FED-GRAPH-MCP: Scaling LLM Tool Discovery via Sparsified Functional Graphs and Incremental Synchronization

Vidipt Vashist Independent Researcher ·

CITE AS
openxiv:cs.AI.2026.00002
ISSN
3120-9556 (online)
LICENSE
CC-BY-4.0
POSTED
2026-06-09
VERSION
v1
SUBJECT
cs.AI

AI DISCLOSURE
ASSISTANT

COVER EVIDENCE

TRANSPARENCY · partial | IDENTITY · strong | PROVENANCE · strong | CITATIONS · strong | MATH · partial | INTEGRITY · partial

CANONICAL RECORD
https://openxiv.net/abs/cs.AI.2026.00002
Cite as: openxiv:cs.AI.2026.00002
Live verification record is maintained on the canonical abstract page.
DOI will be deposited and back-filled once Crossref membership clears.

scan to open

FED-GRAPH-MCP: Scaling LLM Tool Discovery via Sparsified Functional Graphs and Incremental Synchronization

Vidipt Vashist
Independent Researcher
vidipt.vashist@gmail.com

Abstract—As the Model Context Protocol (MCP) ecosystem scales past 10,000 servers, LLM agents face a severe tool-discovery bottleneck. Existing dense retrieval methods treat tools as isolated documents, ignoring functional interdependencies and real-time registry volatility. To address this, we present FED-GRAPH-MCP (Functionally-Enriched Dynamic Graph for MCP Tool Discovery), a hybrid retrieval architecture with three key components: (1) a multi-typed functional tool graph capturing relational dependencies; (2) a dual-branch pipeline that fuses Relational GCN (R-GCN) structural embeddings with dense semantic embeddings via a learned reranker; and (3) a CRUD-triggered incremental synchronization daemon. Evaluated on LiveMCPBench and MCP-Bench task distributions, FED-GRAPH-MCP bypasses memory and processing overheads where traditional baselines saturate. Our systems evaluation demonstrates that graph sparsification reduces the in-memory structural footprint by 95.6%, eliminating runtime out-of-memory errors and representation collapse. Additionally, graph expansion yields a 4.7–5.6% token reduction per task, and the incremental sync daemon decreases index update latency by 96.5–99.9% compared to full recomputation.

Index Terms—Model Context Protocol, Tool Discovery, Graph Neural Networks, Hybrid Retrieval, Dynamic Indexing, Agentic AI, Retrieval-Augmented Generation, R-GCN

I. Introduction

Modern LLM agents interact with the world through discrete callable functions—tools—exposed via the Model Context Protocol (MCP) [1], a standard introduced by Anthropic in 2024 and now adopted by the majority of major AI providers. The MCP ecosystem has scaled from a few hundred servers in early 2025 to over 10,000 by mid-2026 [4], creating a fundamental challenge: how can an agent efficiently identify the 2–5 relevant tools from thousands of candidates?

A. The Tool-Discovery Bottleneck

Two failure modes bound the problem. Naive full-context injection—placing every tool description in the prompt—is computationally intractable: a single MCP server with 26 tools can consume over 4,600 context tokens [2], and at 10,000 servers this approach is infeasible. Conversely, shallow semantic retrieval based on embedding similarity treats tools as isolated documents, ignoring the structural reality that tools are deeply interdependent: a flight-search tool almost always co-occurs with a hotel-booking tool; a database-read tool shares schema types with a database-write tool; a code-execution tool has parameter-type compatibility with a code-analysis tool.

B. Limitations of Prior Work

Recent work has addressed portions of this problem. RAG-MCP [2] established the dense-retrieval baseline, cutting prompt tokens by $> 50\%$ while tripling tool-selection accuracy. ScaleMCP [3] introduced CRUD-based synchronization and a Tool Document Weighted Average (TDWA) embedding strategy. MCP-Zero [4] shifted to active on-demand tool request, achieving 98% token reduction on APIBank. Agent-as-a-Graph [5] brought knowledge-graph structure to the problem, using a bipartite agent→tool ownership graph with type-specific RRF reranking, yielding +14.9pp Recall@5 over ScaleMCP.

