Representing Agentic Tools in Knowledge Graphs for Structure-Aware Tool Discovery Under Tool Overload

Isaiah Onando Mulang'[1], Johannes Thaller[1], Tushar Trivedi[1], Lars Heling[1] and Felix Sasaki[1]

[1]SAP SE, Dietmar-Hopp-Allee 16, 69190 Walldorf, Germany

Abstract

Large language model (LLM) agents increasingly rely on external tools, yet most tool ecosystems still expose those tools as unstructured textual descriptions or JSON schemas. As tool inventories grow, this becomes a retrieval problem where the agent must surface a small relevant set under context and tool-budget constraints. We study knowledge-graph-based tool representation for agentic systems through a lightweight ontology for Model Context Protocol (MCP) tools. The ontology models tools, servers, capabilities, and parameters, and treats required versus optional inputs as first-class relations. Using real MCP tool schemas extracted from publicly available servers, we build an RDF knowledge graph. We instantiate this Knowledge Graph on MCP-Atlas, a benchmark for tool-use competency built around real MCP servers, and compare a KG-augmented discovery workflow against a text-only baseline across multiple frontier models and two exposure regimes: the benchmark's smaller task-level tool menus and an overload setting with an all-tools registry of approximately 269 tools over 258 executed tasks. The early empirical results show specific and actionable insights. In smaller curated tool settings, direct text-only exposure remains stronger for all tested models. However, under overload where the unstructured baseline is constrained by a maximum tool budget, KG-based filtering improves GPT-5 from 0.478 to 0.542 mean coverage. For Claude 4.6 Sonnet in the all-tools condition, the KG retains roughly 89% of the text baseline's coverage while reducing the candidate set from about 270 tools to 4.6 tools on average. Qualitative error analysis indicates that the KG helps primarily by reducing tool overload, name ambiguity, and backend confusion, while its main weakness is incomplete recall caused by missing or imperfect capability assignments. The central conclusion validates the value of knowledge graphs as a structure-aware compression layer for large, noisy tool registries, and opens a larger research question on best approaches to represent tools-knowledge-graphs together with strong textual tool descriptions.

Keywords

knowledge graphs, tool discovery, MCP, agentic AI, ontology engineering, tool use, LLM agents

1. Introduction

Tool use has become a core capability of modern LLM agents [1, 2, 3]. Instead of answering from parametric memory alone, agents are designed to search documents, query databases, call APIs, read files, or compose multiple tools into a task-specific workflow that meets a target goal. An incipient research direction for tool use in agentic settings follows multi-tool orchestration over long trajectories [3] which in turn demands efficient discovery of the tools in use. In practice, however, tool interfaces are still mostly exposed as flat lists of names, descriptions, and JSON schemas. This is workable at small scale, but it becomes brittle once agents face tens or hundreds of overlapping, potentially ambiguous tools across multiple servers and domains. Concomitantly, Knowledge Graphs [4, 5] have been established as authoritative way to structure information in the enterprise and for consumption by generative models through ideas such as graph-retrieval-augmented generation (Graph-RAG) [6, 7] or as encoded graph tokens [8, 9, 10] imbibed into large language models, and several tasks have been evolved over the last decades concerning efficient graph representation.

The overarching research question in this work is to investigate whether a Knowledge Graph (KG) can serve as a better representation layer for tool discovery anchored on the central idea that tool selection is not only a lexical matching problem. It is also a structural reasoning problem involving

GENAIK-NORA 2026: Joint Workshop on Generative AI and Knowledge Graphs and Knowledge Graphs & Agentic Systems Interplay, IJCAI-ECAI 2026 Workshops, August 2026, Bremen, Germany

mualang.onando@sap.com (I. O. Mulang'); johannes.thaller@sap.com (J. Thaller); tushar.trivedi@sap.com (T. Trivedi); lars.heling@sap.com (L. Heling); felix.sasaki@sap.com (F. Sasaki)

capability hierarchies, server provenance, parameter semantics, and constraints on how tools can be used. A Knowledge Graph makes these structures explicit, supports transparent traversal and validation, and reduces the number of tools that must be surfaced to the downstream agent.

