DOME: Dynamic On-Premise MCP Edge Orchestration for AI Agents

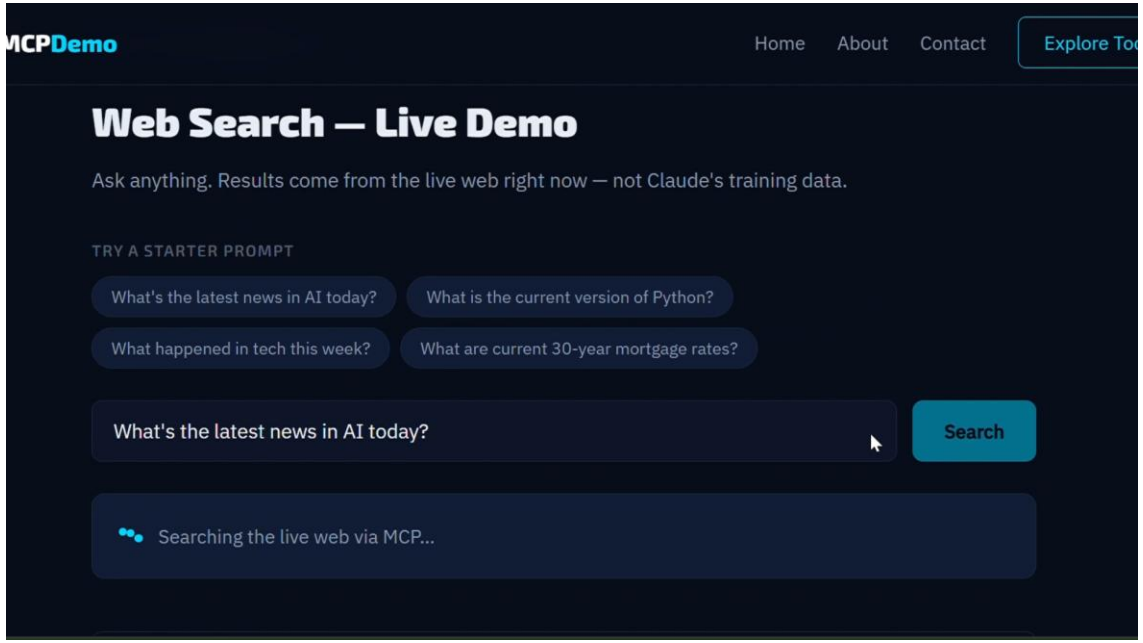
Cheng-Chia Lai

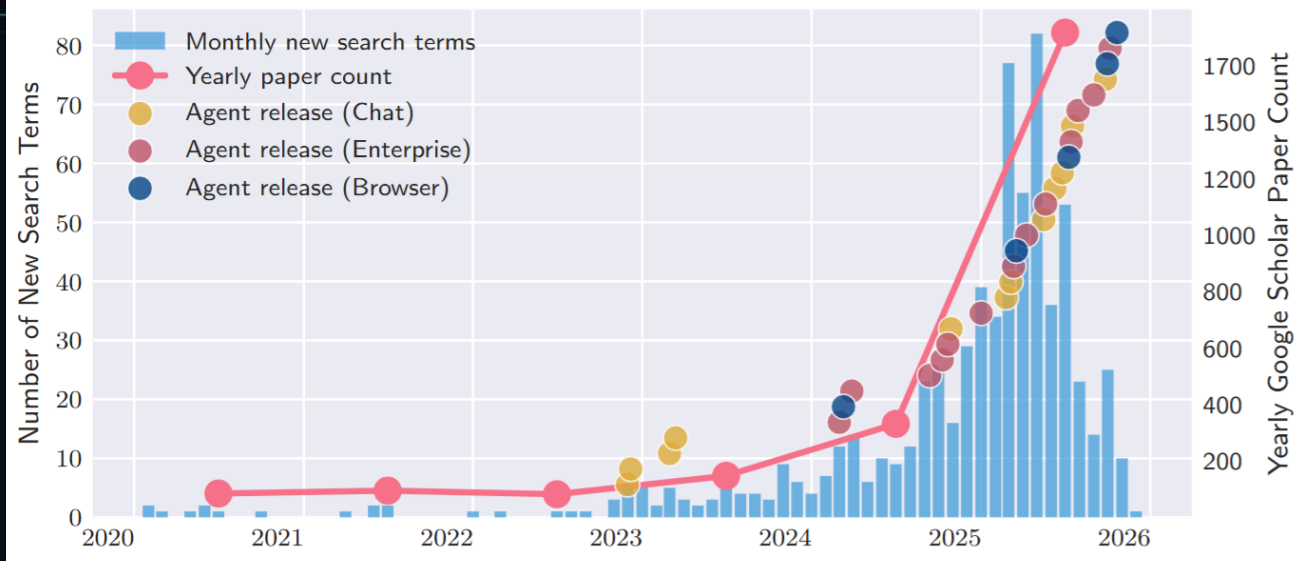
Advisor: **Cheng-Hsin Hsu**

AIINS@NTHU
Ambient Intelligence for
Immersive Networked Systems Lab

國立清華大學
NATIONAL TSING HUA UNIVERSITY

More Capable Agents, Higher Demands





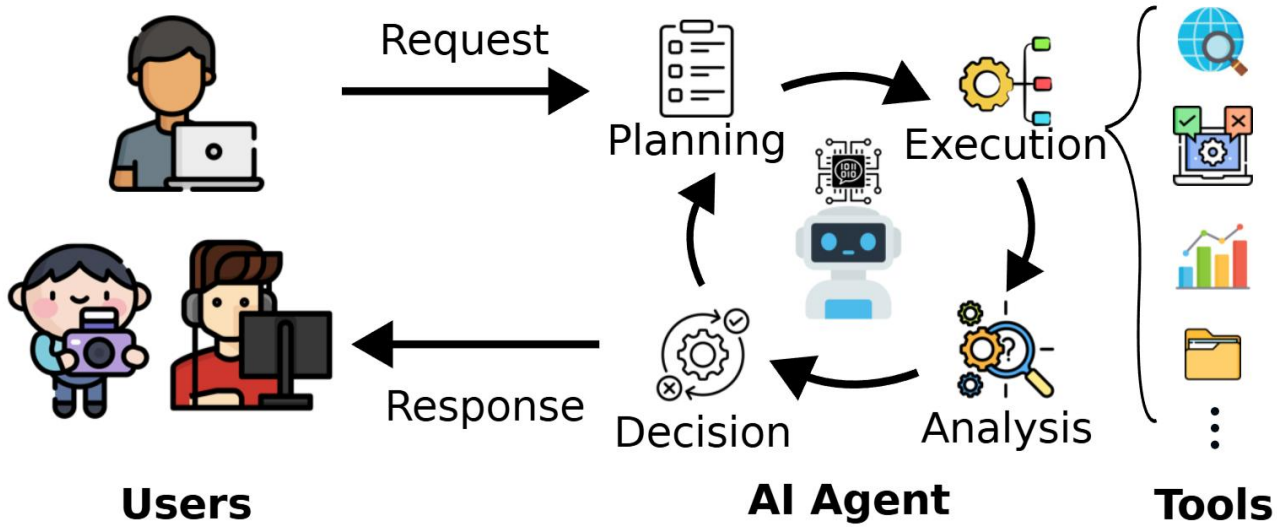
Agents are becoming increasingly capable and widely adopted.

[1] MCPDemo, https://mcpdemo.com/tools/web-search/demo.html
[2] Islam, Md Asadul et al. "The rise of agentic AI: Synthesis of current knowledge and future research agenda." Global Business and Organizational Excellence 45.3 (2026): 402-416.

What is an AI Agent?

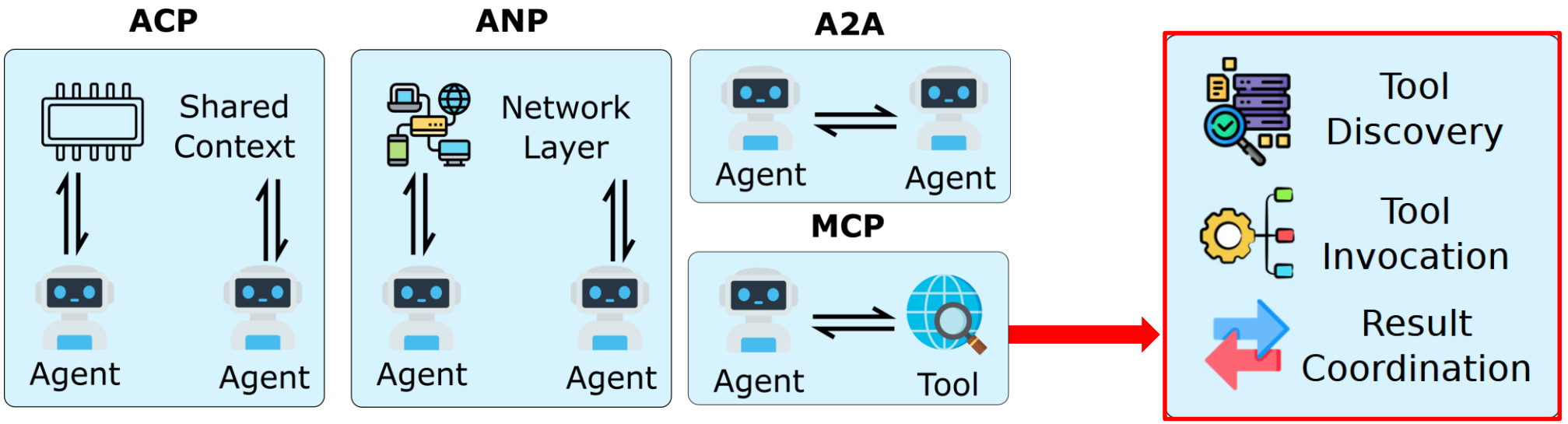
LLM-Driven Autonomous Loop

- *Planning*: Decompose request into subtasks
- *Execution*: Invoke external tools
- *Analysis*: Interpret tool results
- *Decision*: Determine next action or return response



