+1}, o_{t+1}, \mathcal{D}_0, m_3)$ ▷ Sample user query x
  $y_{t+1} \sim p_{\mathcal{G}_E}(y|I_{t+1}, o_{t+1}, x_{t+1}, \mathcal{D}_0, m_4)$ ▷ Sample corresponding answer y
  $\mathcal{R}_q \sim p_{\mathcal{G}_E}(r|y_{t+1}, I_{t+1}, o_{t+1}, x_{t+1}, \mathcal{D}_0, m_5)$ ▷ Evaluate example quality
  $\mathcal{R}_e \leftarrow -\mathcal{P}\{\mathcal{M}_T(x_{t+1}; \mathcal{D}_0, \varnothing); y_{t+1}, F, I_{t+1}\}$ ▷ Evaluate example difficulty
  $r_{t+1} \leftarrow \mathcal{R}_q + \mathcal{R}_e$
  $a_{t+1} \sim p_{\mathcal{G}_E}(a|r_{t+1}, y_{t+1}, i_{t+1}, o_{t+1}, x_{t+1}, \mathcal{D}_0, m_6)$ ▷ Generate self-reflection action
end for

Subset	Question	Available Tools	Ground Truth
Simple-Python	During last night's basketball game, one of the star players was on fire, attempting a whopping 30 free throws. It's generally known that the average success rate for free throws hovers around 50%. I'm curious, with that success probability, what are the chances that the player made exactly 15 out of those 30 attempts?	`[calc_binomial_prob]`	`["calc_binomial_prob( n=30, k=15, p=0.5)"]`
Simple-REST	Can you provide me with the timezone information for the GPS coordinates of the Eiffel Tower (having latitude of 48.8584 and longitude of 2.2945), ensuring the response data is in a compact format, using my API key `<api_key>` and the host 'timezone-by-location.p.rapidapi.com'?	`[requests.get]`	`[requests.get( url=<url>, headers={ 'X-RapidAPI-Key': <api_key>, 'X-RapidAPI-Host': <url>}, params={'lat': 48.8584, 'lon': 2.2945, 'c': 1})]`
Parallel	I'm trying to understand my chances in a game with a 30% win probability per round. Can you calculate the probability of winning exactly 3 out of 10 rounds, 5 out of 15 rounds, and 7 out of 20 rounds?	`[calc_binomial_prob]`	`["calc_binomial_prob( n=10, k=3, p=0.3)", "calc_binomial_prob( n=15, k=5, p=0.3)", "calc_binomial_prob( n=20, k=7, p=0.3)"]`
Multiple	I have a set of vertices: [[1,2],[3,4],[1,4],[3,7]], and I'm curious about the area that these points, when connected in order, would enclose to form a polygon. Could you calculate the area of this polygon for me?	`[convert_coordinates, polygon_area, validate_polygon]`	`["polygon_area( vertices= [[1,2], [3,4], [1,4], [3,7]])"]`
Multiple-Parallel	I'm planning a small outdoor event in Ottawa, and I need to make sure the weather is going to cooperate. Could you fetch the current weather for me at latitude 45.4215 and longitude -75.6972 using theOpen-Meteo API? Also, I'm running a small game at the event, and I'm curious about the chances of winning. If I have 10 attempts at this game and the chance of winning each time is 50%, how likely is it that I'll win 5 times?	`[get_weather_data, calc_binomial_prob]`	`["get_weather_data( coordinates=[45.4215, -75.6972])", "calc_binomial_prob( n=10, k=5, p=0.5)"]`

Figure C.1: Dataset examples from BFCL.