Although there are numerous ways to represent and serve tools for agentic use, we focus on the Model Control Protocol (MCP) [11, 12] because it has emerged as a practical interoperability layer for agentic tools and because MCP-Atlas provides a community-relevant benchmark built on real servers, controlled distractor tools, multi-step workflows, and claims-based evaluation [13]. Our ontology centers on four core classes: *Server*, *Tool*, *Capability*, and *Parameter*. We map MCP schemas into RDF triples, validate the graph with SPARQL, and use it as a discovery layer before execution. Our evaluation supports a more precise claim than "KGs outperform text." The current KG helps mainly in overload regimes, where the model must choose under too many tools, semantically overlapping descriptions, or provider-specific tool limits. In smaller curated settings, unstructured text remains better because it preserves recall and avoids an additional routing stage. This distinction matters for the GenAIK-NORA audience: the contribution is not only an ontology, but also an empirical characterization of when symbolic structure helps agentic tool use and when it does not. This paper makes four contributions:

1. A lightweight ontology for MCP-style tools that models servers, tools, capabilities, and parameters, while explicitly distinguishing required from optional inputs.
2. A KG construction workflow from real MCP tool schemas to RDF, with validation queries for graph integrity and relation traversal.
3. A comparative evaluation of KG-augmented tool discovery versus text-only tool exposure across multiple models and tool-budget regimes.
4. A fine-grained analysis showing that the KG's main benefit is structure-aware filtering under tool overload, whereas its main failure mode is recall loss from incomplete capability coverage.

2. Background and Related Work

2.1. Tool Schemas and Tool-Use Evaluation

Recent agent ecosystems have converged on schema-based tool descriptions, typically centered on JSON-style parameter specifications. Practitioner guidance from Anthropic emphasizes that effective tools need clear names, high-signal descriptions, token-efficient responses, and explicit evaluation, because agents are easily confused by overlapping or overly generic tool interfaces [14]. Likewise, tool-schema engineering guides describe JSON Schema as the de facto foundation for specifying name, description, parameter types, required fields, defaults, and constraints [15]. These sources are vital since our ontology is intentionally grounded in the fields that appear consistently across real tool schemas.

Benchmarking work has moved from isolated function-calling tasks toward more realistic agentic evaluation. Berkeley Function Calling Leaderboard (BFCL) evaluates tool and function calling performance across diverse scenarios and has evolved from function-call accuracy toward broader agentic evaluation [16]. ToolSandbox argues that realistic benchmarking requires stateful execution, implicit dependencies between tools, and conversational interaction [17]. Tool Playgrounds and StableToolBench likewise highlight the need for large-scale, analyzable, and stable evaluation environments for tool-using agents [18, 19]. Most directly relevant to this paper, MCP-Atlas evaluates tool-use competency with real MCP servers, multi-step workflows, controlled tool exposure, distractors, and claims-based scoring [13]. The public release contains 500 tasks, while the public leaderboard evaluates 1,000 tasks across 36 servers. Our evaluation sits within this broader trend, but asks a different question: whether structured tool representation changes which tools are discovered and therefore which tasks are solvable.

2.2. Semantic Service Discovery and Composition

The semantic-web community studied automated service discovery long before LLM agents. OWL-S and related work argued that machine-interpretable service descriptions are required for automatic

matching, composition, and invocation of services [20, 21]. Other work extended semantic matching with non-functional criteria such as quality-of-service and later adapted ontology-driven discovery to cloud services [22, 23]. These works are clear predecessors to tool discovery for LLM agents. However, classical semantic service discovery targeted web services and enterprise integration, not the interactive, prompt-driven, context-limited behavior of modern LLM agents. Our work revisits the same semantic discovery problem, but under a new operating constraint: agents must reason over tool interfaces using limited context windows and imperfect natural-language plans.

2.3. Knowledge Graphs for Agent and Tool Retrieval

Recent work has begun to combine graph-based retrieval with MCP-style agent ecosystems. Agent-as-a-Graph represents tools and parent agents as nodes in a knowledge graph and reports improvements in Recall@5 and nDCG@5 on a live MCP benchmark [24]. That work is closely aligned with the present paper, but focuses on graph-based retrieval for multi-agent systems rather than ontology design for tool schema semantics or a controlled comparison against direct text exposure. More broadly, graph-based reasoning has been used in domain-specific agent systems such as SciAgents, where ontological graphs help organize concepts and support multi-agent reasoning [25]. Our work differs in scope: it targets the representation and retrieval of executable tools themselves.

3. Problem Statement and Formalization

Let $\mathcal{S}$ be a set of servers, $\mathcal{T}$ a set of tools, $\mathcal{C}$ a set of capabilities, and $\mathcal{P}$ a set of parameters. We model the tool ecosystem as a typed directed graph:

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \tau_V, \tau_E),$$

where

$$\mathcal{V} = \mathcal{S} \cup \mathcal{T} \cup \mathcal{C} \cup \mathcal{P}$$

and τ_V and τ_E assign node and edge types. The graph contains at least the following edge families:

$$\begin{aligned} E_{\text{host}} &\subseteq \mathcal{T} \times \mathcal{S} \to \textit{hostedOn}, \\ E_{\text{cap}} &\subseteq \mathcal{T} \times \mathcal{C} \to \textit{hasCapability}, \\ E_{\text{req}} &\subseteq \mathcal{T} \times \mathcal{P} \to \textit{hasRequiredInput} \\ E_{\text{opt}} &\subseteq \mathcal{T} \times \mathcal{P} \to \textit{hasInput}, \\ E_{\text{sub}} &\subseteq \mathcal{C} \times \mathcal{C} \to\!\!\to \textit{capability-parent links}. \end{aligned}$$