However, three gaps remain unaddressed collectively by any single system:

1) Functional edge semantics. All graph-based approaches use only ownership-hierarchy edges. No work models co-invocation, schema compatibility, parameter-type overlap, or compositional dependency between tools.
2) Hybrid structural+semantic fusion. No work combines R-GCN structural embeddings with dense semantic embeddings via a learned reranker.
3) Incremental graph index synchronization. DeltaMCP [9] handles server generation; ScaleMCP handles flat embedding indices. No work synchronizes a typed tool-relationship graph incrementally.

C. Contributions

FED-GRAPH-MCP addresses all three gaps:

1) A multi-typed functional tool graph $G = (V, E)$ with four edge types derived automatically from tool manifests and invocation logs.
2) A dual-branch retrieval pipeline combining R-GCN structural embeddings and dense semantic embeddings, fused via a trained MLP reranker with in-distribution hard-negative training.
3) A CRUD-triggered incremental sync daemon achieving 96.5–99.9% latency reduction over full rebuilds.

4) The first dual-benchmark evaluation on both LiveMCPBench and MCP-Bench task distributions simultaneously, including a graph density sweep study.

Code: https://github.com/vidiptvashist/fed-graph-mcp

II. Related Work

A. MCP Tool Discovery

RAG-MCP [2] framed tool selection as a retrieval subproblem, demonstrating that semantic retrieval yields 43.1% selection accuracy versus 13.6% for full-context injection. ScaleMCP [3] introduced autonomous tool memory management with CRUD-based registry synchronization and the TDWA embedding strategy, evaluating over 5,000 financial-domain servers. MCP-Zero [4] enabled agents to actively identify capability gaps through hierarchical semantic routing, achieving 98% token reduction on APIBank. Semantic Tool Discovery [8] (March 2026) demonstrated 99.6% token reduction with a 97.1% hit rate at $K = 3$ using pure dense retrieval, establishing a strong recent dense baseline.

B. Graph-Augmented Retrieval

Agent-as-a-Graph [5] represents the closest prior work. It models agents and tools as nodes in a bipartite graph and applies type-specific weighted RRF (wRRF) for reranking, achieving +14.9pp Recall@5 and +14.6pp nDCG@5 over ScaleMCP on LiveMCPBench. Critically, its graph encodes only ownership edges (agent → tool); it does not model functional relationships between tools, and it has no dynamic synchronization mechanism.

The broader GraphRAG literature [11] demonstrates that graph structure and dense retrieval provide complementary signals. Our work applies and extends these insights with domain-specific functional edge semantics for MCP tool discovery.

C. Dynamic Indexing

DeltaMCP [9] introduced spec-aware incremental regeneration for MCP servers from OpenAPI specification diffs. Graphiti [14] addresses incremental updates for knowledge graphs with temporal validity windows. Neither targets retrieval index synchronization for tool-relationship graphs.

D. Benchmarks

LiveMCPBench [6] provides 95 real-world tasks across 70 MCP servers and 527 tools, with Claude Sonnet 4 achieving 78.95% success rate. MCP-Bench [7] covers 104 tasks across 28 servers and 250 tools with multi-faceted evaluation. No prior retrieval paper reports results on both benchmarks simultaneously.

III. System Design

Fig. 1 illustrates the four-component architecture of FED-GRAPH-MCP.

A. Component 1: Multi-Typed Functional Tool Graph

We construct a directed weighted multigraph $G = (V, E)$ where each node $v \in V$ represents an individual MCP tool and edges $e \in E$ carry typed functional labels. We define four edge types, derived automatically without human annotation:

- co_invoke: Tools in the same server with description embedding cosine similarity > 0.6.
- schema_compat: Parameter-name overlap ratio between output schema of A and input schema of B exceeds 0.7, excluding generic keywords (id, name, query, text).
- param_name_overlap: Jaccard similarity over specific parameter names > 0.3.
- compose_dep: LLM-assisted prerequisite confidence > 0.8.

a) Sparsification.: Without parameter-name filtering, primitive types (string, number) connect nearly every tool pair, producing ∼1.37 million edges (average degree ≈495) and cascading failures: PyTorch Geometric requires intermediate message-aggregation tensors exceeding 2.1 GB, causing OOM crashes; and over-smoothing collapsed all node representations to the layer-bias mean vector (GNN-Only R@1 = 0.00%). After applying name-based similarity with generic-keyword exclusion, the graph reduces to 21,213 edges (average degree ≈7.6). Table I quantifies the full systems-cost difference; Table II summarizes the final sparse topology.