Algorithm 3 DocumentationOptimizationStep

Input: $\mathcal{S}_t = D_t$: documentation, $\mathcal{E} = \{(x_j, F, I_j, y_j)\}_{j=1}^{W}$: validation set, a_t: reflection

Output: : $\mathcal{S}_{t+1} = \mathcal{D}_{t+1}$: updated documentation, r_{t+1}: score, a_{t+1}: reflection

1: $\mathcal{D}_{t+1} \sim p_{\mathcal{G}_D}(\mathcal{D}|\mathcal{D}_t, a_t, m_7)$ ▷ Sample documentation $\mathcal{D}_{t+1}$ from $\mathcal{G}_D$, applying reflection a_t

2: $\hat{M}_j, e_j \leftarrow \mathcal{M}_T(x_j; \mathcal{D}_{t+1}, \varnothing)\ \forall j$ ▷ Gather I/O & errors e_j from running $\mathcal{M}_T$ on validation set $\mathcal{E}$ with $\mathcal{D}_{t+1}$

3: $r_{t+1} \leftarrow \mathbb{E}_j[\mathcal{P}\{\hat{M}_j; y_j, F, I_j\}]$ ▷ Evaluation score on $\mathcal{E}$

4: $a_{t+1} \sim p_{\mathcal{G}_D}(a|\mathcal{D}_{t+1}, r_{t+1}, \{x_j, I_j, y_j, \hat{M}_j, e_j\}, m_8)$ ▷ Self-reflection action

---

You are given an API tool with the following documentation: {`Documentation`}

Your task is to write 1 example API call for the given API tool given its purpose and parameters list. The API call you produced will be executed as function call later and return result if correct, or error if you provide incorrect syntax, format, or parameters. Given the documentation and description, think of possible example API calls and produce those that are likely to be correctly executed. Think of parameter values that are reasonable, make sense, and are likely API calls that people use in the real world. The generated API call must be executable and real. Parameter values must be filled in and not placeholding text. You must include the required parameters, and optionally give parameters that are labeled as "optional parameters". Do not hallucinate and produce parameters that are not under "required" or "optional". Produce diverse parameter values, but be factual and do not use fake parameters.
You can only use the given function {`function_name`}. Create an API call that include the function name, and the parameters to be input to the API. Include all the required and optional parameters in a single dictionary without separating them. Do not include the URL or other irrelevant information. The output should be in the following JSON format that represents a function call: {"name": "function_name","parameters": {"properties": {"parameter_1": value 1}}} You must strictly follow the output format, including "name", "parameters", "properties", and parameters.

Previously you generated the following API calls for this function, which where then executed and critiqued:
fn_call="{`fn_call`}" fn_output="{`fn_output`}" status={`status`} reflection="{`reflection`}"

Figure C.2: Meta-prompt m_1 for Play2Prompt

You are given an API tool with the following documentation: {`Documentation`}

Previously you were asked to write an example API call for the function {`function_name`} given its purpose and parameters list, and you generated the following function call: {`fn_call`}. The function call you produced was later executed and returned the following result: "{`fn_output`}".

Your task is to analyze the response and check if there are any errors.
1. If there are no errors and everything looks reasonable, give an err_code of 0, and don't provide analysis.
2. If there is an error, give an err_code of -1. Then in your analysis, describe and analyze in detail why the error occurred based on the error message. Then, based on your analysis, give detailed suggestions to improve the function call so that no errors will be produced. You must give detailed analysis and suggestions, do not simply repeat the error message. The analysis and suggestions should be in the "analysis" field in the output.
Note that even if the "error" field in the result is empty, the "response" field may contain an error when using the function call. If this is the case you must treat this as an error and analyze the failure. The response field may also be in HTML format.

Your output should be in the following JSON format: {"analysis": your analysis and suggestions, "err_code": error code (-1 for error, 0 for correct)}

Figure C.3: Meta-prompt m_2 for PLAY2PROMPT

You are given an API tool with the following documentation: {Documentation}

For the function {function_name}, you are given the following function call: {fn_call}, and executing the function call returned the following result: {fn_output}.

