RAG-MCP: MITIGATING PROMPT BLOAT IN LLM TOOL SELECTION VIA RETRIEVAL-AUGMENTED GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) struggle to effectively utilize a growing number of external tools, such as those defined by the Model Context Protocol (MCP) (Anthropic, 2024), due to prompt bloat and selection complexity. We introduce RAG-MCP, a Retrieval-Augmented Generation framework that overcomes this challenge by offloading tool discovery. RAG-MCP uses semantic retrieval to identify the most relevant MCP(s) for a given query from an external index before engaging the LLM. Only the selected tool descriptions are passed to the model, drastically reducing prompt size and simplifying decision-making. Experiments, including an MCP stress test, demonstrate RAG-MCP significantly cuts prompt tokens (e.g., by over 50%) and more than triples tool selection accuracy (43.13% vs 13.62% baseline) on benchmark tasks. RAG-MCP enables scalable and accurate tool integration for LLMs.

1 INTRODUCTION

1.1 BACKGROUND AND MOTIVATION

Large Language Models (LLMs) have demonstrated remarkable capabilities in natural dialogue, reasoning, and even code generation. However, they remain fundamentally constrained by the knowledge encoded in their parameters and the fixed context window available at inference time. In essence, an LLM without external access is "trapped" with only its training data and cannot easily update its knowledge or perform actions in the world (Patil et al., 2024). To address this limitation, recent efforts have focused on augmenting LLMs with **external tools and function-calling** abilities (Chen et al., 2024). By invoking tools (e.g. web search, databases, calculators) via defined functions or APIs, an LLM can fetch up-to-date information and execute complex operations beyond its built-in repertoire (Patil et al., 2024). This paradigm - often referred to as zero-shot tool use or function calling — allows AI assistants to interface with the latest data and services, unlocking applications from real-time knowledge queries to specialized tasks in finance and travel planning (Chen et al., 2024). In fact, major AI providers have embraced this trend: for example, leading LLM platforms now support plugin APIs and structured function calls so that models like GPT-4 or Claude can invoke external services through well-defined interfaces (Patil et al., 2024).

In the research community, a variety of approaches have been proposed to enable and improve LLM tool use. Prompt-based strategies such as **ReAct** intermix reasoning steps with action commands, allowing an LLM to decide when to consult a tool in the context of a multi-turn "thought process" (Yao et al., 2023). Model-centric approaches have also emerged: for instance, **Toolformer** fine-tunes an LLM to autonomously decide which API to call, when to call it, and how to incorporate the result, given only a handful of demonstrations per tool (Schick et al., 2023). Other researchers have improved tool-use by incorporating it into training data and model tuning. This includes blending function call demonstrations into instruction-following datasets and exploring prompt formats that effectively describe available functions to the model (Chen et al., 2024). Such efforts have markedly enhanced zero-shot tool usage performance. For example, fine-tuning a model on API call tasks with extensive tool-use data can yield impressive results – the **Gorilla** system augmented a 7B LLaMA-based model with relevant API documentation retrieval, enabling it to outperform even GPT-4 in generating correct API calls for a wide range of tools (Patil et al., 2024). An important

insight from these works is that providing just-in-time relevant context (be it through optimized prompts or retrieved documentation) greatly boosts the accuracy of an LLM's tool selection and use, while mechanisms for the model to explicitly decide on tool use (such as special decision tokens for "answer vs. act") can further improve reliability (Chen et al., 2024).

Despite this progress, a new challenge arises as we scale up the number of tools available to an LLM. Most prior studies and deployments consider a relatively small set of tools or APIs, often hand-picked and easy for the model to handle within a prompt (Patil et al., 2024). In practice, however, the ecosystem of tools is rapidly expanding. For instance, Anthropic's recently introduced **Model Context Protocol (MCP)** defines a universal, open standard for connecting AI systems with external data sources and services. MCP enables a single assistant to interface with many data repositories and business tools through a unified protocol, replacing fragmented one-off integrations. As a result, an advanced LLM agent could soon have dozens of functions at its disposal – from Google Drive and Slack connectors to GitHub, databases, maps, and more – all registered as MCP "tools" it can call (Anthropic, 2024). This proliferation of available tools brings significant hurdles.