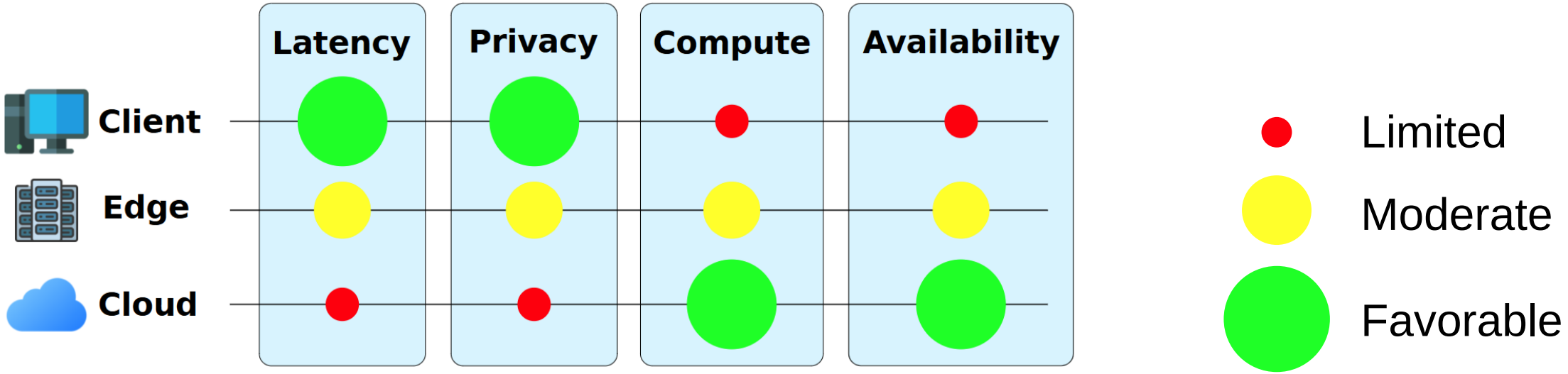
Agent Communication Protocols

- *Agent Context Protocol (ACP)* - Open source community
- *Agent Network Protocol (ANP)* - Open source community
- *Agent to Agent (A2A) - Google*
- *Model Context Protocol (MCP) - Anthropic*



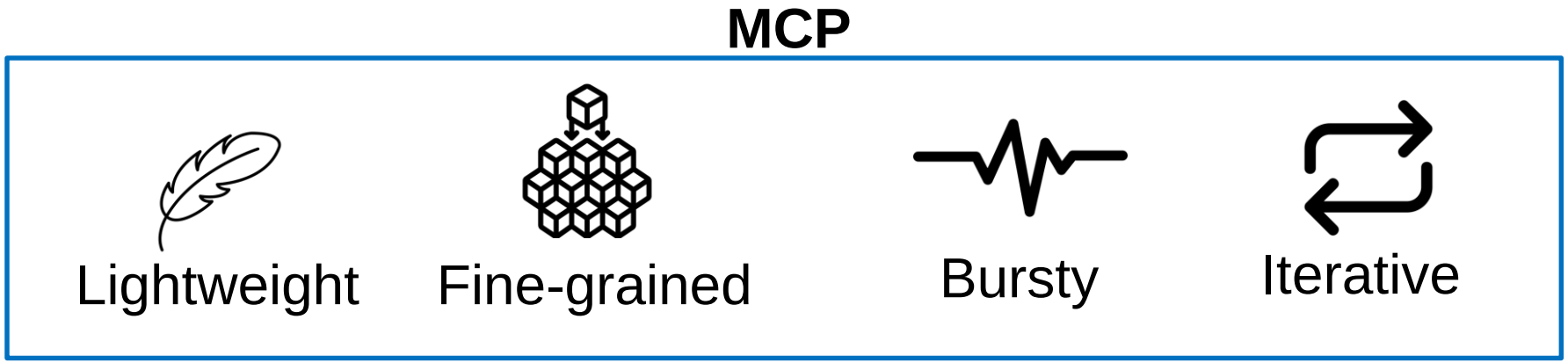
Deployment Options for MCP Tools

- *Client:* Low latency, but limited computing resource
- *Cloud:* High availability, but privacy risks
- ***Edge:*** Balanced, but orchestrating remains an open problem



Existing Orchestration Paradigms

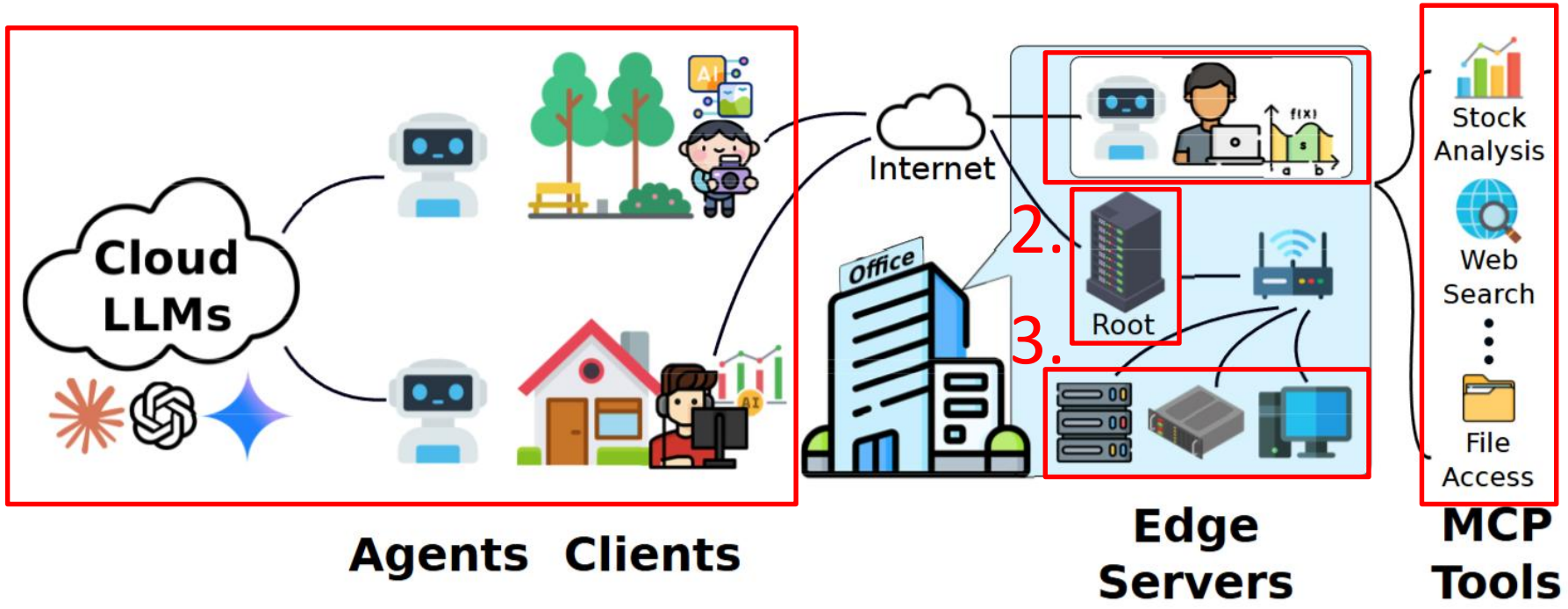
- *Container*: Heavyweight
- *Function-as-a-Service*: Cold-start overhead
- *Service Mesh*: Overengineered
- *Workflow Engine*: Coarse-grained



None of these were designed for MCP tools

DOME: Dynamic On-Premise MCP Edge

Client → Root edge server → Edge servers → MCP tool



Orchestrating MCP tools across heterogeneous edge servers introduces new challenges

Challenges

Routing: Where to handle the tool call with minimum latency

- Heterogeneous server capabilities
- Dynamic system conditions
- Unpredictable tool call patterns

Per-call Dispatch

Deployment: How many instances should run on each server

- Dynamic and unpredictable tool demand
- Heterogeneous resource capacity
- Non-negligible scaling overhead

Periodic Scaling

Contributions

- **DOME**: first MCP-native edge orchestration framework for on-premise edge servers
- **SSD**: probabilistic delay-aware routing algorithm that prevents persistent load concentration
- **GS**: utility-driven deployment algorithm that adapts instance placement under resource constraints
- Validated on a real heterogeneous testbed with extensive agent-driven workload evaluations

Outline

- Introduction
- Related Work
- System Overview
- Routing
- Deployment
- Experiments
- Conclusion

Related Work

MCP Tool Routing - network-aware routing [1]

- Static deployment assumed

MCP Tool Deployment: tool management & synchronization [2]

- Runtime efficiency ignored

Joint Routing & Deployment: delay-aware edge orchestration [3]

- Not MCP-native environments

To the best of our knowledge, DOME is the first to jointly orchestrate MCP tool routing and deployment for on-premise edge environments.