Your task is to generate a user instruction in natural language that requires the given function call to be completed. Here are some guidelines to follow:
1. The instruction must be a scenario or problem that cannot be solved without calling the given function {function_name}. This is your main objective.
2. You should not directly or explicitly ask for the function to be called; the problem itself must inherently be solved by the function.
3. Based on the function, function call, its parameters, parameter values, and function execution responses, you should produce a real and reasonable instruction.
4. You must use information from the parameter values of the function call to create the response. You must include the value of every parameter from the given function call in the user instruction you generated, including each list/dict element of the parameter values. Do not ignore any parameters/values from the function call.
5. You must not include specific function calls in your response. You should not explicitly show the function names. You should also never explicitly name the parameter names in your response. You should not show any variable names.
6. Your response has to be in natural language. Do not show any variables, function calls, or code.
7. You should respond in the user's first-person perspective.
8. You are a human user. You are asking a question or giving an instruction. Do not answer in the perspective of an AI assistant. Remember, the user does not know about the API function and thus cannot ask to call the function.
9. Remember, you are asking a question, so do not answer your own question in the response. Your goal is to give a querying instruction or question, not producing answers or function calls.

Your output should be in the following JSON format: {"instruction": generated instruction}

Previously you generated the following instructions for this function call, which were rated and analyzed: instruction="{instruction}" score={score}
Based on these ratings, you are given the following analysis: {reflection}. You should improve your instructions based on these suggestions.

Figure C.4: Meta-prompt m_3 for PLAY2PROMPT

You are given an API tool with the following documentation: {`Documentation`}

You are given the following instruction: "{`instruction`}" To produce a response to the instruction, you made an API call to the given tool, which returned the following results: {`fn_output`}
Given the instruction and the results of API call, produce an effective and short answer to the user in natural language. Your answer must be based on the results of the API call, do not hallucinate or answer anything not in the API results. You must not include code, comments, JSON data structures, notes, or other irrelevant information in your answer. If there is an error or failure using the tool, you must report the error in your answer and do not make things up, especially when you receive an input about invalid parameters.

Figure C.5: Meta-prompt m_4 for PLAY2PROMPT

You are given an instruction "{`instruction`}", function call "{`fn_call`}" and an answer "{`answer`}", your task is to give a 'score' based on the following rules:
1. You must return 1 if any of the following conditions is met (for instruction only): (1) instruction is empty, nonsense, or not in natural language; or (2) instruction is explicitly including function calls or asking for function calls or contains function names; or (3) instruction includes exact function parameter names; or (4) instruction includes code or variable assignment; or (5) instruction is longer than 3 sentences or 300 letters; or (6) instruction does not include a question, query, request, or problem to be solved; or (7) instruction is not in first-person perspective, or is in the perspective of an AI assistant instead of a user; or (8) any parameter value in the function call is not present in the instruction
An instruction that satisfies any of these conditions is a bad instruction and should be scored a 1.
2. If the answer is a error message or mentions any errors (API error, invalid parameter error, ..., etc.), mentions cannot use API or cannot respond, return 1.
3. If the answer is a positive response for the given instruction, you have to further check.
3.1 If the answer is not sufficient to determine whether they solve the instruction or not, return 2.
3.2 If you are confident that the answer is sufficient to determine whether the solve the instruction or not, return 3 if solvable or 1 if unsolvable.

Finally, organize your output in the following JSON format:{"analysis": your reasoning, "score": score}

Figure C.6: Meta-prompt m_5 for PLAY2PROMPT

You are given an API tool with the following documentation: {Documentation}

Previously, given the function call {fn_call}, you were asked to generate example instructions that require the use of the function {function_name} to complete. The example instructions generated by you were then scored by an expert on whether the instructions can be fulfilled using the given API function. Scores are in a scale between 1 (lowest) and 3 (highest).

Below are the generated instructions, scores, and analyses:
instruction="{instruction}" score={score} analysis="{analysis}"

Task:
1. Firstly, identify and contrast the patterns of instructions and function calls that have achieved good scores with those that have not. If there are no bad scores, only summarize the patterns of the good ones.
2. Next, specify the suggestions that can lead to improved performance for the generated instructions and function calls with bad scores. You should focus on capturing the high-level pattern of the examples relevant to the API documentation. Note that both the function and the function call cannot be changed, and focus your suggestions on how to improve the example instructions, including deciding what information to use from parameters of the function call.

