Scaling LLM Agent Tool Access at Cloud Scale

Mingxin Li[†‡], Enge Song[‡], Yueshang Zuo[‡], Xiaodong Liu[‡], Rong Wen[§‡], Jian He[‡], Jing Tie[‡], Zhou Shao[‡], Qiang Fu[¶], Xiaobo Xue[‡], Xiong Xiao[‡], Luyao Zhong[‡], Shaokai Zhang[‡], Jiangu Zhao[‡], Jianyuan Lu[‡], Shize Zhang[‡], Xiaoqing Sun[‡], Changgang Zheng[‡], Zihao Fan[‡], Haonan Li[‡], Tian Pan[‡], Xiaomin Wu[‡], Yang Song[‡], Xing Li[‖‡], Biao Lyu[‡], Meng Li[†], Haipeng Dai[†], Guihai Chen[†], Shunmin Zhu[‡]

[†]Nanjing University [‡]Alibaba Cloud [§]Fudan University [¶]RMIT University [‖]Zhejiang University

Abstract

LLM agents increasingly rely on tool calling, and the Model Context Protocol (MCP) standardizes it between agents and tool providers, reducing integration cost and driving rapid growth in tool scale. Yet a standardized interface does not make tool access *work* at production scale: legacy services are not MCP-callable, fast protocol evolution creates compatibility cost, large tool sets exhaust the context window, and stateful sessions complicate load balancing. We solve these with a shared control point, a *centralized MCP Gateway System* that makes MCP operational at cloud scale. The gateway breaks the direct-connect data plane and consolidates legacy API integration, protocol bridging, access control, and session-aware routing, while scaling out elastically at low per-call overhead. It scales agent tool access to thousands of cloud operations.

CCS Concepts

• **Networks** → **Cloud computing**; **Application layer protocols**; **Middle boxes / network appliances**.

Keywords

Model Context Protocol, Cloud Networks, Gateway

ACM Reference Format:

Mingxin Li, Enge Song, Yueshang Zuo, Xiaodong Liu, Rong Wen, Jian He, Jing Tie, Zhou Shao, Qiang Fu, Xiaobo Xue, Xiong Xiao, Luyao Zhong, Shaokai Zhang, Jiangu Zhao, Jianyuan Lu, Shize Zhang, Xiaoqing Sun, Changgang Zheng, Zihao Fan, Haonan Li, Tian Pan, Xiaomin Wu, Yang Song, Xing Li, Biao Lyu, Meng Li, Haipeng Dai, Guihai Chen, and Shunmin Zhu. 2026. Scaling LLM Agent Tool Access at Cloud Scale. In *The 10th Asia-Pacific Workshop on Networking (APNet 2026), August 06–07, 2026, Singapore, Singapore*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3820441.3821235

1 Introduction

LLM applications are rapidly moving from prompt-only chatbots into tool-using agents that plan, select, and invoke external tools. The ecosystem now centers on the Model Context Protocol (MCP) [6], which standardizes how providers expose capabilities and how an agent (*MCP Host*) discovers and invokes them, cutting point-to-point integration cost. This unlocked *massive MCP*: $O(10^4)$ MCP servers appeared within thirteen months of release [4].

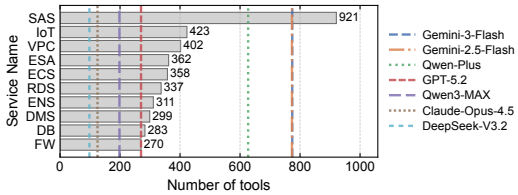


Figure 1: API operations exceed the LLM context window.

The default MCP workflow—*direct-connect* (one MCP client per service), *expose-all* (every tool schema inlined into the LLM context), and *stateful* (the server holds session state)—is not production-ready at this scale. *(C1) Legacy infrastructures are not MCP-native.* Legacy capabilities live behind $\sim 10^6$ OpenAPI servers [8], dwarfing the $\sim 10^4$ MCP servers. A single cloud provider exposes $\sim 10^4$ operations across $\sim 10^2$ services in 300+ categories, all candidates for MCP; per-API rewrapping does not scale. *(C2) MCP evolves faster than its deployments.* Four protocol revisions appeared in thirteen months; HTTP+SSE and Streamable HTTP coexist, over half of public clients support only the former [3], and authentication remains vendor-specific, so every backend absorbs the compatibility surface. *(C3) Expose-all tool listing crushes the LLM context window.* In our evaluation, inlining 120 cloud tools consumes > 166k tokens, pushes inference latency to 100 s on a multi-tenant MaaS, and lowers tool-selection accuracy. Even a single service exceeds LLM context (Figure 1). *(C4) Stateful MCP makes load balancing non-trivial.* MCP sessions must stick to the replica holding their state, but conventional L7 gateways route on host/path/headers and treat the MCP request body as opaque; under multi-replica deployment, the system must additionally extract and keep session-to-backend mappings consistent across instances.