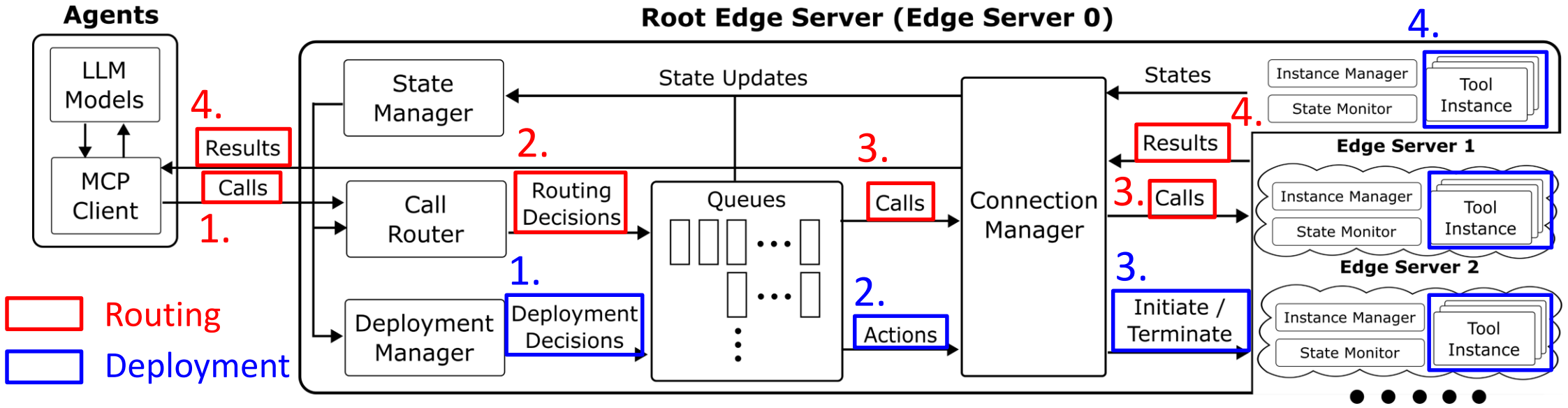
[1] E. Li et al., “NetMCP: Network-aware model context protocol platform for LLM capability extension”, arXiv preprint arXiv:2510.13467, 2025
[2] E. Lumer et al., “ScaleMCP: Dynamic and auto-synchronizing model context protocol tools for LLM agents”, arXiv preprint arXiv:2505.06416, 2025.
[3] K. Peng et al., “Delay-aware optimization of fine-grained microservice deployment and routing in edge via reinforcement learning”, IEEE Transactions on Network Science and Engineering, vol. 11, no. 6, pp. 6024–6037, 2024.

Outline

- Introduction
- Related Work
- System Overview
- Routing
- Deployment
- Experiments
- Conclusion

DOME Framework Overview

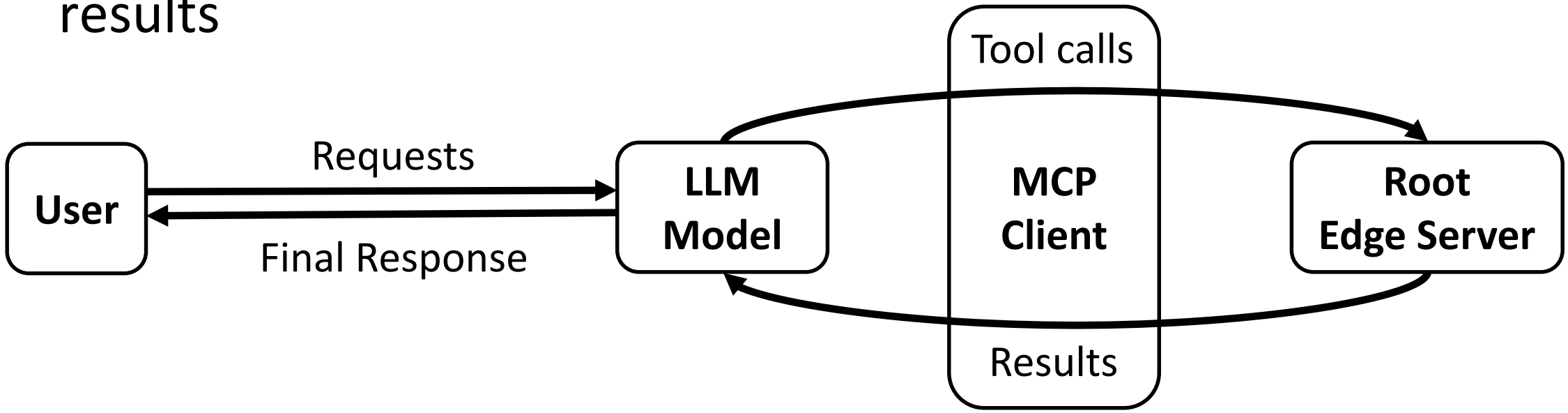
- *Agents:* Receive user requests and generate MCP tool call
- ***Root Edge Server:*** Orchestrate multiple edge servers
- *Edge Servers:* Host MCP tool instances and execute calls



Agents

Bridge the users and distributed edge servers

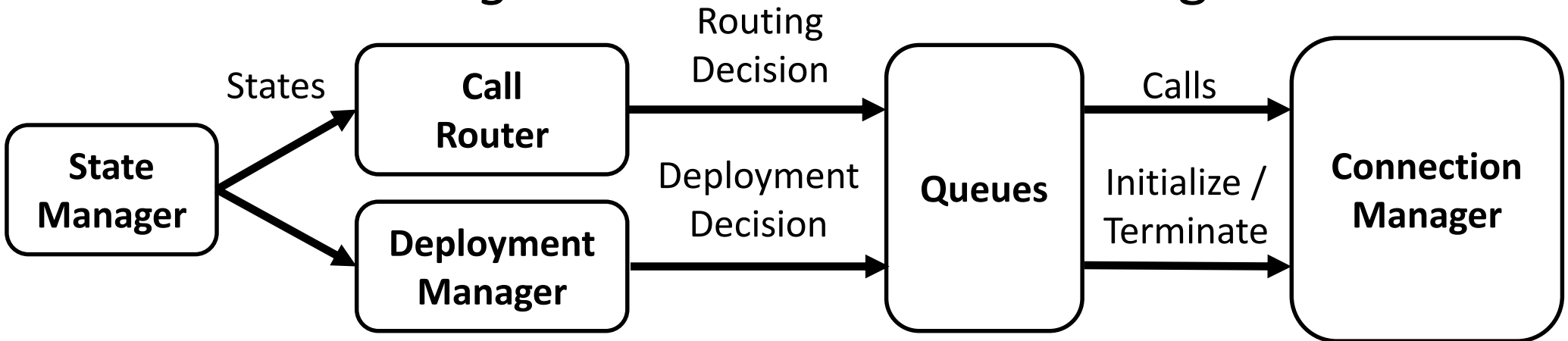
- *LLM Models*: Resolve request, select suitable tools, and generate structured tool calls
- *MCP Clients:* Abstract protocol details, send tool calls and receive results



Root Edge Server

Maintain states for fine-grained routing and deployment control

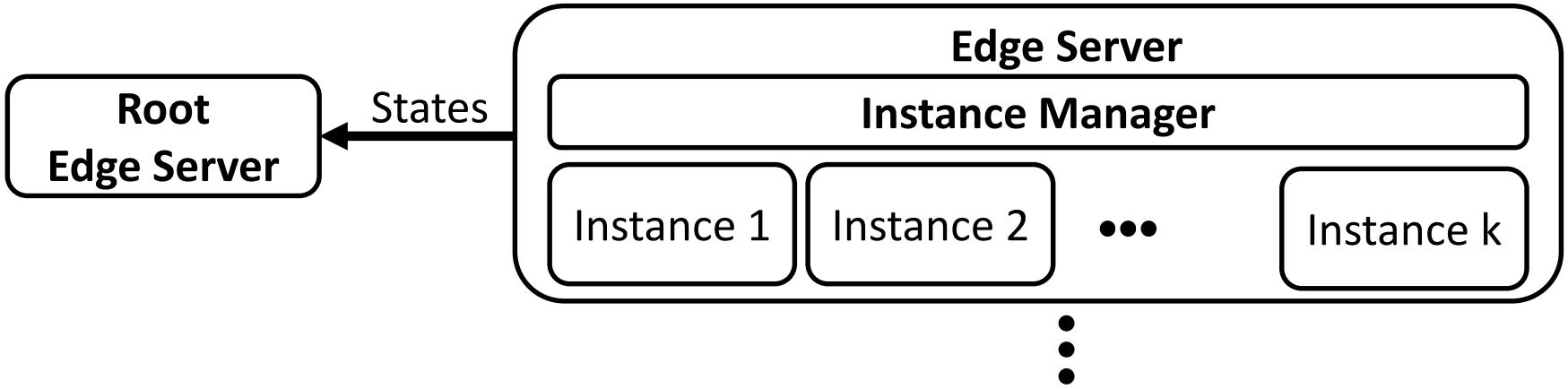
- *State Manager*: Maintain global runtime states
- *Call Router*: Dispatch tool calls to edge servers
- *Deployment Manager*: Adjust tool instances periodically
- *Queues*: Buffer tool calls
- *Connection Manager*: Communicate with edge servers



Edge Servers

Provide distributed execution environment for MCP tools

- *Tool Instances*: Execute tool calls
  - Process-level isolation
  - Sequential execution
- *Instance Manager*: Control instance lifecycle
- *State Monitor*: Collect local system states



Outline

- Introduction
- Related Work
- System Overview
- Routing
- Deployment
- Experiments
- Conclusion

Routing Problem Formulation

Goal: Select an optimal queue to minimize total delay from candidate queues whenever receiving a tool call

$$q^* = \arg\min_{q \in \mathcal{Q}_\tau} D(c, q)$$

c	: Call
q	: Queue
τ	: Tool type

Solution:

1. Predict total delay $D(c, q) \approx \tilde{D}(c, q)$
2. Apply a routing algorithm $q^* = f(\tilde{D}(c, q))$

Delay Modeling

Total delay consists of ***network*** and ***processing*** **delay**

$$\tilde{D}(c,q) = \tilde{D}_n(c,q) + \tilde{D}_p(c,q)$$

Processing Delay:

1. Queueing delay
2. Service delay

1. 2.

$$\tilde{D}_p(c,q) = \left(\frac{l_q}{k_q \mu_q} + \delta_q\right) + \frac{1}{\mu_q}$$

l	: Queue length
k	: Instances
μ	: Service rate
δ	: Remaining service time

Routing Algorithm: Soft Shortest Delay (SSD)