Given a task instance $x \in \mathcal{X}$ with natural-language request $q(x)$, a discovery policy D returns a candidate set of tools

$$R_D(x) \subseteq \mathcal{T}, \quad |R_D(x)| \leq B,$$

where B is the tool budget that can be passed to the execution agent. An execution agent A then uses only the candidate set $R_D(x)$ to produce an answer $\hat{y}(x)$ and possibly a tool-use trace $\pi(x)$. Let $y^*(x)$ denote the reference answer and let

$$\text{Cov}(\hat{y}(x), y^*(x)) \in [0, 1]$$

be the benchmark coverage score used by the evaluation harness. The system objective is to maximize expected task coverage:

$$J(D, A) = \mathbb{E}_{x \sim \mathcal{X}} \left[\text{Cov}(\hat{y}(x), y^*(x)) \right].$$

This setup makes the core trade-off explicit. A text-only strategy D_{text} can have high recall because it exposes many or all tools directly, but it risks overload when B is large or when the model must reason over many overlapping descriptions. A KG-based strategy D_{kg} can reduce the candidate set by exploiting structure in $\mathcal{G}$, but it may fail if the graph omits relevant capabilities or routes the query to the wrong subgraph. To analyze discovery quality more directly, let $T^*(x) \subseteq \mathcal{T}$ denote the oracle set of tools sufficient for solving task x. Then discovery precision and recall are

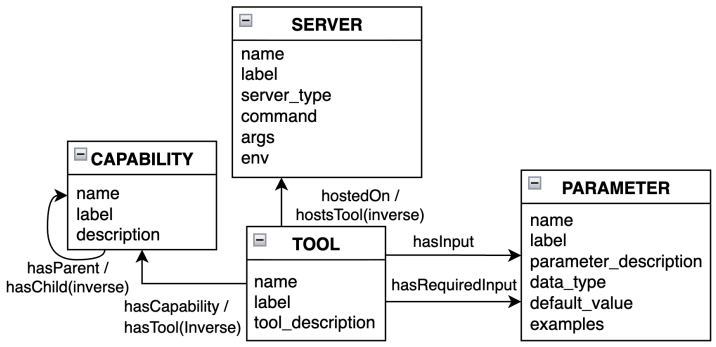


Figure 1: The Data Model: Implemented core ontology used in the KG prototype.

$$\mathrm{Prec}_D(x) = \frac{|R_D(x) \cap T^*(x)|}{|R_D(x)|}, \qquad \mathrm{Rec}_D(x) = \frac{|R_D(x) \cap T^*(x)|}{|T^*(x)|}.$$

The evidence in our experiments indicates that the current KG achieves very high precision but insufficient recall. This pos assists achieve good at filtering out irrelevant tools, but not yet good enough at recovering all relevant tools for hard multi-step tasks.

4. Ontology and Knowledge Graph Construction

4.1. Ontology Scope

We intentionally scope the ontology to the entities that matter most for discovery on MCP-style benchmarks. The current graph models four entity types:

- **Server**: the MCP server that hosts one or more tools.
- **Tool**: the main executable entity, described by a name, label, and natural-language description.
- **Capability**: an abstract functional class used for semantic grouping and retrieval.
- **Parameter**: reusable node describing an input argument with type and default value when available.

Figure 1 shows the implemented core class diagram. One noteworthy design choice is that requirement status is modeled relationally rather than as a Boolean attribute on the parameter alone. A parameter may be required for one tool and optional for another, so the distinction belongs naturally to the edge type: *hasRequiredInput* versus *hasInput*. This keeps the ontology closer to the semantics of invocation. The graph is tool-centric where the relation is modeled as *Tool* → *hostedOn* → *Server*, because the tool is the primary retrieval object and this direction eases extension to future non-MCP tool types such as REST APIs or local tools but we keep a reverse relation. Scope wise, tool outputs are excluded from the current core ontology. Whereas MCP input schemas are standardized, output schemas are often runtime-dependent or absent. Output modeling is therefore deferred to future work.