Figure C.7: Meta-prompt m_6 for PLAY2PROMPT

You are given an API tool with the following documentation: {Documentation}

Previously, the given tool was used in solving instructions by a tool assistant with the following descriptions:
Iteration #{iteration}, description="{description}", score={score}%, stdev={stdev}.

Furthermore, an analysis was performed on the descriptions for the previous iterations: "{analysis}"

Your task is to further enhance the description for the function {function_name} based on these results for the next iteration, with the objective of maximizing the score, minimizing the stdev, and help the assistant correctly use the function without errors. The descriptions for each parameter might be unclear, underspecified, or incorrect, so you should include clear parameter descriptions and usage for every single required and optional parameter, including its type, usage, and possible values. Be as clear, descriptive, and comprehensive as possible. Be factual and do not consider parameters that are not listed. Incorporate the analysis and generate the enhanced descriptions.

Figure C.8: Meta-prompt m_7 for PLAY2PROMPT

You are given an API tool with the following documentation: {Documentation}

Previously, the given tool was used in solving instructions by a tool assistant with the following descriptions:
Iteration #{iteration}, description="{description}"

Here are the instructions the assistant tried to solve with this tool description, with their corresponding answers and errors produced by the assistant:
instruction="{instruction}", answer="{answer}", errors: function_call={function_name}, arguments={arguments}, error={error_message}, ground truth should be {fn_call}

Overall the performance of this description is: score={score}

Now your task is to critique the descriptions based on these results. A good description maximizes the score, minimizing the stdev, and helps the assistant correctly use the function without errors. In your analysis:
(1) Identify how the descriptions affect the function call errors of the assistant. Be specific on which errors the assistant tends to make, and find patterns in the description that causes the assistant to make such errors.
(2) Identify and contrast the patterns of descriptions that have achieved good scores ($> 60\%$) with those that have not. Analyze how the description can be improved.

Figure C.9: Meta-prompt m_8 for PLAY2PROMPT

Bibliography

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*.

L. Bahl, P. Brown, P. de Souza, and R. Mercer. 1986. Maximum mutual information estimation of hidden markov model parameters for speech recognition. In *ICASSP '86. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 11, pages 49–52.

Alexander Bondarenko, Maik Fröbe, Meriem Beloucif, Lukas Gienapp, Yamen Ajjour, Alexander Panchenko, Chris Biemann, Benno Stein, Henning Wachsmuth, Martin Potthast, and Matthias Hagen. 2020. Overview of touché 2020: Argument retrieval. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction*, pages 384–395, Cham. Springer International Publishing.

Vera Boteva, Demian Gholipour, Artem Sokolov, and Stefan Riezler. 2016. A full-text learning to rank dataset for medical information retrieval. In *Advances in Information Retrieval*, pages 716–722, Cham. Springer International Publishing.

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*.

Danqi Chen and Wen-tau Yih. 2020. Open-domain question answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 34–37, Online. Association for Computational Linguistics.

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.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin,

Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

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.

Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel Weld. 2020. SPECTER: Document-level representation learning using citation-informed transformers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2270–2282, Online. 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*.

Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407.

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

Thomas Diggelmann, Jordan Boyd-Graber, Jannis Bulian, Massimiliano Ciaramita, and Markus Leippold. 2020. Climate-fever: A dataset for verification of real-world climate claims. *arXiv preprint arXiv:2012.00614*.

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*.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. 2023. Faith and fate: Limits of transformers on compositionality. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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 and James Glass. 2026. Beyond single-shot: Multi-step tool retrieval via query planning. *arXiv preprint arXiv:2601.07782*.

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.

Fernando Ferraretto, Thiago Laitz, Roberto Lotufo, and Rodrigo Nogueira. 2023. Exaranker: Synthetic explanations improve neural rankers. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 2409–2414, New York, NY, USA. Association for Computing Machinery.