Minimize delay while distributing load across edge servers

1. Compute $\tilde{D}(c, q)$ for each candidate queue
2. Normalize into z-score
3. Softmax-style routing probability with γ
   - Higher γ → toward lowest-delay
   - Smaller γ → more balanced

$$P(q \mid c) = \frac{\exp(-\gamma \cdot z(c, q))}{\sum_{j \in \mathcal{Q}_\tau} \exp(-\gamma \cdot z(c, j))}$$

Outline

- Introduction
- Related Work
- System Overview
- Routing
- Deployment
- Experiments
- Conclusion

Deployment Problem Formulation

Goal: Select optimal scaling actions to maximize utility increase from candidates actions at each deployment window

$$\mathcal{A}^* = \arg\max_{\mathcal{A} \subseteq \mathcal{U}} G(\mathcal{A})$$

Each scaling action is represented as a tuple $u = (a, \tau, s)$

→ *Perform instance adjustment* ***a*** $\in \{+1, -1\}$ *with tool* $\boldsymbol{\tau}$ *in server* ***s***

Solution:

1. Model utility increase of each scaling action $\tilde{G}(u) \approx G(u)$
2. Apply a deployment algorithm $\mathcal{A}^* = g(\{\tilde{G}(u)\}_{u \in \mathcal{U}})$

Deployment Utility Increase Model

Evaluate how beneficial of a scaling action in ***Tool Congestion Reduction*** ***and Server Capacity Increase***

$$\tilde{G}(u) = \tilde{G}_\tau(u)(1 + \tilde{G}_c(u))$$

Balances between ***tool demand*** *and* ***service capacity***

- *Tool demand:* Arrival rate of tool calls
- *Service capacity:* Service rate of the tool

$$\text{Tool Utilization} = \frac{\text{Tool Demand}}{\text{Service Capacity}} \longrightarrow \rho(u.\tau) = \frac{\lambda(u.\tau)}{\mu(u.\tau)}$$

➢ ***Tool utilization:*** Tool congestion level

Deployment Algorithm: Greedy Scaling (GS)

Exhaustively search over all candidate actions is infeasible

- Greedily selects the highest-utility scaling actions without violating constraints:
  - *Availability*: At least one instance per tool type
  - *Stability*: Each tool and server participates in at most one action
  - *Saturation*: Scaling-up action is invalid on overloaded server

1. **foreach** $u \in \mathcal{U}$ **do**
   Compute $\tilde{G}(u)$ — **Evaluate**
2. Sort all actions in descending order of $\tilde{G}(u)$ into $\mathcal{U}'$ — **Rank**
3. **while** $\mathcal{U}' \neq \emptyset$ **do**
   Select and remove the top-ranked action u' from $\mathcal{U}'$
   if $u'.\tau \notin \mathcal{T}' \wedge u'.s \notin \mathcal{S}'$ **then**
   if $C_{u'.s} \geq \theta_c \wedge u'.a = +1$ **then**
   continue
   $\mathcal{A}^* \leftarrow \mathcal{A}^* \cup \{u'\}$ $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{u'.\tau\}$ $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{u'.s\}$
   if $|\mathcal{T}'| = |\mathcal{T}| \vee |\mathcal{S}'| = |\mathcal{S}|$ **then**
   break
   — **Select**

Outline

- Introduction
- Related Work
- System Overview
- Routing
- Deployment
- Experiments
- Conclusion

Baseline Algorithms

Routing Baselines

- *Random (RD)*
- *Round Robin (RR) [1]*
- *Power-of-Two Choices (PO2) [2]*
- *Join-the-Shortest Queue (JSQ) [3]*

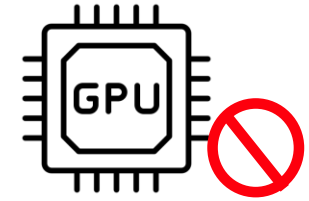
Deployment Baselines

- *Static*
- *Threshold-based Scaling (TS)*

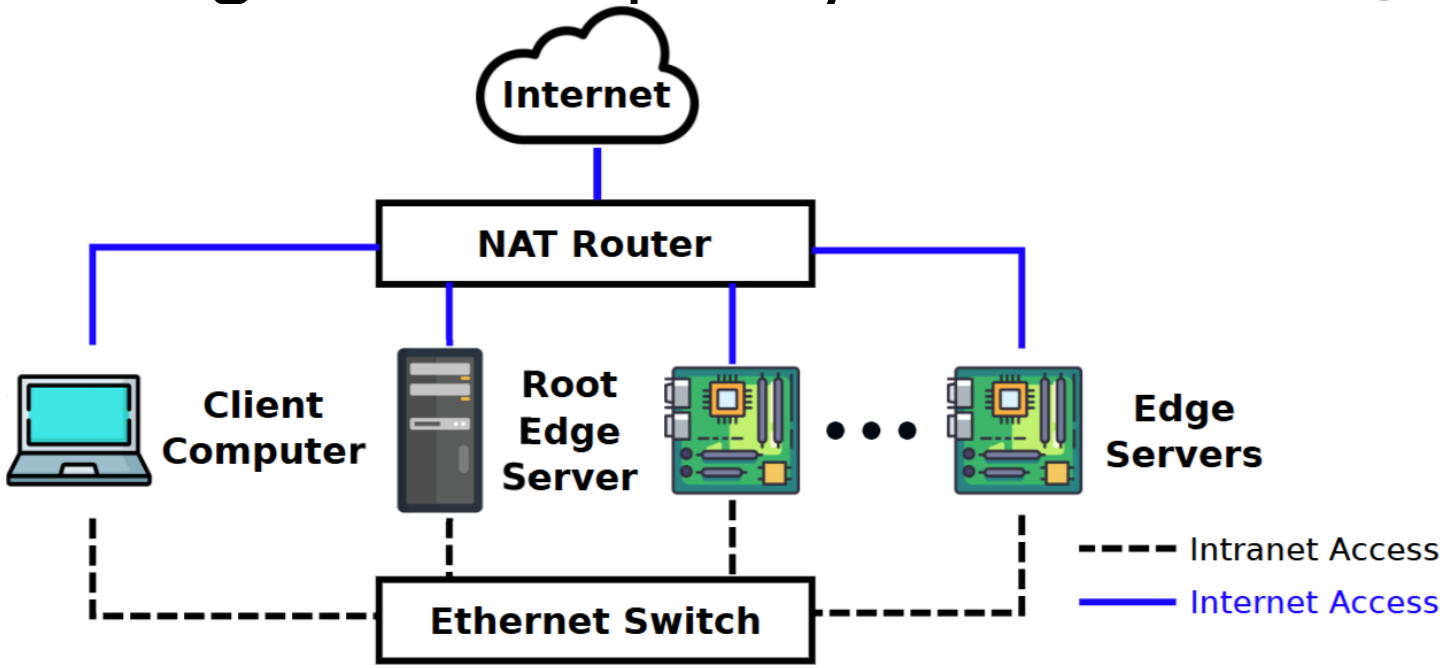
[1] R. V. Rasmussen and M. A. Trick, “Round Robin scheduling–a survey”, European Journal of Operational Research, vol. 188, no. 3, pp. 617–636, 2008.
[2] M. Mitzenmacher, “The power of two choices in randomized load balancing”, IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 10, pp. 1094–1104, 2002.
[3] M. Harchol-Balter, Performance modeling and design of computer systems: queueing theory in action. Cambridge University Press, 2013.

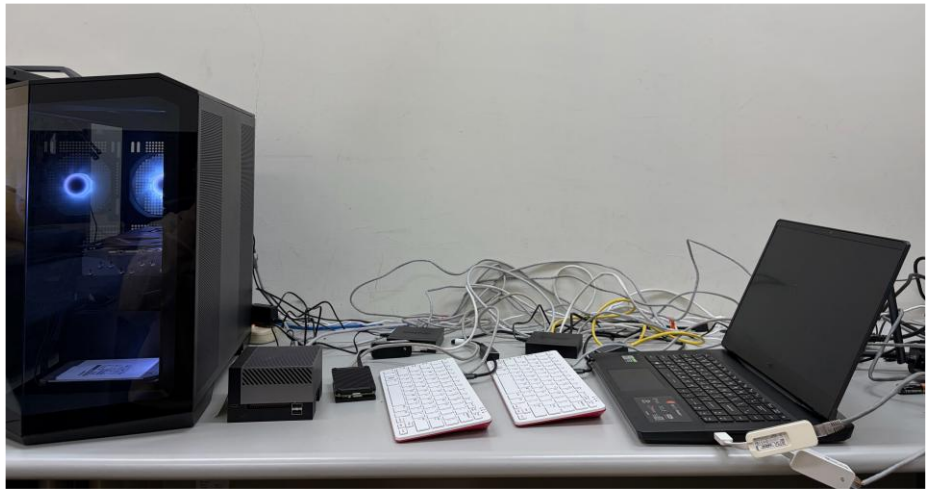
Experiment Setup

- **Root**: Intel i5 10-core @ 4.7GHz
- **Edge 1**: NVIDIA Jetson AGX Orin 8-core @ 2.2GHz
- **Edge 2**: Raspberry Pi 5 4-core @ 2.4GHz
- **Edge 3&4**: Raspberry Pi 400 4-core @ 1.8GHz