4.2. Capability Taxonomy

Capabilities are the semantic bridge between raw tool schemas and higher-level tool discovery. Tool selection is driven primarily by what a tool can accomplish, not by the surface form of its name. We therefore organize capabilities into a lightweight hierarchy that reflects recurring functional categories across contemporary tool ecosystems and benchmarks. The current taxonomy includes top-level clusters such as information access, content generation, data processing, and system interaction, with leaf capabilities including web search, database querying, external API access, text generation, code generation, image generation, information transformation, computation, file management, and shell execution. Figure 2 shows the capability taxonomy used for this clustering. This taxonomy serves two roles. First, it supports retrieval by grouping tools that are semantically related even when their

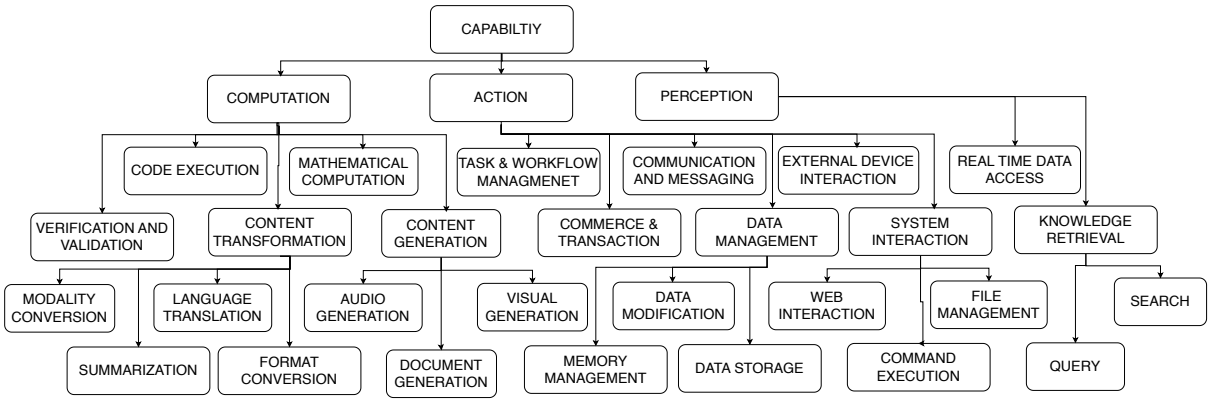


Figure 2: Capability taxonomy used to organize tool functionality into retrieval-relevant semantic classes.

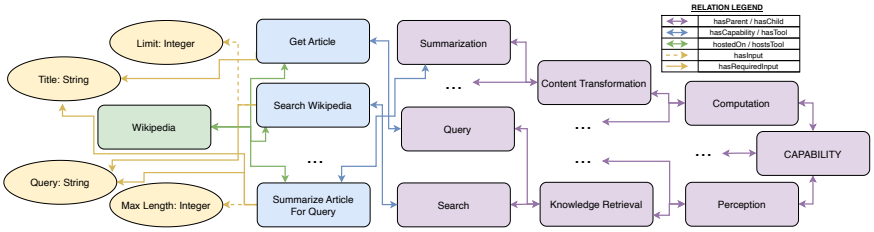


Figure 3: Instance-level validation of the ontology using Wikipedia MCP tools. The example highlights server, tool, capability, and parameter mappings.

names differ. Second, it provides an abstraction layer that can eventually align MCP-native tools with alternative representations such as OpenAPI or other agent-tool registries.

4.3. ETL Pipeline and Validation

The KG is built from real MCP tool schemas in the MCP-Atlas environment [13]. We initially validated the mapping on representative servers such as Wikipedia, whose tools include search, retrieval, summarization, and section-level operations. The same mapping procedure is then applied across benchmark servers and checked with SPARQL queries for relation traversal, inverse consistency, hierarchy traversal, and basic data integrity. The implemented mapping strategy is intentionally lightweight:

1. Extract raw MCP tool schemas from server manifests or JSON outputs.
2. Normalize tool names, descriptions, server metadata, and JSON-schema parameter fields.
3. Create parameter nodes with type and default metadata.
4. Attach parameters using either *hasRequiredInput* or *hasInput*.
5. Map each tool to one or more capabilities from the taxonomy.
6. Serialize the resulting graph to RDF and validate it using SPARQL.

In the current system, only tools with assigned capabilities are included in the KG. This choice improves precision, because unsupported tools are excluded, but it also creates a recall bottleneck whenever relevant tools have not yet been assigned a capability. Figure 3 shows an instance-level Mermaid rendering used to validate mappings for representative tools from the Wikipedia server.

5. KG-Augmented Tool Discovery Workflow

The KG-augmented workflow separates *discovery* from *execution*. Instead of presenting the downstream agent with all tool descriptions up front, the system first queries the graph to retrieve a small candidate

subset based on task semantics, capability associations, and structural metadata. The execution agent then operates over that reduced tool set. At a high level, the workflow is:

1. Parse the task request and identify likely capabilities, data sources, or action types.
2. Traverse the graph to retrieve tools linked to those capabilities and their hosting servers.
3. Inspect tool descriptions and parameter schemas for a small number of candidates.
4. Return the final candidate set to the execution agent.
5. Execute one or more MCP tools and produce the final answer.