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. 2023a. 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.

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

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.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11143–11156, Bangkok, Thailand. 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.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3929–3938. PMLR.

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

Faegheh Hasibi, Fedor Nikolaev, Chenyan Xiong, Krisztian Balog, Svein Erik Bratsberg, Alexander Kotov, and Jamie Callan. 2017. Dbpedia-entity v2: A test collection for entity search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 1265–1268, New York, NY, USA. Association for Computing Machinery.

Doris Hoogeveen, Karin M. Verspoor, and Timothy Baldwin. 2015. Cqadupstack: A benchmark data set for community question-answering research. In *Proceedings of the 20th Australasian Document Computing Symposium (ADCS)*, ADCS '15, pages 3:1–3:8, New York, NY, USA. ACM.

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*.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2023. Atlas: Few-shot Learning with Retrieval Augmented Language Models. *Journal of Machine Learning Research*, 24(251):1–43.

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].

Samy Jelassi, Stéphane d'Ascoli, Carles Domingo-Enrich, Yuhuai Wu, Yuanzhi Li, and François Charton. 2023. Length generalization in arithmetic transformers. *arXiv preprint arXiv:2306.15400*.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. 2023. Large language models struggle to learn long-tail knowledge. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 15696–15707. PMLR.

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

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.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

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.

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

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6086–6096, Florence, Italy. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

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).

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016. A diversity-promoting objective function for neural conversation models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 110–119, San Diego, California. Association for Computational Linguistics.

Jiwei Li and Dan Jurafsky. 2016. Mutual information and diverse decoding improve neural machine translation. *arXiv preprint arXiv:1601.00372*.

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 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*, pages 2356–2362.

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*.

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 Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. 2024a. APIGen: Automated PIpeline for Generating Verifiable and Diverse Function-Calling Datasets. In *Advances in Neural Information Processing Systems*, volume 37, pages 54463–54482. Curran Associates, Inc.

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*.

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Jing Lu, Gustavo Hernandez Abrego, Ji Ma, Jianmo Ni, and Yinfei Yang. 2021. Multi-stage training with improved negative contrast for neural passage retrieval. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6091–6103, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

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 under-documented: Simple document expansion boosts tool retrieval. *arXiv preprint arXiv:2510.22670*.

Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. 2021. Sparse, dense, and attentional representations for text retrieval. *Transactions of the Association for Computational Linguistics*, 9:329–345.

Hongyin Luo, Shang-Wen Li, Mingye Gao, Seunghak Yu, and James Glass. 2022. Cooperative self-training of machine reading comprehension. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 244–257, Seattle, United States. Association for Computational Linguistics.

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.

Zhuang Ma and Michael Collins. 2018. Noise contrastive estimation and negative sampling for conditional models: Consistency and statistical efficiency. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3698–3707, Brussels, Belgium. 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.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: iterative refinement with self-feedback. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 46534–46594.

Macedo Maia, Siegfried Handschuh, André Freitas, Brian Davis, Ross McDermott, Manel Zarrouk, and Alexandra Balahur. 2018. Www'18 open challenge: Financial opinion mining and question answering. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, page 1941–1942, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

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. 2021a. 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.

Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. 2021b. Reader-guided passage reranking for open-domain question answering. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 344–350, 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.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, and 1 others. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2017. MS MARCO: A human-generated MAchine reading COmprehension dataset.

Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. 2025. Arithmetic without algorithms: Language models solve math with a bag of heuristics. In *The Thirteenth International Conference on Learning Representations*.

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

Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. 2020. Document ranking with a pretrained sequence-to-sequence model. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 708–718, Online. Association for Computational Linguistics.

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

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

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

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

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7957–7968, Singapore. Association for Computational Linguistics.

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*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, and 1 others. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.