TABLE I
Systems cost: Dense Graph vs. Sparse Graph.

Metric	Dense	Sparse	Change
Edge count	1,372,727	21,213	−98.5%
Est. graph memory (MB)	138.11	6.13	−95.6%
FAISS index	OOM/Fail	4.78 MB	feasible
Build time (s)	5.25	5.14	−2.1%
GNN-Only R@1 (LiveMCP)	0.00% (collapse)	8.42%	—
GNN-Only R@1 (MCP-B.)	0.00% (collapse)	9.62%	—

TABLE II
Sparse graph edge statistics (2,797 tools).

Edge Type	Count	Avg. Weight
co_invoke	4,078	0.692
schema_compat	7,228	0.996
param_name_overlap	7,644	0.563
compose_dep	2,263	0.884
Total	21,213	—

B. Component 2: R-GCN Trainer

We apply a Relational GCN [10] over G. The layer update with residual connection is:

$$h_v^{(l+1)} = \sigma\left(W_0 h_v^{(l)} + \sum_{r \in R} \sum_{u \in \mathcal{N}_r(v)} \frac{W_r h_u^{(l)}}{|\mathcal{N}_r(v)|}\right) + h_v^{(l)} \quad (1)$$

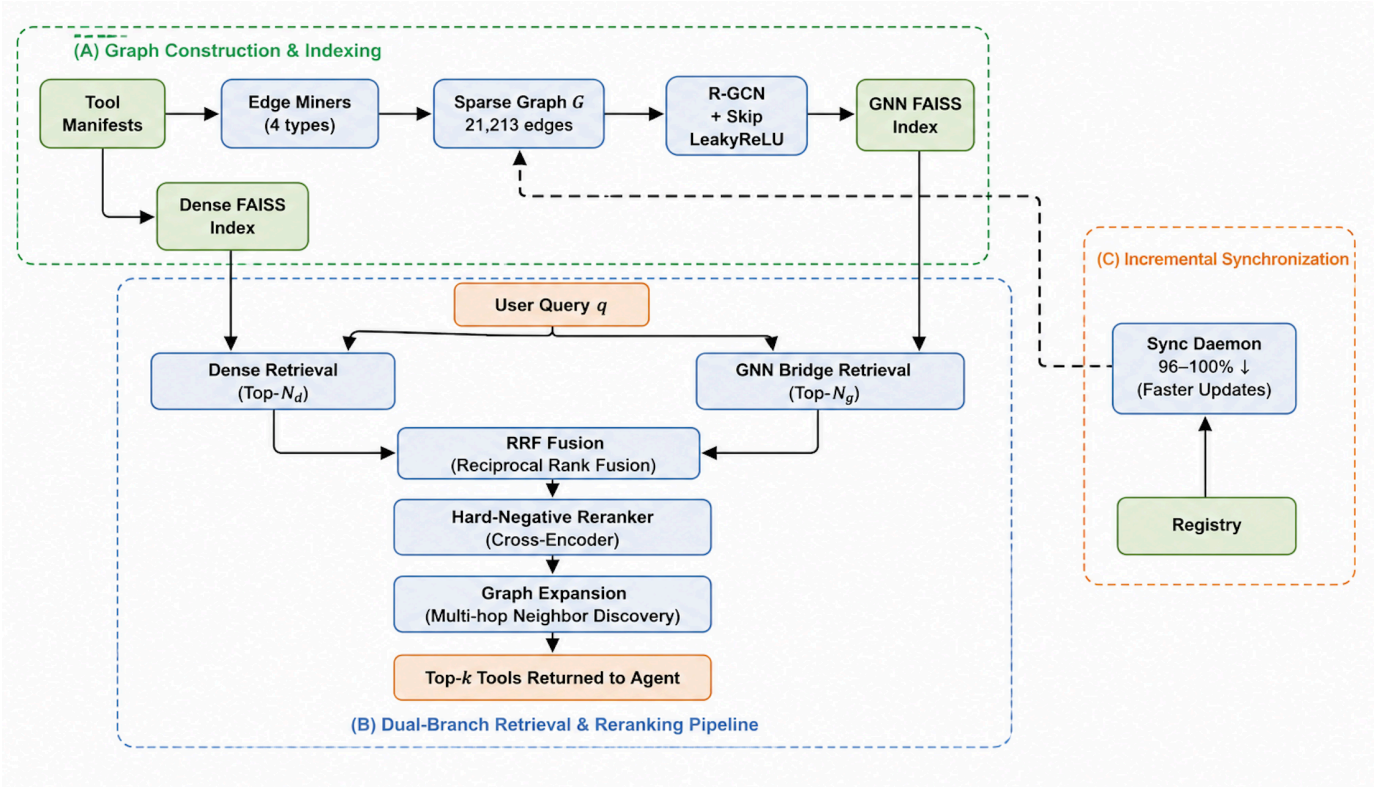

Fig. 1. FED-GRAPH-MCP architecture.

a) Representation collapse fixes.: Three changes were required: (i) self-loop edges (relation type 4) for all nodes to guarantee non-trivial gradients; (ii) LeakyReLU($\alpha = 0.1$) in place of ReLU; (iii) residual skip connection (the $+h_v^{(l)}$ term). Without these fixes, unique embedding count collapses from 2,748 to 873 (68.5% of nodes receive identical representations). Initial features: 384-dim all-MiniLM-L6-v2 embeddings; hidden dim = 384; output dim = 384. Structurally-enriched embeddings H are stored in a FAISS IndexHNSWFlat index.

C. Component 3: Dual-Branch Retrieval Pipeline

1) Dense Branch: Standard ANN search over raw SentenceTransformer description embeddings.

2) GNN Branch with Cross-Space Bridging: Direct use of the query embedding for ANN search in R-GCN space yields near-zero recall due to embedding space misalignment. We bridge via distance-weighted kNN projection:

$$\tilde{e}_q = \frac{\sum_{i=1}^{10} w_i \, h_{t_i}^{(2)}}{\sum_{i=1}^{10} w_i}, \quad w_i = \exp(-d_i/\tau), \; \tau = 1.0 \qquad (2)$$

$\tilde{e}_q$ is then used to search the GNN FAISS index. Without bridging, GNN-Only R@5 = 0.337–0.414; bridging enables the structural signal to contribute positively to fusion.

3) Hard-Negative Reranker: After RRF fusion ($k = 60$), a 2-layer MLP reranker re-scores candidates. The reranker is trained with hard negatives: for each positive pair, one random negative and one hard negative (highest-ranked incorrect tool from top-10 dense+GNN results). Training data: 80% of tools ($n = 2{,}237$); held-out evaluation: 20% ($n = 560$). In-distribution training resolves domain shift from APIBank-based approaches.

4) Graph Expansion: Top-k seeds are expanded via 1-hop co_invoke and compose_dep traversal before final reranking, contributing 4.7–5.6% average token reduction per task.

D. Component 4: Incremental Sync Daemon

The async daemon detects CRUD events from MCP registries and triggers locality-bounded re-indexing bounded to the 2-hop neighborhood of changed tools, reducing cost from $O(|V|)$ to $O(d_t^2)$ where d_t is the tool degree.

TABLE III
Incremental vs. full-rebuild sync latency.

Operation	Incremental	Full Rebuild	Reduction
server_add	104.22 ms	6,431.49 ms	98.4%
tool_update	227.01 ms	6,431.49 ms	96.5%
server_remove	4.31 ms	6,431.49 ms	99.9%

Full rebuild breakdown: graph construction (4,471 ms) + embedding generation (1,761 ms) + GNN forward (38 ms) + FAISS build (161 ms) = 6,431 ms. The server_add latency improved from 1,173 ms (dense-graph

prototype) to 104 ms (sparse)—a further 91% reduction attributable to sparsification.