This is qualitatively different from a text-only baseline that loads all available tools or a selected subset of them directly into the agent context. The KG workflow performs explicit candidate compression before the model commits to execution. In the all-tools regime, this reduces the average candidate set from roughly 270 tools to approximately 4.6 tools in the Claude 4.6 Sonnet setting. Such compression is valuable because practical tool use is constrained not only by retrieval quality but also by context limits and provider-specific caps on the number of tools that can be supplied at once.

6. Experimental Setup

6.1. Benchmark and Protocol

We evaluate on MCP-Atlas, a large-scale benchmark for tool-use competency with real MCP servers [13]. MCP-Atlas is designed for realistic agentic workflows rather than isolated function calls: tasks are written in natural language, typically require 3–6 tool calls, avoid naming the target tool directly, and are scored with a claims-based coverage metric. Each benchmark instance provides a prompt, a task-specific enabled tool menu, ground-truth claims, and a tool-call trajectory for diagnosis. The public release contains 500 tasks, while the leaderboard evaluates 1,000 tasks across 36 servers spanning search, analytics, productivity, finance, and coding. The reported runs use the MCP-Atlas environment and scoring methodology. Under the active server configuration for our experiments, the executable benchmark slice covers 258 tasks. In the overload regime, the accessible registry expands to approximately 269 tools, which is large enough to expose the retrieval bottleneck that this paper targets.

6.2. Compared Conditions

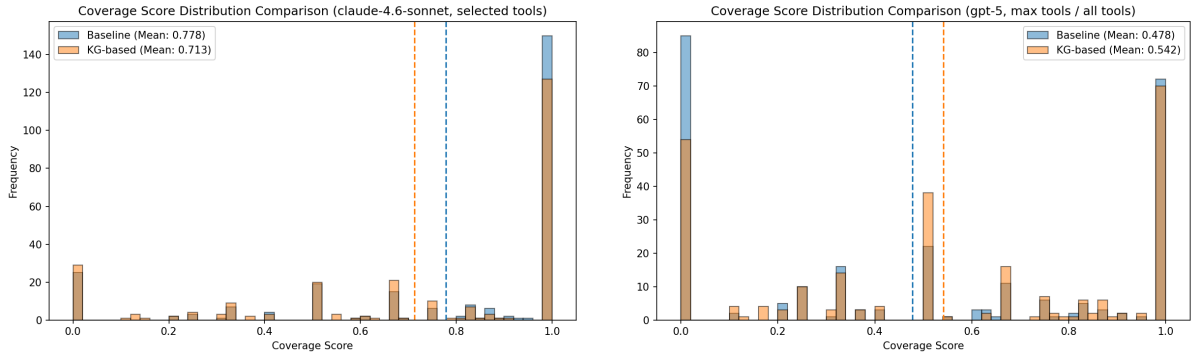
We compare two retrieval settings. **Selected-tools setting:** This setting is closest to the standard MCP-Atlas protocol. The execution agent receives a smaller task-level tool menu rather than the full registry. It tests whether KG mediation still helps when tool overload is already controlled.
All-tools setting: This is an additional stress test beyond the default benchmark configuration. The KG agent discovers tools from the full graph, while the text baseline is exposed to the full executable registry directly, subject to provider-specific limits. This regime stresses context saturation and retrieval ambiguity. GPT-5 is especially informative because the text baseline was capped at 128 tools, while the KG pipeline could still search over the full tool pool.

6.3. Models and Metric

The reported runs cover Claude 4.6 Sonnet, Claude 4.6 Opus, GPT-5, and Gemini 2.5 Pro. We report mean coverage on a $[0, 1]$ scale, because it is more sensitive than pass rate to partial task completion and near misses. MCP-Atlas defines a task as passed when coverage is at least 0.75, which is appropriate for leaderboard ranking; mean coverage is the more informative measure here because the intervention affects discovery quality before it changes binary task success. We also analyze task outcomes, tool-use traces, and candidate-set sizes in order to explain not only whether the KG helps, but why.

Table 1
Mean coverage scores for KG-based discovery versus text-only tool exposure. Positive Δ means the KG is better.

Model	Setting	Text	KG	Δ
Claude 4.6 Sonnet	Selected tools	0.778	0.713	-0.065
GPT-5	Selected tools	0.658	0.584	-0.074
Gemini 2.5 Pro	Selected tools	0.201	0.088	-0.113
Claude 4.6 Opus	Selected tools	0.871	0.814	-0.057
Claude 4.6 Sonnet	All tools	0.645	0.575	-0.070
GPT-5	All tools	0.478	0.542	+0.064
Gemini 2.5 Pro	All tools	0.152	0.102	-0.050
Claude 4.6 Opus	All tools	0.764	0.653	-0.111



(a) Coverage histogram comparison for Claude 4.6 Sonnet.