Yasaman Razeghi, Robert L Logan IV, Matt Gardner, and Sameer Singh. 2022. Impact of pretraining term frequencies on few-shot numerical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 840–854, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Revanth Gangi Reddy, Vikas Yadav, Md Arafat Sultan, Martin Franz, Vittorio Castelli, Heng Ji, and Avirup Sil. 2021. Towards robust neural retrieval models with synthetic pre-training. *arXiv preprint arXiv:2104.07800*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Stephen Robertson, Hugo Zaragoza, and 1 others. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

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

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

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.

G. Salton, A. Wong, and C. S. Yang. 1975. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620.

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, and 21 others. 2022. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*.

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*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Christopher Sciavolino, Zexuan Zhong, Jinhyuk Lee, and Danqi Chen. 2021. Simple entity-centric questions challenge dense retrievers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6138–6148, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

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*.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. 2024. Taskbench: Benchmarking large language models for task automation. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

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.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online. Association for Computational Linguistics.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. 2021. Retrieval augmentation reduces hallucination in conversation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3784–3803, Punta Cana, Dominican Republic. 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*.

Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21. _eprint: https://www.emerald.com/jd/article-pdf/28/1/11/1336479/eb026526.pdf.

Shang-Yu Su, Yung-Sung Chuang, and Yun-Nung Chen. 2020. Dual inference for improving language understanding and generation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4930–4936, Online. Association for Computational Linguistics.

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*.

Duyu Tang, Nan Duan, Tao Qin, Zhao Yan, and Ming Zhou. 2017. Question answering and question generation as dual tasks. *arXiv preprint arXiv:1706.02027*.

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*.

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

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*.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. FEVER: a large-scale dataset for fact extraction and VERification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 809–819, New Orleans, Louisiana. Association for Computational Linguistics.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Christophe Van Gysel and Maarten de Rijke. 2018. Pytrec_eval: An extremely fast python interface to trec_eval. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '18, page 873–876, New York, NY, USA. Association for Computing Machinery.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Ellen Voorhees, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, William R. Hersh, Kyle Lo, Kirk Roberts, Ian Soboroff, and Lucy Lu Wang. 2021. Trec-covid: Constructing a pandemic information retrieval test collection. *SIGIR Forum*, 54(1).

Ellen M Voorhees and 1 others. 1999. The trec-8 question answering track report. In *Trec*, volume 99, pages 77–82. Citeseer.

Henning Wachsmuth, Shahbaz Syed, and Benno Stein. 2018. Retrieval of the best counterargument without prior topic knowledge. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 241–251, Melbourne, Australia. Association for Computational Linguistics.

David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. 2020. Fact or fiction: Verifying scientific claims. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7534–7550, Online. Association for Computational Linguistics.

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. 2024a. 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.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. 2024b. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *The Twelfth International Conference on Learning Representations*.

Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. 2013. A theoretical analysis of ndcg type ranking measures. In *Conference on learning theory*, pages 25–54. PMLR.

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

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

P.C. Woodland and D. Povey. 2002. Large scale discriminative training of hidden markov models for speech recognition. *Computer Speech & Language*, 16(1):25–47.

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*.

Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis N Ioannidis, Karthik Subbian, Jure Leskovec, and James Zou. 2024. Avatar: Optimizing llm agents for tool-assisted knowledge retrieval. *arXiv preprint arXiv:2406.11200*.

Yingce Xia, Tao Qin, Wei Chen, Jiang Bian, Nenghai Yu, and Tie-Yan Liu. 2017. Dual supervised learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3789–3798.

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*.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Berkeley function calling leaderboard.

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*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

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.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

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.

Donggeun Yoo and In So Kweon. 2019. Learning loss for active learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

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.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Ren Kan, Dongsheng Li, and Deqing Yang. 2024. Easytool: Enhancing llm-based agents with concise tool instruction. *arXiv preprint arXiv:2401.06201*.

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*.

Kexun Zhang, Hongqiao Chen, Lei Li, and William Wang. 2023. Syntax error-free and generalizable tool use for llms via finite-state decoding. *arXiv preprint arXiv:2310.07075*.

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.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*.

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*.