Prompt Bloat is one critical issue: providing the definitions or usage instructions for every possible tool in the model's context would consume an enormous number of tokens and risk overwhelming the model. It has been observed that it is effectively impossible to describe a large collection of APIs or tools in a single prompt as their number grows, and many APIs have overlapping functionalities with only nuanced differences. Including too many at once not only exhausts the context length, but can also confuse the model – the functions may start to blur together. This leads directly to a second issue: **decision overhead**. With a long list of tools (many of them similar in scope), the model faces a more complex decision when choosing if and which tool to invoke. The greater the choice, the higher the chance of error, such as selecting an suboptimal tool or misinterpreting what a tool does. Indeed, even state-of-the-art models can misfire in such settings: for example, in a scenario with numerous API options, GPT-4 was reported to hallucinate an API that doesn't actually exist, and Anthropic's Claude picked the wrong library for the user's request (Patil et al., 2024). These failure cases underscore that **naively scaling up the toolset can degrade an LLM's performance**, due to both the capacity strain on the prompt and the ambiguity in the model's decision process.

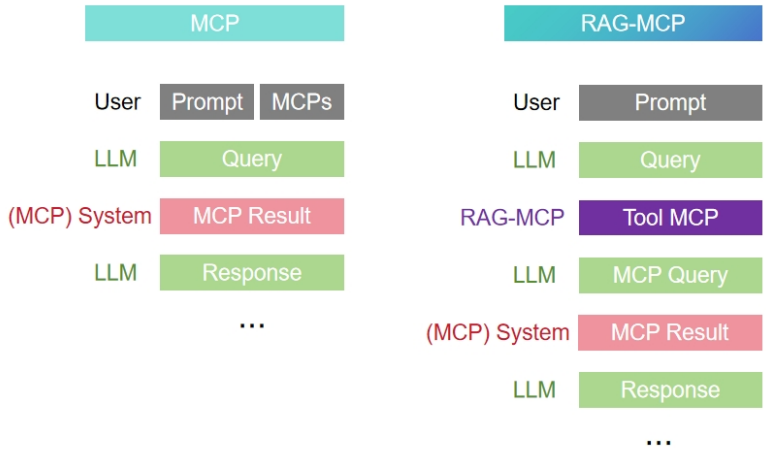

Figure 1: Comparation between MCP and RAG-MCP during inference

To tackle these challenges, we propose **RAG-MCP**, a solution that marries Retrieval-Augmented Generation (RAG) with the Model Context Protocol framework. The key idea of RAG-MCP is to avoid presenting all tools to the language model at once, and instead dynamically retrieve a relevant subset of tools based on the user's query. In our approach, the numerous available tool descriptions (MCP function schemas, usage examples, etc.) are stored in an external memory indexed by their semantics. When a new query arrives, a dedicated retriever (e.g. a vector-space semantic search) first selects the top-k candidate tools that are most likely to be useful for that query. Only these k tool descriptions are then injected into the LLM's prompt (or provided via the function-calling

API), greatly reducing context length and complexity. This retrieval step serves as a form of focused **context filtering**, which cuts down on prompt bloat and guides the model's choice. The approach is analogous to how retrieval-augmented QA systems work: rather than feed the entire Wikipedia to the model, one retrieves only the relevant articles (Lewis et al., 2020). Here, instead of static knowledge, we retrieve **actionable tool knowledge** on the fly. An added benefit is extensibility – because the tool information lives in an external index, new tools or updated APIs can be incorporated by updating that index without retraining the LLM, ensuring the system remains up-to-date (Patil et al., 2024). In short, **retrieval helps tame the growing toolset** by providing the right tools at the right time, thereby reducing the model's decision burden.

1.2 Contributions

This paper's contributions are threefold. First, we introduce the **RAG-MCP Framework**, a novel architecture that integrates a retrieval mechanism with LLM function calling. This framework empowers an LLM to manage a large arsenal of tools by dynamically querying a tool repository for relevant options, rather than being prompted with an exhaustive list. Second, we develop a **Scalable Tool Retrieval** module that efficiently matches user queries to the most pertinent tools in a vector space, significantly mitigating prompt bloat and narrowing the decision space for the LLM. This module's design allows new tools to be indexed on the fly without LLM fine-tuning. Finally, we provide empirical evidence of **Improved Tool-Use Performance**. Through comprehensive experiments, we demonstrate that RAG-MCP substantially enhances tool selection accuracy and overall task success, effectively counteracting the performance degradation observed when naively scaling the number of available tools.