(b) Coverage histogram comparison for GPT-5 in the all-tools setting.

Figure 4: Task-level coverage distributions for the two most discussion-relevant experimental settings. Panel (a) shows Claude 4.6 Sonnet, while panel (b) shows GPT-5 under all-tools exposure.

7. Results

Table 1 reports the mean coverage scores from our MCP-Atlas runs, figure 4 shows the distribution of coverage on two models a(claude Sonnet) and b(GPT-5). The key observation is that the KG is *not* uniformly superior. In selected-tools setting, the text baseline outperforms the KG for every tested model suggesting that when the tool space has already been narrowed sufficiently, direct text descriptions remain easier for frontier models to exploit than an additional discovery layer. Secondly, for tool overload in the all-tools setting, GPT-5 improves under KG-based filtering, moving from 0.478 to 0.542 mean coverage. This indicates that a structured retrieval layer can recover useful tools more effectively than direct flat exposure in overloaded contexts. Finally, raw model capability remains a dominant factor. Claude 4.6 Opus is the strongest model in both settings, while Gemini 2.5 Pro performs poorly regardless, meaning that the KG should be is not a standalone substitute for strong tool reasoning.

The Claude 4.6 Sonnet all-tools condition is especially informative because task-level analysis is available for that run. In that setup, both approaches had access to approximately 270 tools, but the KG reduced the discovered set to about 4.6 tools on average. Despite this roughly 98% reduction in candidate tools, the KG still achieved 0.575 mean coverage versus 0.645 for the text baseline, or about 89% of the baseline's performance. This trade-off shows that structured filtering can preserve most of the baseline's effectiveness even while dramatically compressing the search space. For systems deployed with tight tool budgets, high latency, or a need for interpretable retrieval, that may be a worthwhile exchange even before the KG surpasses the text baseline outright.

7.1. Why and When the KG Helps

Task-level inspection points to three main scenarios in which the KG helps.

Table 2
Task-level outcome breakdown for the Claude 4.6 Sonnet all-tools condition.

Outcome	Main observations
KG wins (56 tasks, 22%)	Better backend selection under overload, fewer naming confusions, fewer failures from web-search blocking and security-rule issues.
Tie (119 tasks, 46%)	Mostly simpler single-tool tasks such as weather, museum search, or direct file reads where both approaches succeed.
Text wins (83 tasks, 32%)	Wrong data-source steering, asking the user for file paths instead of discovering them, upstream HTTP 500 errors, and schema mismatches.

Name and namespace ambiguity. When many tools overlap lexically, the text baseline can be misled by superficial name similarity or naming conventions. Capability-and server-aware retrieval helps by grouping tools semantically rather than relying only on surface form.

Backend overload and wrong-source confusion. In large tool pools, the text baseline sometimes chooses the wrong backend, for example using MongoDB where Airtable or Notion would be more appropriate. The KG helps by steering retrieval toward a capability-consistent region of the tool space.

7.2. Why the KG Still Loses in Many Cases

The same error analysis also identifies why the KG underperforms in the other conditions.

The Recall bottleneck: Discovered tools are almost always correct, with precision around 98% to 99.9% in the selected-tools setting, but recall is only around 25%. This means the KG often returns a clean but incomplete tool set.

Capability assignment errors steer the graph incorrectly. The most frequent issue in the Claude 4.6 Sonnet all-tools analysis was wrong data-source routing, especially when the graph steered tasks toward MongoDB or Airtable even though the task depended on local CSV files. This this indicates a failure of semantic coverage and routing quality in the current graph.

Some losses are implementation bugs rather than conceptual limits. Task traces expose fixable sources of failure such as a TwelveData parameter-wrapper mismatch, the wrong filesystem base path, and under-routing for Airtable. These issues likely understate the KG's eventual ceiling.

The divergent tasks are also qualitatively harder. They involve cross-domain tool chaining, file system exploration, computation, and multi-step planning. This matters because it suggests that the KG's missing recall is most costly on precisely the tasks where semantic abstraction should matter most.

8. Discussion

The results reveal a succinct interpretation of KG-based tool discovery beyond average-score comparison.

1.) The KG is best understood as a filtering mechanism, not yet a superior discovery oracle. The one clear positive result appears exactly in the regime where the text baseline is overloaded. This is consistent with both the GPT-5 all-tools result and the Claude Sonnet observation that the KG can compress roughly 270 tools to 4.6 candidates while preserving most of the baseline's task coverage.