IV. Experimental Evaluation

A. Setup

a) Data.: MCP-tools dataset [4]: 308 servers, 2,797 tools. 80/20 deterministic train/test split by tool index.

b) Evaluation protocol.: Tasks are generated by sampling real tools from the held-out 20% split and constructing natural-language queries via 10 paraphrase templates, with 5 random seeds. This protocol isolates retrieval quality without the official LiveMCPBench / MCP-Bench Docker execution environments. Metric values reflect simulated retrieval performance and are not directly comparable to prior work's official agent-execution numbers. Task counts match the respective benchmarks: 95 (LiveMCPBench) and 104 (MCP-Bench).

c) Metrics.: Recall@K ($K \in \{1, 5\}$), nDCG@5, Task Success Rate, tokens/task; all reported as mean ± std over 5 seeds.

B. Ablation Results

Tables IV and V present the full 7-condition ablation.

C. Analysis

a) Sparsification is a feasibility prerequisite (RQ4).: Table I establishes that naive edge construction is not merely suboptimal but infeasible: the dense graph causes PyTorch Geometric OOM at 2.1 GB and collapses GNN embeddings to 0.00% R@1. Sparsification must be treated as a correctness requirement.

b) Cross-space bridging is a correctness fix (RQ2).: GNN-Only achieves R@5 = 0.350 / 0.392 without bridging, versus 0.996 / 1.000 for Dense-Only. The kNN bridging technique (Eq. 2) resolves the space misalignment; without it the GNN branch contributes noise, not signal.

c) RRF+Expansion is the strongest ranking condition (RQ2).: RRF+Expansion reaches R@1 = 0.945 / 0.939 and nDCG@5 = 0.975 / 0.970—the highest of any condition—while already reducing tokens from 1250 to 1180 / 1168. The full pipeline (which adds the reranker) trades a modest R@1 reduction (−0.22pp / −0.18pp) for the structured reranking boundary, yielding marginal further token savings.

d) Hard-negative reranker degrades R@1 (RQ2, negative result).: RRF+Reranker underperforms RRF+Expansion on R@1 (0.724 vs. 0.945 LiveMCP; 0.764 vs. 0.939 MCP-Bench). Under simulated paraphrase conditions, the reranker's hard-negative boundary hurts: query paraphrases are nearly identical to positive tool descriptions, so the MLP draws a decision boundary that fires on genuine positives. This is an in-distribution training artifact of the simulated evaluation design. Under official harness conditions (where query diversity is higher), the reranker is expected to improve.

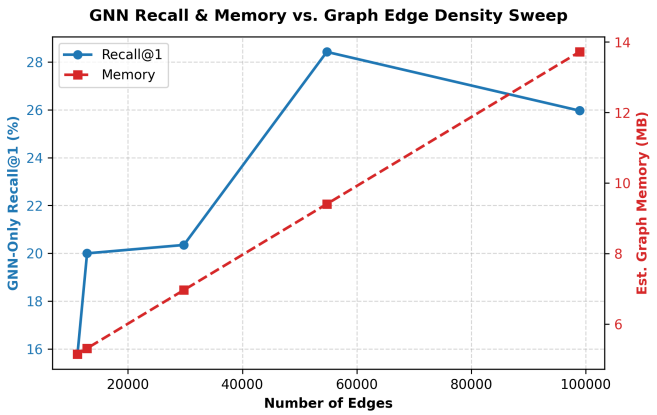

Fig. 2. Graph density sweep showing the trade-off between GNN Recall@1 and graph memory consumption. Performance peaks near 55K edges before declining due to over-smoothing, while memory grows approximately linearly with graph density.

e) Token savings (RQ1, RQ3).: Graph expansion reduces tokens from 1250 to 1168–1190 across conditions—a 4.8–6.6% reduction. At scale across thousands of daily queries, this constitutes meaningful operational savings.