Overall, our work demonstrates that the integration of retrieval-based context management is a promising direction to counteract the challenges of tool proliferation in LLMs. By enabling models to learn which tool to use out of many and only providing information for those tools, **RAG-MCP** offers a practical solution for the next generation of AI agents operating with extensive toolkits. It combines the strengths of retrieval augmentation and standardized tool APIs to ensure that more tools do not mean worse performance but rather a broader range of skills that the model can deploy accurately and efficiently.

2 Related Work

2.1 Tool Use in LLMs

LLMs have been augmented with external tools to overcome limitations in arithmetic, retrieval, and code execution. **Toolformer** demonstrates a self-supervised method by which a model learns when and how to call APIs such as calculators or search engines, improving zero-shot performance across tasks (Schick et al., 2023). **ReAct** interleaves chain-of-thought reasoning with action steps to interact with external environments (e.g., a Wikipedia API), yielding more interpretable and accurate multi-step solutions (Yao et al., 2023). **WebGPT** fine-tunes GPT-3 in a simulated browser environment, training it to navigate, search, and cite sources for long-form Q&A, reducing hallucinations via grounded retrieval (Nakano et al., 2022). More recently, **ChatGPT Plugins** introduced a production plugin ecosystem, allowing ChatGPT to access up-to-date information and third-party services in a controlled, safety-oriented framework (OpenAI, 2023).

2.2 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) first combined parametric LLMs with non-parametric memory in a dense vector index, retrieving relevant passages at inference time to improve knowledge-intensive tasks (Lewis et al., 2020). Subsequent work has extended RAG to broad NLP paradigms, including modular and advanced RAG variants that dynamically adapt retrieval per token or per query (Gao et al., 2023). RAG's decoupling of memory access and generation inspires our MCP-RAG approach, wherein MCP discovery is treated as a retrieval subproblem, orthogonal to core text generation.

2.3 Model Context Protocol

The Model Context Protocol standardizes LLM-to-API interactions by bundling resource prompts, authentication, and parameter schemas into modular "MCP" servers. MCPs act as function-call extensions, similar to OpenAI's function-calling API, but with greater community extensibility. The rapid growth of MCP repositories (4,400+ servers on mcp.so as of April 2025 (ShipAny, 2025)) underscores the need for scalable discovery and validation mechanisms.

3 Methodology

Overview. We study how the number of available MCP servers affects an LLM's ability to select and invoke the correct tool ("prompt bloat") and present MCP-RAG, a retrieval-augmented framework that mitigates this degradation by dynamically retrieving only the most relevant MCP for each query.

3.1 Prompt Bloat and the MCP Stress Test

Modern LLMs must often choose among many possible external tools, each described by an MCP schema. As the count of MCPs grows, including all their descriptions in a single prompt leads to prompt bloat: the context window becomes saturated with distractors, reducing the model's capacity to distinguish and recall the correct tool.

This phenomenon parallels the Needle-in-a-Haystack (NIAH) test, which embeds a random fact (the "needle") in the middle of a long context (the "haystack") and measures an LLM's ability to retrieve it under varying context lengths and depths (Lewis et al., 2020; OpenAI). In NIAH, performance drops sharply as the haystack grows, revealing limits of in-context retrieval.

Inspired by NIAH, we design an **MCP stress test** on WebSearch tasks: for each trial, we present the model with N MCP schemas (one ground-truth and $N-1$ distractors) and ask it to select and invoke the correct WebSearch MCP. We vary N from 1 to 11100 in 26 intervals, measuring selection accuracy, task success, prompt token usage, and latency. This setup quantifies how tool-selection ability degrades with increasing MCP pool size.

3.2 RAG-MCP Framework

To overcome prompt bloat, RAG-MCP applies **Retrieval-Augmented Generation (RAG)** principles to tool selection. Instead of flooding the LLM with all MCP descriptions, we maintain an external vector index of all available MCP metadata. At query time, the process unfolds in three stages:

1. **Retrieval.** A lightweight LLM-based retriever (e.g., Qwen) encodes the user's task description and performs a semantic search over the MCP index, returning the top-k candidate MCPs most similar to the task (Lewis et al., 2020).
2. **Validation.** For each retrieved MCP, RAG-MCP can generate a few-shot example query and test its response to ensure basic compatibility, functioning as a "sanity check" before invocation.
3. **Invocation.** Only the single best MCP description, including its tool-use parameters, is injected into the LLM prompt or function-calling API, which then performs planning and execution without concern for tool discovery (Blog, 2025).