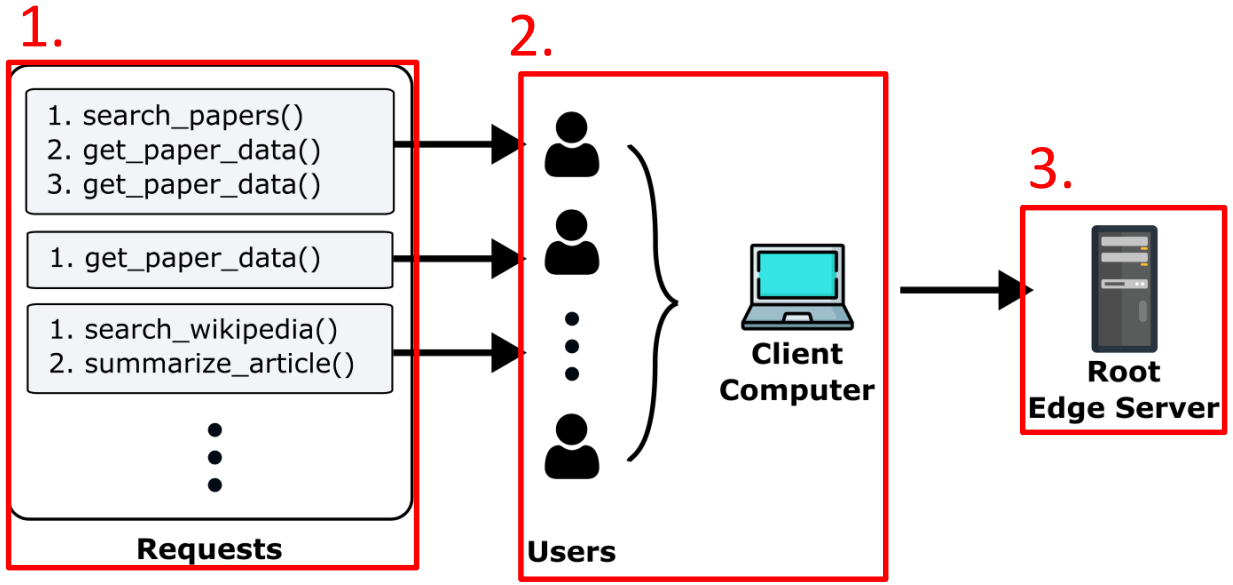
No GPU Considered





Workload Replay

Emulating real-world MCP usage through workflow replay

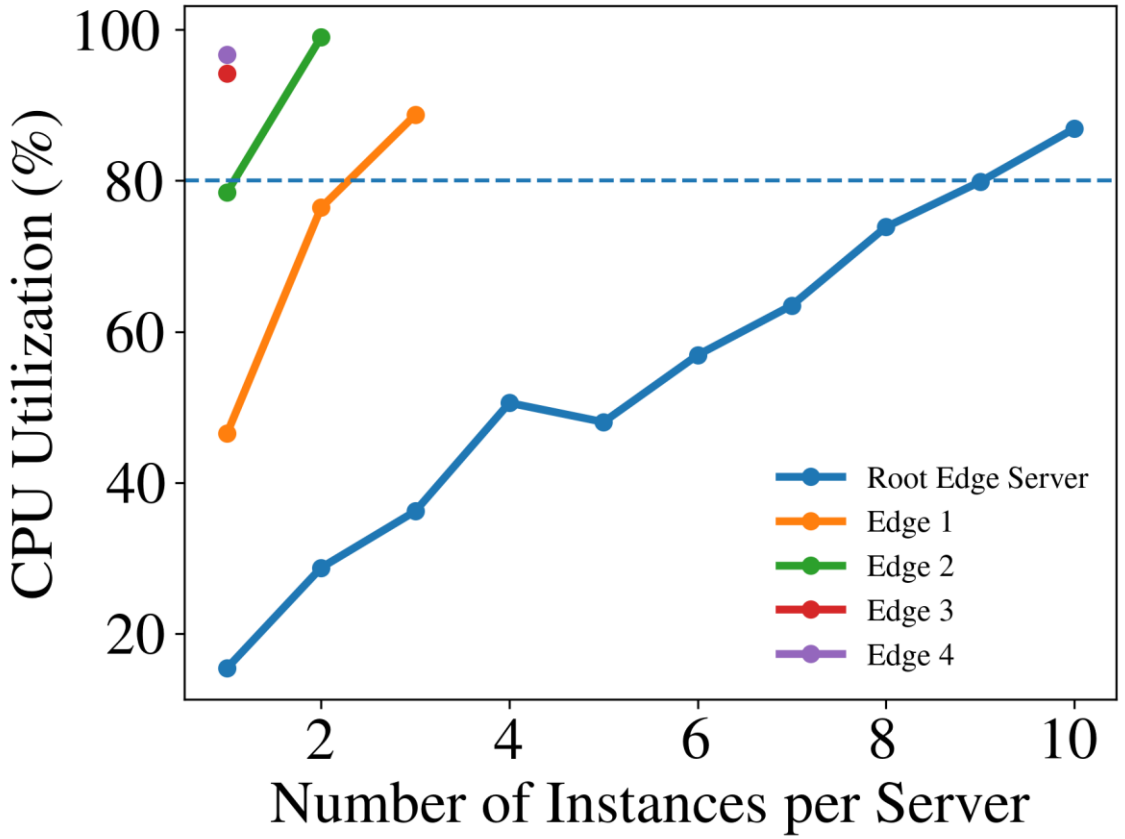


Dataset	Tools	Requests	Calls	
Lin et al. [1]	V	X	X	
Wu et al. [2]	V	X	X	
Fei et al. [3]	V	X	X	
Wang et al. [4]	V	V	X	
Luo et al. [5]	V	V	X	
Xu et al. [6]	△	V	V	
Bandi et al [7]	V	V	V	**Future work**
Mo et al. [8]	V	V	V	**This work**

[1] Z. Lin, B. Ruan, J. Liu, and W. Zhao, “A large-scale evolvable dataset for modelcontext protocol ecosystem and security analysis”, arXiv preprint arXiv:2506.23474,2025.
[2] M. Wu et al., “MCPZoo: A large-scale dataset of runnable model context protocolservers for AI agent”, arXiv preprint arXiv:2512.15144, 2025.
[3] X. Fei, X. Zheng, and H. Feng, “MCP-Zero: Active tool discovery for autonomousllm agents”, arXiv preprint arXiv:2506.01056, 2025.
[4] Z. Wang et al., “MCP-Bench: Benchmarking tool-using LLM agents with complexreal-world tasks via MCP servers”, arXiv preprint arXiv:2508.20453, 2025.
[5] Z. Luo et al., “MCP-Universe: Benchmarking large language models with realworld model context protocol servers”, arXiv preprint arXiv:2508.14704, 2025.
[6] Z. Xu et al., “TOUCAN: Synthesizing 1.5M tool-agentic data from real-worldMCP environments”, arXiv preprint arXiv:2510.01179, 2025.
[7] C. Bandi et al., “MCP-Atlas: A large-scale benchmark for tool-use competencywith real MCP servers”, arXiv preprint arXiv:2602.00933, 2026.
[8] G. Mo et al., “LiveMCPBench: Can agents navigate an ocean of MCP tools?”,arXiv preprint arXiv:2508.01780, 2025.