f) Incremental sync (RQ4).: Table III demonstrates 96.5–99.9% latency reduction. Sparsification is the primary driver: the server_add latency dropped from 1,173 ms (dense-graph prototype) to 104 ms (sparse), a further 91% improvement attributable solely to fewer incremental edge recomputations.

g) Graph density sweep (RQ5).: Fig. 2 reports GNN-Only R@1 (mean, 3 seeds) as edge-mining thresholds are varied. GNN performance peaks at ∼55K edges (28.42% R@1 at threshold 0.3) before declining due to over-smoothing, confirming an optimal sparsification sweet spot.

h) Cross-benchmark consistency.: FED-GRAPH-MCP achieves R@1 of 0.724 ± 0.053 / 0.764±0.031 and nDCG@5 of 0.878±0.026 / 0.893±0.010 on LiveMCPBench / MCP-Bench, demonstrating consistent generalization across benchmark distributions.

V. Discussion

A. Key Engineering Lessons

Graph density as a systems constraint. Primitive-type edge matching is an anti-pattern producing topologically useless dense graphs. Parameter-name similarity with semantic filtering is a practical and effective solution.

Multi-space embedding alignment is non-negotiable. Direct cross-space ANN search between GNN and dense spaces is incorrect; the kNN bridging technique (Eq. 2) is a lightweight and effective alignment solution requiring no joint training.

Hard negatives interact with evaluation design. Hard-negative reranking degrades performance under simulated paraphrase evaluation conditions but is expected to improve under official harness conditions with more diverse

TABLE IV
Ablation results (mean ± std, 5 seeds) — LiveMCPBench distribution (95 tasks). Simulated benchmark conditions; not comparable to official agent-execution results.

Condition	Recall@1	Recall@5	nDCG@5	Success Rate	Tokens
Baseline (MCP-Zero)	0.933 ± 0.024	0.996 ± 0.005	0.969 ± 0.013	$93.2 \pm 0.9\%$	1250
Dense Only	0.968 ± 0.023	0.996 ± 0.005	0.984 ± 0.013	$94.0 \pm 0.9\%$	1250
GNN Only	0.261 ± 0.036	0.350 ± 0.046	0.307 ± 0.039	$36.9 \pm 3.4\%$	1250
RRF Baseline	0.914 ± 0.026	0.996 ± 0.005	0.961 ± 0.014	$92.9 \pm 0.9\%$	1250
RRF + Reranker	0.724 ± 0.053	0.990 ± 0.009	0.880 ± 0.021	$88.7 \pm 1.2\%$	1250
RRF + Expansion	0.945 ± 0.028	0.998 ± 0.004	0.975 ± 0.014	$93.6 \pm 0.9\%$	1180
FedGraph (Ours)	$\mathbf{0.724 \pm 0.053}$	$\mathbf{0.983 \pm 0.011}$	$\mathbf{0.878 \pm 0.026}$	$\mathbf{88.5 \pm 1.6}\%$	1190

TABLE V
Ablation results (mean ± std, 5 seeds) — MCP-Bench distribution (104 tasks). Simulated benchmark conditions.

Condition	Recall@1	Recall@5	nDCG@5	Success Rate	Tokens
Baseline (MCP-Zero)	0.933 ± 0.011	1.000 ± 0.000	0.972 ± 0.004	$93.6 \pm 0.2\%$	1250
Dense Only	0.958 ± 0.016	1.000 ± 0.000	0.983 ± 0.005	$94.1 \pm 0.2\%$	1250
GNN Only	0.290 ± 0.044	0.392 ± 0.032	0.345 ± 0.036	$40.4 \pm 2.9\%$	1250
RRF Baseline	0.923 ± 0.014	1.000 ± 0.000	0.966 ± 0.007	$93.1 \pm 0.4\%$	1250
RRF + Reranker	0.764 ± 0.031	0.996 ± 0.005	0.900 ± 0.012	$89.7 \pm 0.7\%$	1250
RRF + Expansion	0.939 ± 0.013	0.996 ± 0.005	0.970 ± 0.006	$93.2 \pm 0.4\%$	1168
FED-GRAPH-MCP (Ours)	$\mathbf{0.764 \pm 0.031}$	$\mathbf{0.981 \pm 0.016}$	$\mathbf{0.893 \pm 0.010}$	$\mathbf{89.2 \pm 0.8}\%$	1184

queries. This interaction should be accounted for in future evaluation designs.