2.) The main structural benefit is semantic factorization. A flat registry forces the agent to reason over tool names and descriptions directly. The KG introduces intermediate abstractions such as capabilities, parent-child relations, and server provenance. These abstractions reduce candidate entropy and make tool selection more interpretable. In effect, the KG transforms tool retrieval from "search the entire registry" into "enter the right semantic neighborhood, then inspect a few local candidates."

3.) High precision is not enough. A graph can still lose badly if it filters out a necessary tool. The current system already seems able to avoid many irrelevant candidates, but because the graph includes only capability-mapped tools and some mappings are incomplete, it often narrows the search space too aggressively. This is why the KG underperforms in curated settings, where the text baseline already enjoys manageable context and can afford broader exploration.

4.) Model behavior and graph quality interact strongly. In our runs, Claude Opus inspects the KG far more aggressively during discovery than Gemini, with roughly 25 times more detail-inspection calls in one comparison. This suggests that the KG is not a plug-in improvement that works identically for every model. A model must still know when to inspect, when to traverse, and when to stop filtering.

In summary, these findings imply that KGs are most promising when tool registries are large, heterogeneous, and semantically overlapping; when providers impose a hard limit on the number of tools that can be exposed; or when organizations need interpretable retrieval and provenance. They are less compelling on tasks with a small curated tool set or when capability coverage is incomplete.

9. Conclusion and Future Work

We presented a knowledge graph-centered approach to representing and retrieving agentic tools, instantiated on MCP-Atlas with a focus on MCP servers, and an additional overload condition derived from the same environment. The ontology models tools, servers, capabilities, and parameters, and uses the graph structure to prioritize discovery over full execution semantics. We omit output schemas, preconditions, side effects, or long-horizon composition patterns in the current model. This matters for the hardest cross-domain tasks, where success depends on more than choosing the right entry point.

Empirically, our tools knowledge graph does not universally outperform text descriptions for tool discovery, which remains an active research question for the community. This limitation can be attributed to current limited graph coverage as opposed to the KG formalization. Missing or imperfect capability assignment has an impact on the recall even when the retrieved tools are highly precise. However, we observe a more useful result: KG-based retrieval becomes valuable under tool overload, where flat textual exposure saturates the model's practical tool budget. This finding authoritatively indicates that this direction provides structure-aware compression, provenance, and controllable retrieval for large, noisy tool registries. Expanding and automatically maintaining capability coverage is therefore the main technical requirement for turning KG-based discovery from a useful filter into a consistently superior retrieval layer. Additionally, there is a need to investigate the best approaches to representing the tools KG for improved KG-based search of tools, then evaluate the same approach against stronger retrievers and larger live MCP registries.

Declaration on Generative AI

During the preparation of this manuscript, the authors used a generative AI system to help organize the paper structure and generate initial draft text. After using this tool or service, the authors reviewed, verified, and edited the content as needed and take full responsibility for the publication's content.

References

[1] G. Wölflein, D. Ferber, D. Truhn, O. Arandjelovic, J. N. Kather, LLM agents making agent tools, in: Proceedings of the 63rd ACL, Vienna, Austria, 2025. URL: https://aclanthology.org/2025.acl-long.1266/. doi:10.18653/v1/2025.acl-long.1266.

[2] X. Li, A review of prominent paradigms for LLM-based agents: Tool use, planning (including RAG), and feedback learning, in: Proceedings of the 31st International Conference on Computational Linguistics, ACL, Abu Dhabi, UAE, 2025. URL: https://aclanthology.org/2025.coling-main.652/.

[3] H. Xu, C. Li, X. Ma, X. Ou, Z. Zhang, T. He, X. Liu, Z. Wang, J. Liang, Z. Chu, R. Liu, R. Mu, D. Tu, M. Liu, B. Qin, The evolution of tool use in llm agents: From single-tool call to multi-tool orchestration, 2026. URL: https://arxiv.org/abs/2603.22862.

[4] C. Peng, F. Xia, M. Naseriparsa, F. Osborne, Knowledge graphs: Opportunities and challenges, Artif. Intell. Rev. (2023). URL: https://doi.org/10.1007/s10462-023-10465-9.

[5] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. Ives, Dbpedia: a nucleus for a web of open data, in: Proceedings of the 6th International The Semantic Web and 2nd Asian Conference on Asian Semantic Web Conference, ISWC'07/ASWC'07, Springer-Verlag, Berlin, Heidelberg, 2007.

[6] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-augmented generation for knowledge-intensive nlp tasks, NIPS '20, Curran Associates Inc., Red Hook, NY, USA, ????

[7] B. Peng, Y. Zhu, Y. Liu, X. Bo, H. Shi, C. Hong, Y. Zhang, S. Tang, Graph retrieval-augmented generation: A survey (2025). URL: https://doi.org/10.1145/3777378. doi:10.1145/3777378.