This design yields several significant benefits. Primarily, by supplying only relevant MCP metadata, RAG-MCP offers **Reduced Prompt Size**, thus avoiding context window overload. This focus, in turn, leads to a **Lower Cognitive Load** for the LLM, as it no longer needs to sift through hundreds of distracting tool descriptions, thereby improving selection accuracy and reducing hallucinations (Blog, 2025). Furthermore, the framework achieves greater **Resource Efficiency** by activating only the selected MCP, unlike conventional clients that must instantiate all registered servers. This on-demand activation enables support for arbitrarily large toolsets without creating infrastructure bottlenecks (OpenAI). Finally, this approach ensures **Multi-Turn Robustness**, as the retriever dynamically handles tool recall in ongoing dialogues, freeing up context space for task-specific reasoning.

3.3 Three-Step Pipeline Diagram

We summarize RAG-MCP's operation in three core steps, as depicted in the flowchart in Fig. 2. The pipeline begins with the user's natural-language task being encoded and submitted to the retriever. Next, the retriever searches the vector index of MCP schemas, ranks candidates by semantic similarity, and can optionally test each via synthetic examples for validation. In the final step, the LLM receives only the selected MCP's schema and parameters to execute the task via its function-calling interface.

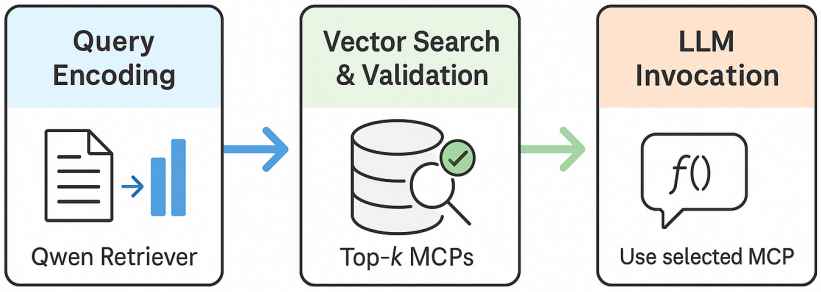

Figure 2: RAG-MCP pipeline: (1) encode user query with Qwen-max, (2) retrieve & validate top-k MCPs, and (3) invoke chosen MCP

By decoupling tool discovery from generation, RAG-MCP ensures that LLMs can scale to hundreds or thousands of MCPs without suffering prompt bloat or decision fatigue, much as RAG systems avoid overwhelming an LLM with entire corpora by retrieving only relevant passages.

3.4 Discussion

Our methodology combines the rigor of **stress testing** (via the MCP stress test) with the effectiveness of retrieval-augmented tool use. The stress test quantifies the sharp performance drop that occurs when distractor MCPs swell the prompt, mirroring long-context recall failures in NIAH evaluations (gkamradt, 2024). RAG-MCP then counteracts this by dynamically narrowing the toolset, reducing both prompt tokens and decision complexity, and thereby restoring—and often improving—task success rates.

Furthermore, by using an external index, RAG-MCP remains extensible: new MCPs can be added by indexing their metadata, without retraining the LLM. And by selectively activating servers on demand, it sidesteps the practical limits on simultaneous MCP instantiation faced by prior tool-augmented LLM deployments.

4 Experiments

4.1 Stress Test

4.1.1 Setup

To quantify how an LLM's tool-selection ability scales with the size of the MCP pool, we conduct a stress test in which the number of candidate MCP servers, N, is varied from 1 to 11100 in intervals, while the key MCP server located from the top to the bottom. For each value of N, we randomly select one "ground-truth" MCP (i.e., the only server capable of satisfying the task requirement) and $N-1$ distractor MCPs drawn from our full registry of over 4,400 publicly listed servers (ShipAny, 2025). This design ensures that exactly one in every N candidates is relevant. We then present the model with each of 20 web-search tasks, requiring it to (a) choose the correct MCP, (b) issue a valid query or answer, and (c) return the final result.