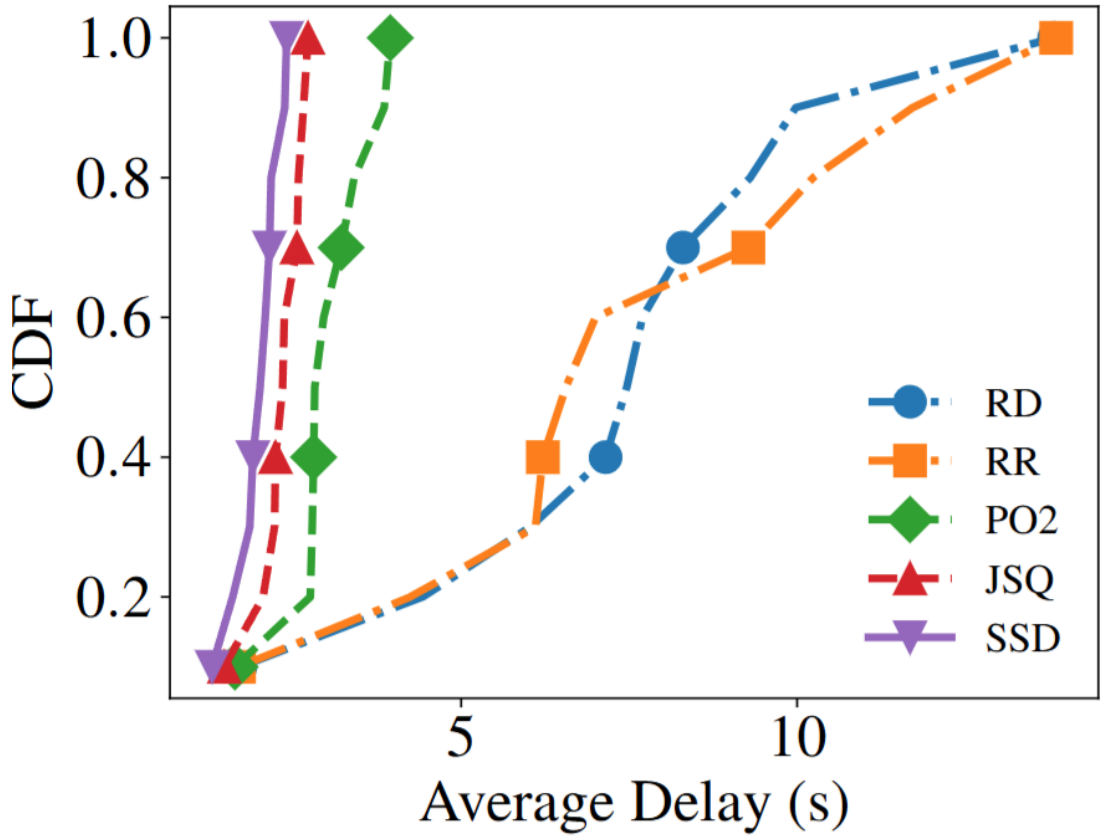
Routing Experiment Setup

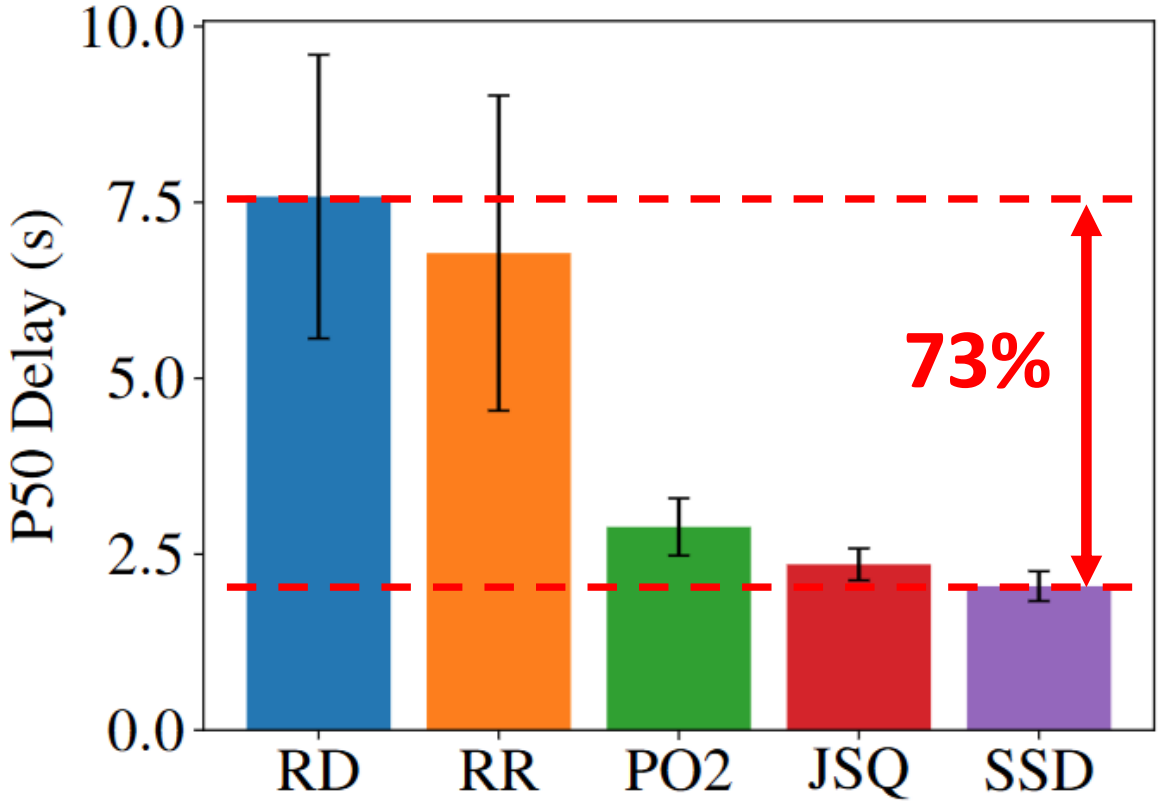
- Number of instances per tool for each server: {9, 2, 1, 1, 1}
- Experiment time: 300 s
- Concurrent users: {14, 28, 42}



SSD Achieves Lowest Delay

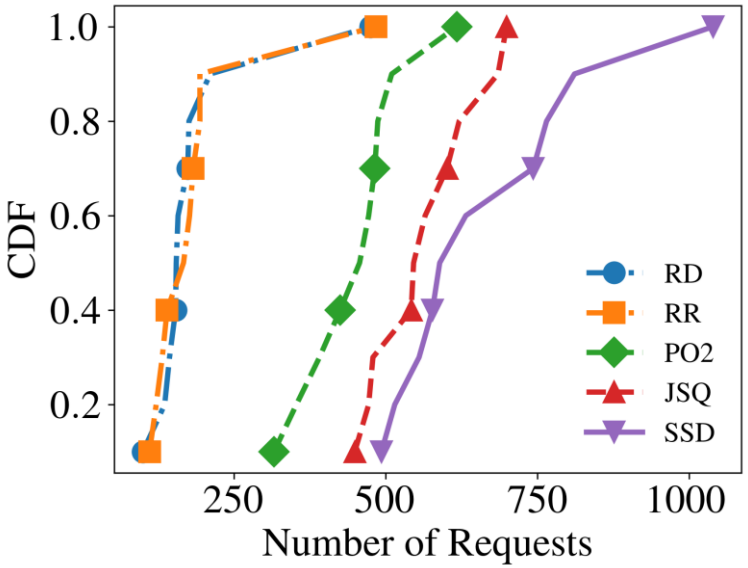
- SSD consistently performs faster than all baselines
- SSD achieves lowest P50 delay with 73% lower than RD

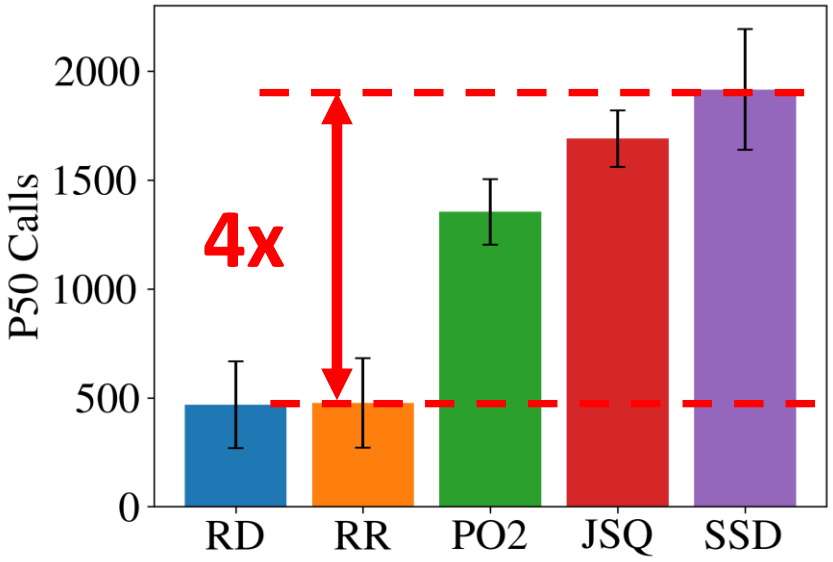


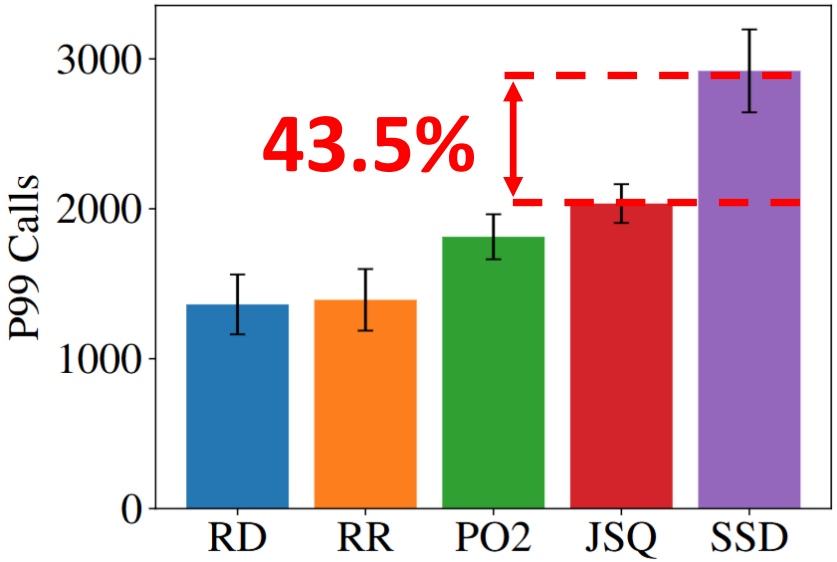


SSD Completes More Requests and Calls

- SSD consistently completes more user requests
- SSD completes nearly 4x more calls averagely than RD/RR
- SSD completes 43.5% more calls at peak performance than JSQ