These problems should be solved at a shared control point, mirroring how L7 load balancers consolidated the web era; the Microsoft MCP Gateway [5] and AWS AgentCore Gateway [1] confirm this direction. We propose an **MCP Gateway System** that onboards massive surface into MCP and operates stateful MCP at cloud scale, with contributions in §2: (D1) *function offloading* unifies heterogeneous backends behind a body-aware data plane, and (D2) *session-aware routing* lets stateful MCP services and gateways scale independently. We further flag two ongoing designs in §4: (F1) *tool recommendation* as the right answer to C3, and (F2) *decentralized synchronization* that advances C4 beyond the centralized store.

Mingxin Li, Enge Song, and Yueshang Zuo contributed equally to this work.

APNet 2026, Singapore, Singapore

ACM ISBN 979-8-4007-2664-4/2026/08
https://doi.org/10.1145/3820441.3821235

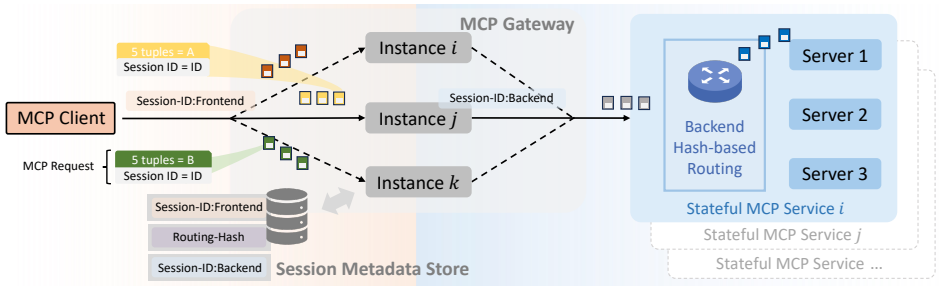
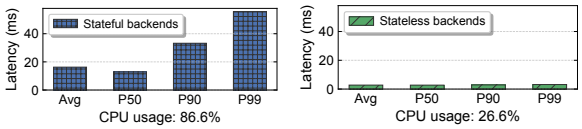


Figure 2: Session-aware routing across replicated gateways and backends.



(a) Stateful backend **(b) Stateless backend**

Figure 3: Gateway request latency, 4 instances.

2 System Design

The gateway breaks the direct-connect model between clients and backends: it parses every request and response on the data plane.

2.1 Function offloading

The gateway centralizes coordination that a client and server would otherwise negotiate directly, decoupling the client from each backend's API surface, transport, version, and security scheme. *API unification.* At the data layer the gateway compiles legacy OpenAPI operations into MCP tools, so the massive existing API surface becomes MCP-callable without backend refactoring; `List Tools` and `Call Tool` primitives are translated to the corresponding API invocations and responses are normalized to the MCP shape. *Protocol adaptation.* At the transport layer the gateway bridges HTTP+SSE and Streamable HTTP per backend via a staged Parse→Annotate→Rewrite→Forward pipeline, hiding protocol churn from clients. *Access control.* The gateway exposes a unified ingress interface, maps identities to backend-specific credentials on egress, and enforces fine-grained per-user tool visibility, without any backend changes.

2.2 Session-aware routing

Stateful MCP requires *session affinity*: every request must reach the replica holding its session context, yet production needs *both* the gateway and backend tiers to scale elastically, so routing must stay consistent regardless of which gateway a request lands on. Our scheme is a *segmented*, two-tier design (Figure 2): a gateway cluster and a backend MCP-service cluster that scale independently. Gateways are kept *stateless*: instead of synchronizing peer-to-peer, each reads and writes a shared *session-metadata store* that maps every frontend session to its routing-hash and backend session. Across the gateway–backend boundary, the gateway tags requests with a *routing-hash* derived from the session, and the backend service applies stateless consistent-hash forwarding on this key alone; the gateway never tracks backend replicas and the backend never interprets session semantics, so the backend tier scales independently.

Table 1: Agent function-calling success rate vs. # tools N.

N (#tools)	15	30	60	120	240	480
Success rate	91.0%	91.6%	89.3%	88.6%	88.2%	**80.8%**

Table 2: Top-K hit rates of single-channel retrievers.