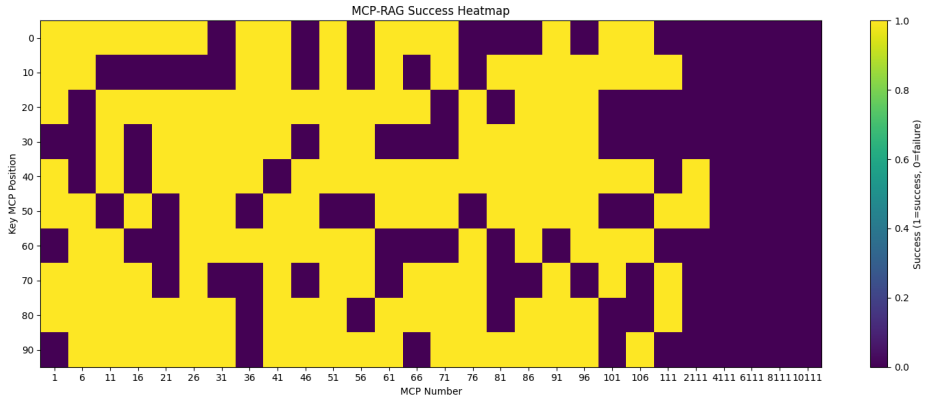

Figure 3: This figure illustrates per-trial success across MCP positions from 1 to 11100, where yellow denotes successful selection and purple denotes failure.

4.1.2 RESULTS

Figure 3 plots selection accuracy and task success as N increases. We observe a clear non-monotonic trend: These results quantitatively confirm that while MCP-RAG greatly mitigates prompt bloat and maintains high performance in small to moderate pools, its retrieval precision and overall throughput degrade as the tool registry scales to thousands of MCPs.

4.2 RAG-MCP

4.2.1 SETUP

We evaluated all methods in the web search subset of MCPBench (Luo et al., 2025), which we used as our heldout testbed. For each baseline, we perform 20 independent trials, and we deem a baseline successful if it produces more than 10 correct answers out of those 20. Within each trial, the model may engage in up to 10 rounds of interaction with the MCP servers in order to arrive at its final response.

To assess answer correctness in an automated and reproducible manner, we employ Deepseek-v3 (Liu et al., 2024) as our evaluator. Because MCP servers require external network access—and can therefore be sensitive to latency or transient failures—we enforce a controlled network environment throughout all experiments, ensuring no requests fail due to connectivity issues. Finally, all trials are driven by qwen-max-0125 as our underlying base LLM.

4.2.2 BASELINES

We evaluate three selection strategies in our experiments:

Blank Conditioning: Prompt the LLM with all N MCP descriptions at once and ask it to choose the correct one.

Actual Match: Pre-filter the candidate pool using simple keyword matching on the task description and MCP metadata, then prompt the model on this reduced set.

RAG-MCP: Employ our vector-index retriever to semantically rank all N MCPs and inject only the top candidate's schema into the LLM prompt for execution.

4.2.3 METRICS

We evaluate performance using three key metrics for each baseline method:

Accuracy (%): Percentage of trials in which the model selected the ground-truth MCP.

Avg Prompt Tokens: Mean number of tokens consumed by the prompt, including injected MCP metadata.

Avg Completion Tokens: Mean number of tokens generated by the model as its final output.

Judgment of the final answer is automated using a Llama-based verifier ("Llama as Judge") to compare model outputs against ground truth.

4.2.4 RESULTS

Table 1 summarizes the performance of the evaluated baseline methods, clearly demonstrating the effectiveness of MCP-RAG:

Baseline	Accuracy (%)	Avg Prompt Tokens	Avg Completion Tokens
MCP-RAG	43.13	1084.00	78.14
Actual Match	18.20	1646.00	23.60
Blank	13.62	2133.84	162.25

Table 1: Baseline performance comparison on accuracy and token usage

As the table shows, **MCP-RAG** achieves the highest accuracy at **43.13%**, significantly outperforming the Actual Match and Blank Conditioning methods, which scored **18.20%** and **13.62%**, respectively. Furthermore, MCP-RAG notably reduces the average number of prompt tokens to **1084**, reflecting a substantial reduction compared to the other baselines, especially Blank Conditioning, which requires **2133.84** tokens. While MCP-RAG shows an increase in completion tokens (**78.14**) compared to Actual Match (**23.60**), this trade-off is beneficial as it correlates with a higher accuracy and overall task success rate.