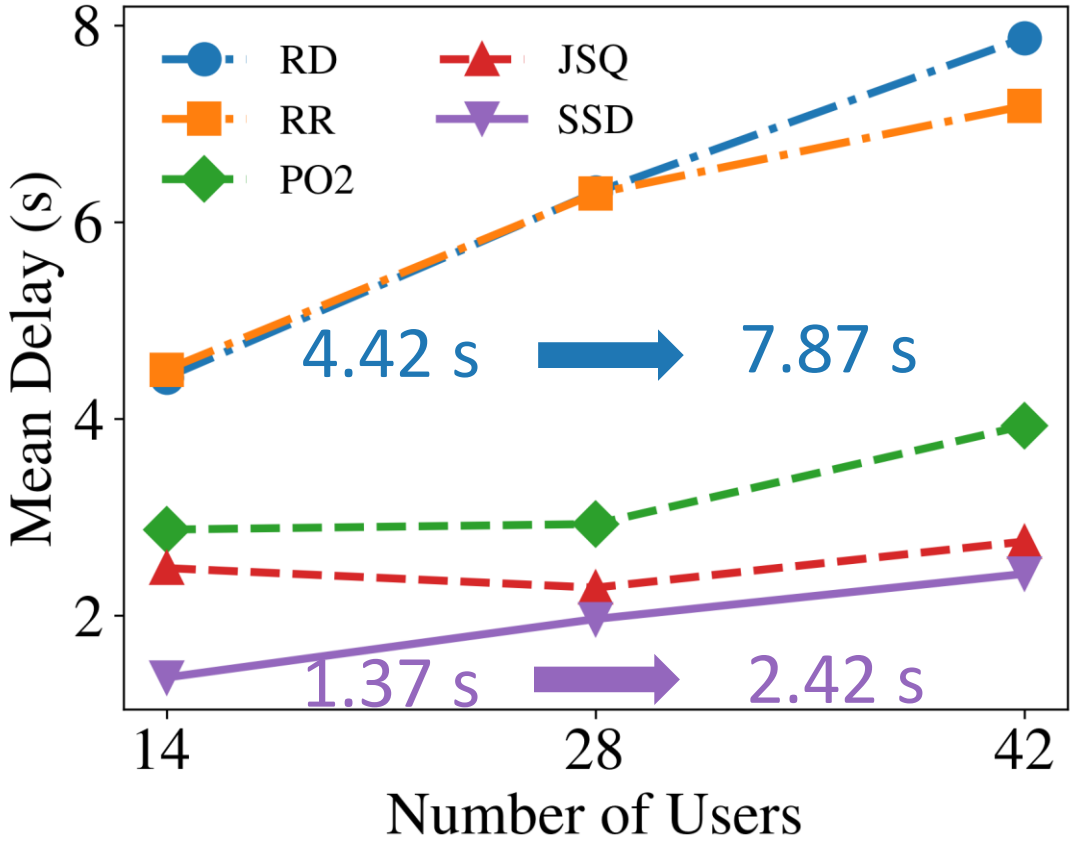


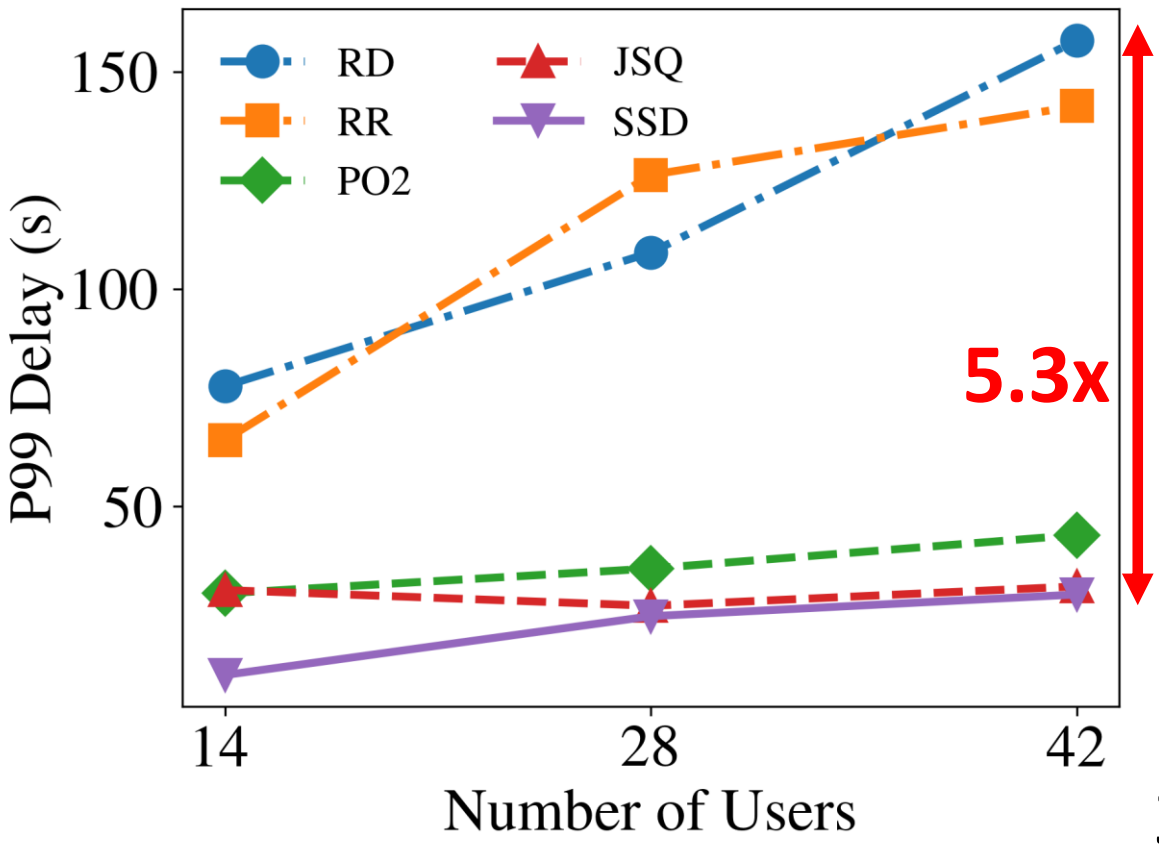




SSD Scales Well Under Increasing Load

- SSD maintains lowest P50 delay across all loads
- SSD achieves 5.3x lower P99 delay than RD

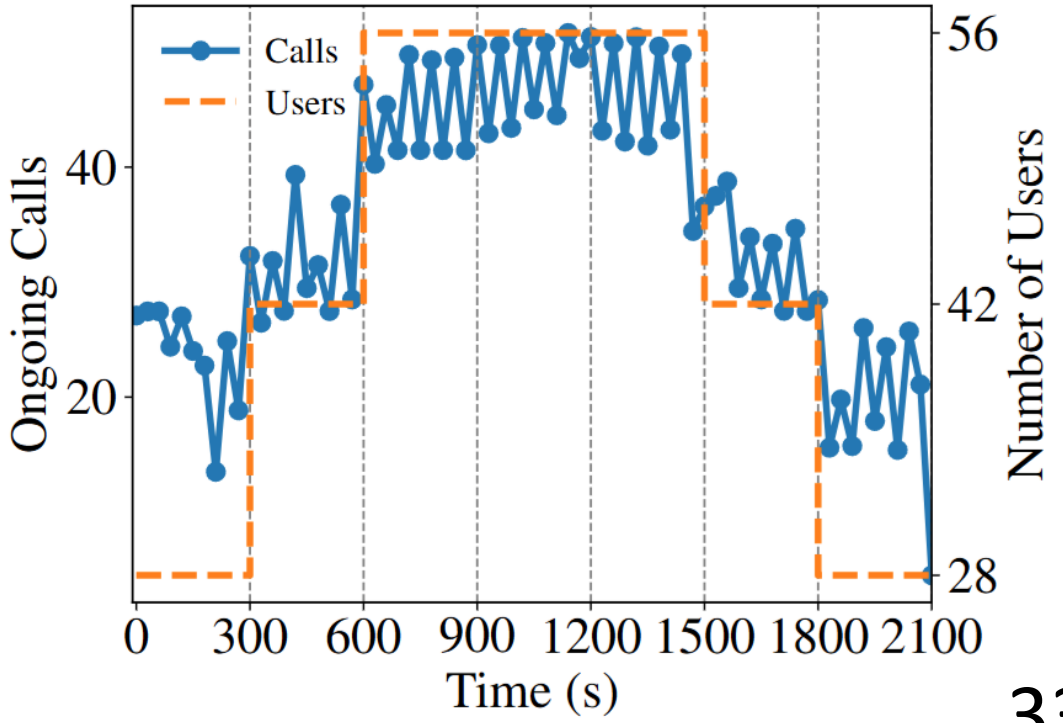




Deployment Experiment Setup

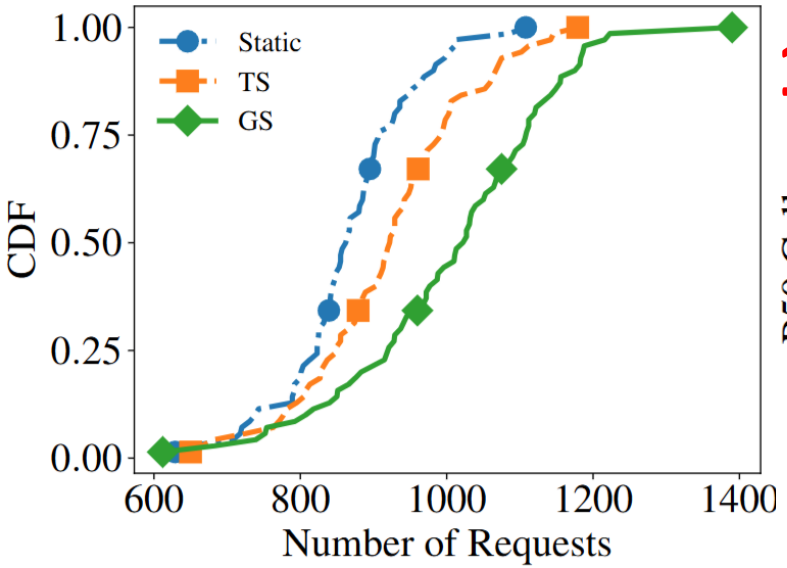
Simulate dynamic load by varying concurrent users over time

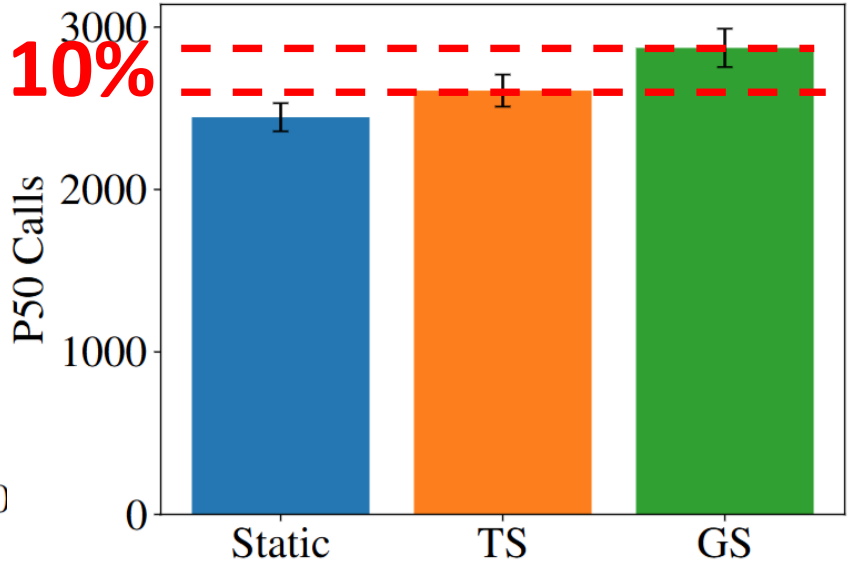
- User count *U* changes every 300 s:
  - U → 1.5U → 2U → 2U → 2U → 1.5U → U
  - U = 28 as default
- Deployment window: 60 s
- SSD as default routing algorithm