[8] B. Fatemi, J. Halcrow, B. Perozzi, Talk like a graph: Encoding graphs for large language models, in: The Twelfth International Conference on Learning Representations, 2024. URL: https://openreview.net/forum?id=IuXR1CCrSi.

[9] B. Perozzi, B. Fatemi, D. Zelle, A. Tsitsulin, M. Kazemi, R. Al-Rfou, J. J. Halcrow, Let your graph do the talking: Encoding structured data for llms, ArXiv abs/2402.05862 (2024). URL: https://api.semanticscholar.org/CorpusID:267547812.

[10] Z. Hu, Y. Li, Z. Chen, J. Wang, H. Liu, K. Lee, K. Ding, Let's ask GNN: Empowering large language model for graph in-context learning, in: Findings of the Association for Computational Linguistics: EMNLP, ACL, Miami, Florida, USA, 2024. URL: https://aclanthology.org/2024.findings-emnlp.75/.

[11] Anthropic, Introducing the model context protocol, Engineering blog, 2024. URL: https://www.anthropic.com/news/model-context-protocol, [Accessed 06-05-2026].

[12] X. Hou, Y. Zhao, S. Wang, H. Wang, Model context protocol (mcp): Landscape, security threats, and future research directions, ACM Trans. Softw. Eng. Methodol. (2026). URL: https://doi.org/10.1145/3796519. doi:10.1145/3796519.

[13] Scale AI, MCP-Atlas: A large-scale benchmark for tool-use competency with real mcp servers, 2026. URL: https://github.com/scaleapi/mcp-atlas.

[14] Anthropic, Writing effective tools for agents - with agents, Engineering blog, 2025. URL: https://www.anthropic.com/engineering/writing-tools-for-agents, [Accessed: 2026-05-03].

[15] N. Dhandala, How to implement tool schemas, OneUptime blog, 2026. URL: https://oneuptime.com/blog/post/2026-01-30-tool-schemas/view, accessed: 2026-05-03.

[16] S. G. Patil, H. Mao, C. C.-J. Ji, F. Yan, V. Suresh, I. Stoica, J. E. Gonzalez, The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models, in: Proceedings of the Forty-second International Conference on Machine Learning, 2025.

[17] J. Lu, T. Holleis, Y. Zhang, B. Aumayer, N. Feng, H. Bai, S. Ma, S. Ma, M. Li, G. Yin, Z. Wang, R. Pang, ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities, in: Findings of the Association for Computational Linguistics: NAACL 2025, 2025. URL: https://arxiv.org/abs/2408.04682. doi:10.18653/v1/2025.findings-naacl.65.

[18] Z. Dong, R. Gong, Y. Yang, S. Wu, Y. Yao, S.-L. Chen, X.-C. Yin, Tool playgrounds: A comprehensive and analyzable benchmark for LLM tool invocation, in: ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing, 2025. doi:10.1109/ICASSP49660.2025.10890828.

[19] Z. Guo, S. Cheng, H. Wang, S. Liang, Y. Qin, P. Li, Z. Liu, M. Sun, Y. Liu, StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models, in: Findings of the ACL, 2024. doi:10.18653/v1/2024.findings-acl.664.

[20] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, K. Sycara, Bringing semantics to web services with OWL-S, World Wide Web (2007). doi:10.1007/s11280-007-0033-x.

[21] U. Keller, R. Lara, A. Polleres, I. Toma, H. Lausen, Ranked matching for service descriptions using OWL-S, in: Technologies for E-Services, 2005. doi:10.1007/3-540-27301-8_8.

[22] Q. Z. Sheng, B. Benatallah, Z. Maamar, A. H. H. Ngu, M. Dumas, Semantic QoS metric matching,

in: Proceedings of the IEEE European Conference on Web Services, 2006. doi:10.1109/ECOWS.2006.34.

[23] M. Parhi, B. K. Pattanayak, M. R. Patra, An ontology-based cloud infrastructure service discovery and selection system, International Journal of Grid and Utility Computing (2018). doi:10.1504/IJGUC.2018.091715.

[24] F. Nizar, E. Lumer, A. Gulati, P. H. Basavaraju, V. K. Subbiah, Agent-as-a-graph: Knowledge graph-based tool and agent retrieval for LLM multi-agent systems, arXiv preprint arXiv:2511.18194, 2025. URL: https://arxiv.org/abs/2511.18194. doi:10.48550/arXiv.2511.18194.

[25] A. Ghafarollahi, M. J. Buehler, SciAgents: Automating scientific discovery through bioinspired multi-agent intelligent graph reasoning, Advanced Materials (2024). doi:10.1002/adma.202413523.