5 ANALYSIS

5.1 STRESS TEST ANALYSIS

Figure 3 illustrates per-trial success across MCP positions from 1 to 11100, where yellow denotes successful selection and purple denotes failure. We observe that:

High Early-Stage Success: MCP positions below 30 exhibit predominantly yellow regions, indicating success rates above 90% when the candidate pool is minimal.

Mid-Range Variability: In the range of positions 31–70, clusters of purple emerge intermittently, reflecting lower accuracy as semantic overlap among MCP descriptions increases.

Performance Degradation at Scale: Beyond position ˜100, purple dominates, signifying that retrieval precision diminishes when handling very large tool registries.

Residual Success Islands: Occasional yellow patches at higher positions suggest that certain MCPs remain well-aligned to specific queries, providing robustness even in extensive pools.

These patterns confirm that while MCP-RAG effectively curbs prompt bloat and maintains high accuracy in small to moderate MCP pools, retrieval precision challenges arise as the total number of MCPs grows, motivating future work on hierarchical or adaptive retrieval mechanisms.

5.2 ANALYSIS OF RAG-MCP RESULTS

The superior performance of RAG-MCP can be attributed to several factors. Its method of **Focused Context Filtering** ensures that the model avoids distraction from irrelevant tool descriptions by injecting only the single most relevant MCP schema, resulting in clearer decision boundaries. This leads to greater **Prompt Efficiency**, as the dramatic reduction in prompt tokens allows the model to allocate more of its context window to reasoning about the task itself rather than parsing extraneous metadata. Finally, while RAG-MCP shows a slight increase in completion tokens, this reflects a

Balanced Generation strategy where the overhead is invested in more thorough reasoning and verification steps, which directly correlate with higher accuracy.

Overall, these findings confirm that retrieval-augmented selection of MCPs effectively tames prompt bloat and enhances an LLM's tool-selection reliability, making RAG-MCP a compelling solution for scalable external tool integration.

6 Conclusion

In this work, we presented **RAG-MCP**, a framework that tames large toolsets by retrieving only the most relevant MCP schema for each query. Our approach **drastically reduces prompt size**—cutting token usage by over half compared to naïve methods—and **boosts selection accuracy** by more than tripling the success rate under heavy tool load. A key advantage of our design is its **extensibility**, as new MCPs can be indexed on-the-fly without retraining the core language model.

In essence, RAG-MCP transforms a sprawling, unmanageable library of tools into a lean, on-demand toolkit, equipping LLM agents to wield vast external services with precision and efficiency. Future work will focus on refining retrieval at extreme scale, potentially through hierarchical indexes or adaptive strategies, and extending the framework to handle multi-tool workflows in real-world agent deployments. RAG-MCP lays the "golden core" for the next generation of scalable and reliable AI agents.

References

Anthropic. Introducing the model context protocol, 2024. URL https://www.anthropic.com/news/model-context-protocol.

NIVIDA Blog. What is retrieval-augmented generation aka rag, 2025. URL https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/.

Yi-Chang Chen, Po-Chun Hsu, Chan-Jan Hsu, and Da-shan Shiu. Enhancing function-calling capabilities in llms: Strategies for prompt formats, data integration, and multilingual translation. *arXiv preprint arXiv:2412.01130*, 2024.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2, 2023.

gkamradt. The needle in a haystack test, 2024. URL https://github.com/gkamradt/LLMTest_NeedleInAHaystack.

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. Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Zhiling Luo, Xiaorong Shi, Xuanrui Lin, and Jinyang Gao. Evaluation report on mcp servers, 2025. URL https://arxiv.org/abs/2504.11094.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback, 2022. URL https://arxiv.org/abs/2112.09332.

OpenAI. Openai function calling. URL https://platform.openai.com/docs/guides/function-calling.

OpenAI. Chatgpt plugins, 2023. URL https://openai.com/index/chatgpt-plugins.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/d842425e4bf79ba039352da0f658a906-Paper-Conference.pdf.

ShipAny. Mcp servers, 2025. URL https://mcp.so/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *International Conference on Learning Representations (ICLR)*, 2023. URL https://par.nsf.gov/biblio/10451467.