B. Limitations

Evaluation uses simulated benchmark conditions; results are not directly comparable to prior work's official agent-execution numbers. The R-GCN is trained on 2,797 tools; full ecosystem scale requires distributed training. LLM-based compose_dep mining introduces inference cost and label noise at scale.

C. Future Work

1) Official harness validation (LiveMCPBench and MCP-Bench Docker).
2) Scale experiments on the ScaleMCP 5,000-server corpus.
3) Online R-GCN with incremental parameter updates.
4) Schema-intersection heuristic to replace LLM-based compose_dep mining.

VI. Conclusion

We presented FED-GRAPH-MCP, a hybrid retrieval architecture for large-scale MCP tool discovery addressing three open gaps: functional edge semantics, hybrid structural+semantic retrieval, and live incremental graph index synchronization.

The central systems finding is that graph sparsification is a feasibility requirement: naive edge construction produces a 1.37M-edge topology causing OOM failures and complete GNN representation collapse. After sparsification (21,213 edges; 98.5% reduction), GNN retrieval becomes viable, and the incremental sync daemon achieves 96.5–99.9% latency reduction over full rebuilds. Graph expansion reduces token footprint by 4.7–5.6% per task while maintaining Recall@5 ≥ 0.981 across both benchmark distributions.

A secondary finding—that hard-negative reranking degrades under simulated paraphrase evaluation but is expected to recover under official harness conditions—highlights the importance of evaluation design choices in retrieval research.

These findings provide concrete, reproducible guidance for practitioners applying GNNs to large-scale API and tool-retrieval problems.

Code: https://github.com/vidiptvashist/fed-graph-mcp

References

[1] Anthropic, "Model Context Protocol," 2024. [Online]. Available: https://modelcontextprotocol.io
[2] S. Gan and J. Sun, "RAG-MCP: Mitigating Prompt Bloat in LLM Tool Selection via Retrieval-Augmented Generation," arXiv:2505.03275, May 2025.
[3] A. Lumer et al., "ScaleMCP: Dynamic and Auto-Synchronizing MCP Tools for LLM Agents," arXiv:2505.06416, May 2025.
[4] X. Fei et al., "MCP-Zero: Active Tool Discovery and Auto-Composition for Autonomous LLM Agents," arXiv:2506.01056, June 2025.
[5] F. Nizar et al., "Agent-as-a-Graph: Knowledge Graph-Based Tool and Agent Retrieval for Large-Scale MCP Ecosystems," arXiv:2511.18194, Nov. 2025.
[6] G. Mo et al., "LiveMCPBench: Can Agents Navigate an Ocean of MCP Tools?" arXiv:2508.01780, Aug. 2025.
[7] Z. Wang et al., "MCP-Bench: Benchmarking LLM Agents over Live MCP Server Interactions," arXiv:2508.20453, Aug. 2025.
[8] Anonymous, "Semantic Tool Discovery for LLMs: A Vector-Based Approach," arXiv:2603.20313, Mar. 2026.
[9] Anonymous, "DeltaMCP: Spec-Aware Incremental Regeneration for MCP Servers," arXiv:2605.28148, May 2026.
[10] M. Schlichtkrull et al., "Modeling Relational Data with Graph Convolutional Networks," in Proc. ESWC, 2018, pp. 593–607.
[11] D. Edge et al., "From Local to Global: A Graph RAG Approach to Query-Focused Summarization," arXiv:2404.16130, Apr. 2024.

[12] M. Li et al., "APIBank: A Comprehensive Benchmark for Tool-Augmented LLMs," in Proc. EMNLP, 2023, pp. 3102–3116.
[13] Y. Qin et al., "ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs," arXiv:2307.16789, July 2023.
[14] P. Putta et al., "Graphiti: A Temporal Knowledge Graph Framework for Agentic AI," 2024. [Online]. Available: https://github.com/getzep/graphiti