Retriever	hit@1	hit@15	hit@30
Dense	50.5%	94.5%	96.5%
Sparse	57.0%	93.0%	97.5%

3 Primary Experimental Results

Gateway overhead is small and grows gracefully. At one instance the stateful SSE path runs at 4.59 ms mean (P99 6.81 ms) and the stateless API path at 2.81 ms mean (P99 3.22 ms); the gap is session-affinity logic. Under a 4-instance scale-out (Figure 3), the stateful path rises to 16.26 ms mean / 55.61 ms P99 while the stateless stays flat. The tail—session-map misses on the centralized store—is the price of stateless, elastically scalable gateways.

LLM function calling degrades with tool scale. We measure agent function-calling success rate as the LLM is given more mounted tools (Table 1). Success holds around 89–92% up to ∼100 tools but collapses to 80.8% at $N = 480$. Even modern LLMs cannot reliably select from a massive tool catalog. This motivates F1 in §4: a sub-LLM-cost pre-retriever that proposes Top-K tools so the LLM can maintain high accuracy on a small subset.

4 Future Work

(F1) Lightweight gateway-side tool recommendation. The function-calling collapse in Table 1 makes gateway-side recommendation essential at massive-MCP scale, yet existing approaches do not fit a gateway. Semantic-only retrievers (e.g., RAG-MCP [2]) are cheap but plateau on identifier-keyed cloud tools: in our preliminary measurements (Table 2), both dense and sparse retrievers stay below 60% hit@1 and around 93–94.5% hit@15, short of what agents need. Re-tuning a learned retriever (e.g., ToolLLM [7]) is impractical on a cloud gateway facing diverse, multi-tenant tools. Since cloud tools are keyed by identifiers and acronyms, we argue for a lightweight gateway-side retriever fusing lexical and semantic signals.

(F2) Decentralized session synchronization. The centralized session-metadata store keeps gateways stateless but becomes the coordination bottleneck. Empirically, a Redis-backed store for cross-gateway session sync degrades end-to-end SLA by 21× unavailable time and amplifies reads/writes per session-affine request. The root causes are twofold: any centralized store caps gateway SLA at

its own availability, and a Raft-replicated alternative for stronger guarantees also amplifies coordination overhead on the per-request critical path. Decentralized alternatives (encoding routing state into the Session-ID, or exchanging lightweight ownership hints across gateways) could remove this dependency while preserving affinity.

5 Conclusion

The MCP Gateway breaks direct-connect with a body-aware control point that L7 load balancers cannot provide, unifying centralized function offloading and session-aware routing into one design that scales agent tool access to thousands of tools per agent.

Acknowledgments

This work is supported in part by the Postgraduate Research & Practice Innovation Program of Jiangsu Province (No. KYCX24_0248). The authors used ChatGPT and Claude to assist with language-level polishing and condensation of author-written text during the preparation of this manuscript, particularly in parts of the Abstract, Introduction, Section 2.2, and Section 4 (F2). All AI-assisted edits were reviewed by the authors, who take full responsibility.

References

[1] AWS Bedrock AgentCore. 2026. *AgentCore Gateway.* https://docs.aws.amazon.com/bedrock-agentcore/latest/devguide/gateway.html Accessed: 2026-01-06.
[2] Tiantian Gan and Qiyao Sun. 2025. Rag-mcp: Mitigating prompt bloat in llm tool selection via retrieval-augmented generation. *arXiv preprint arXiv:2505.03275* (2025).
[3] Hechuan Guo, Yongle Hao, Yue Zhang, Minghui Xu, Peizhuo Lv, Jiezhi Chen, and Xiuzhen Cheng. 2025. A measurement study of model context protocol ecosystem. *arXiv preprint arXiv:2509.25292* (2025).
[4] MCP Market. 2026. *MCP Market.* https://mcpmarket.com/ Accessed: 2026-01-06.
[5] Microsoft. 2026. *Microsoft MCP Gateway.* https://microsoft.github.io/mcp-gateway/ Accessed: 2026-01-06.
[6] Model Context Protocol. 2025. *What is the Model Context Protocol (MCP)?* https://modelcontextprotocol.io/ Accessed: 2025-12-17.
[7] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, Vol. 2024. 9695–9717.
[8] Souhaila Serbout and Cesare Pautasso. 2024. APIstic: A large collection of OpenAPI metrics. In *Proceedings of the 21st International Conference on Mining Software Repositories*. 265–277.