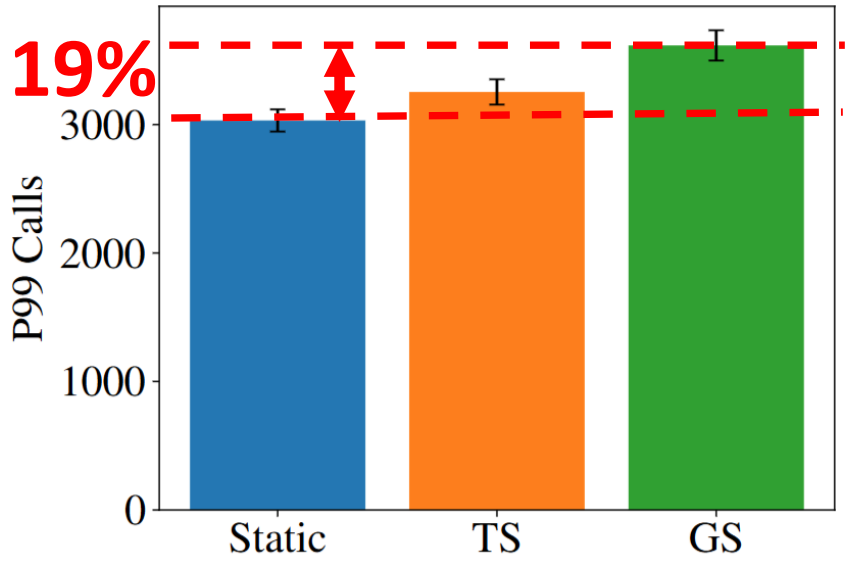


GS Completes More Requests and Calls

- GS consistently completes more requests and calls
- GS completes 10% more P50 calls than *TS*
- GS completes 19% more calls than *Static* at peak performance

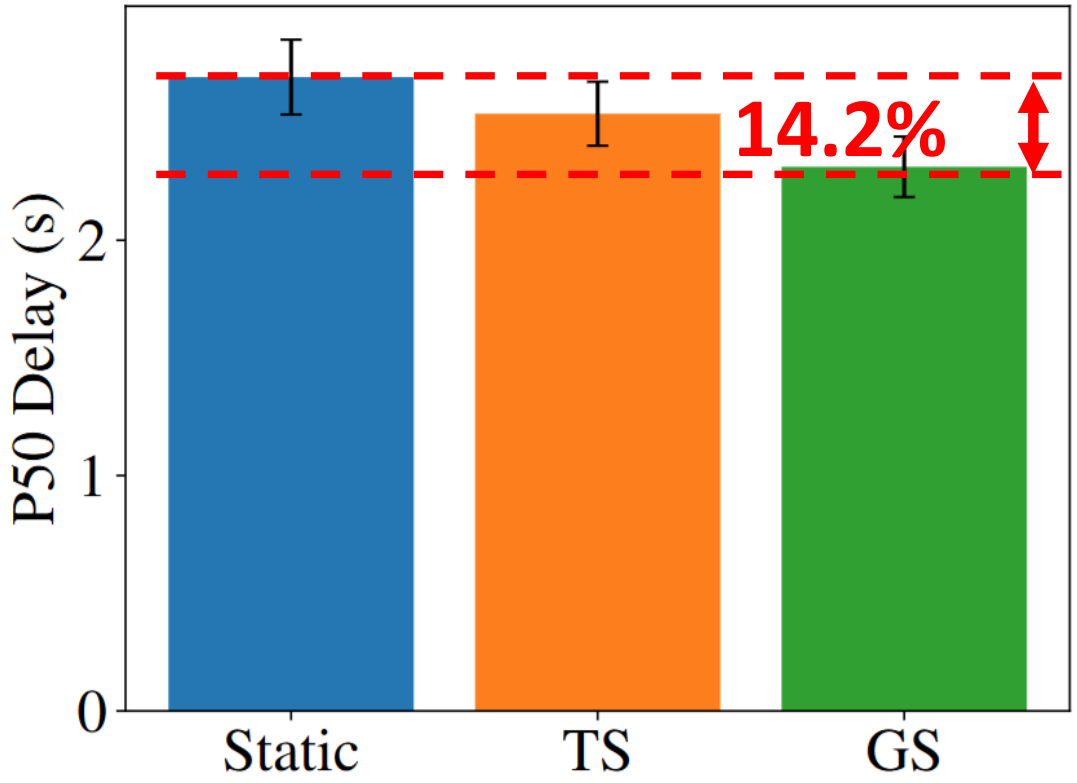


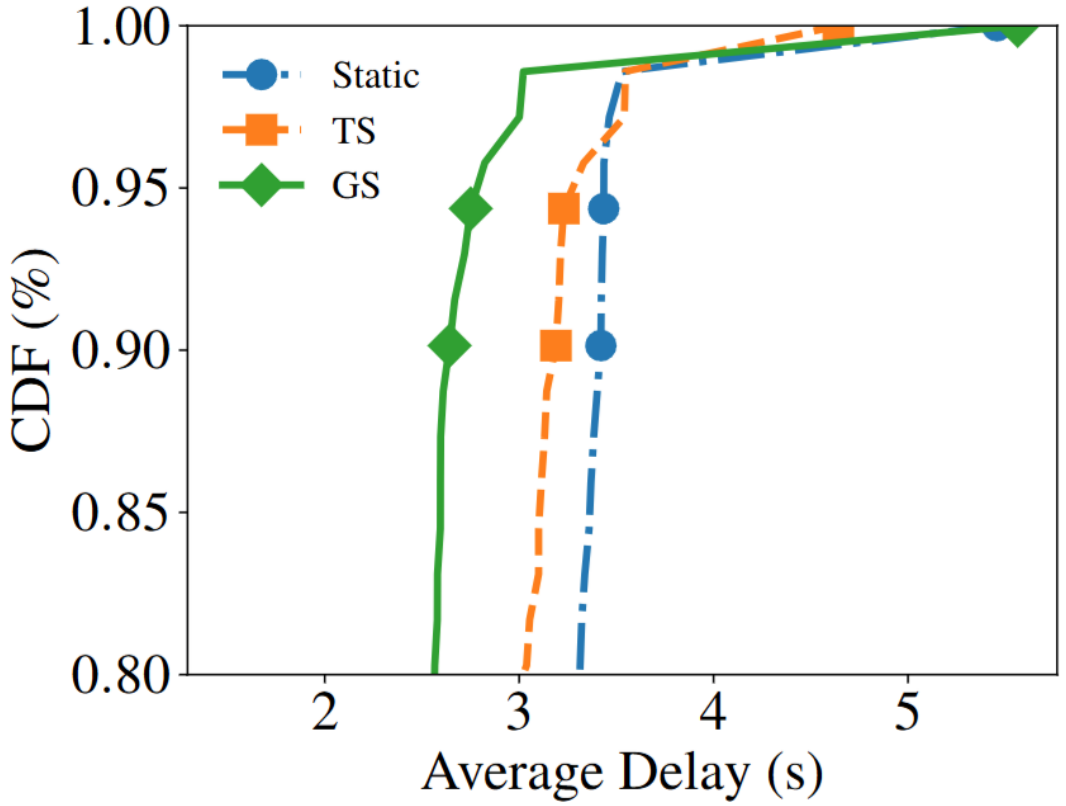




GS Improves Delay Performance

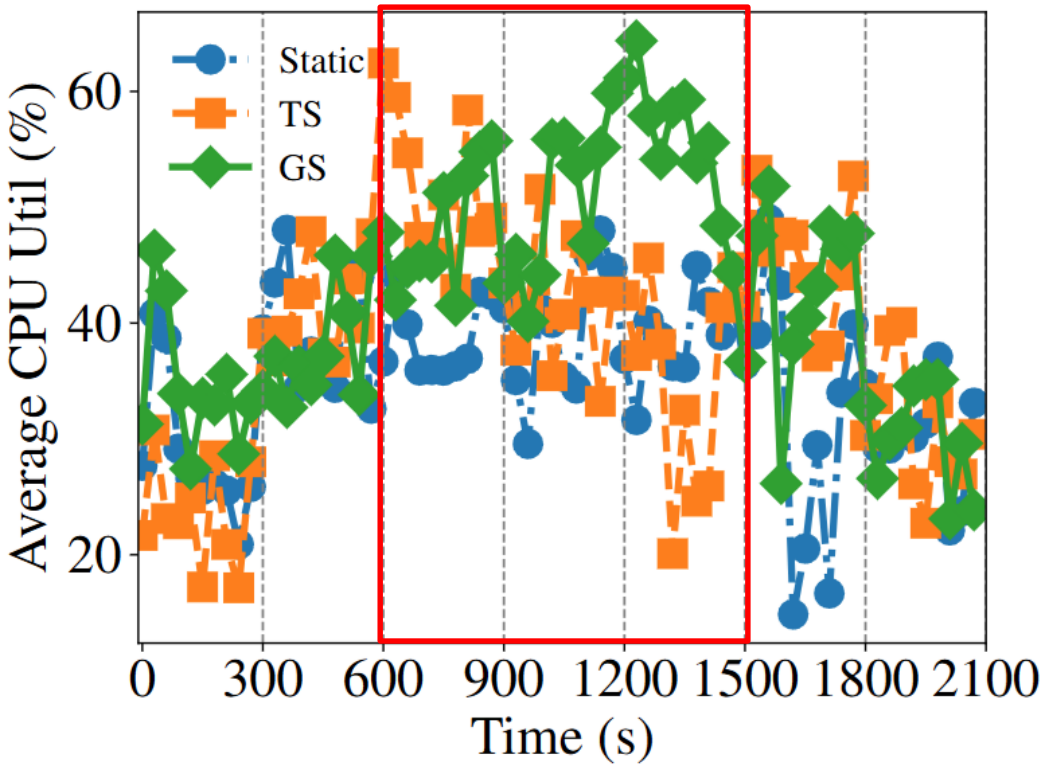
- GS achieves P50 delay with 14.2% lower than *Static*
- GS maintains lowest delay than baselines

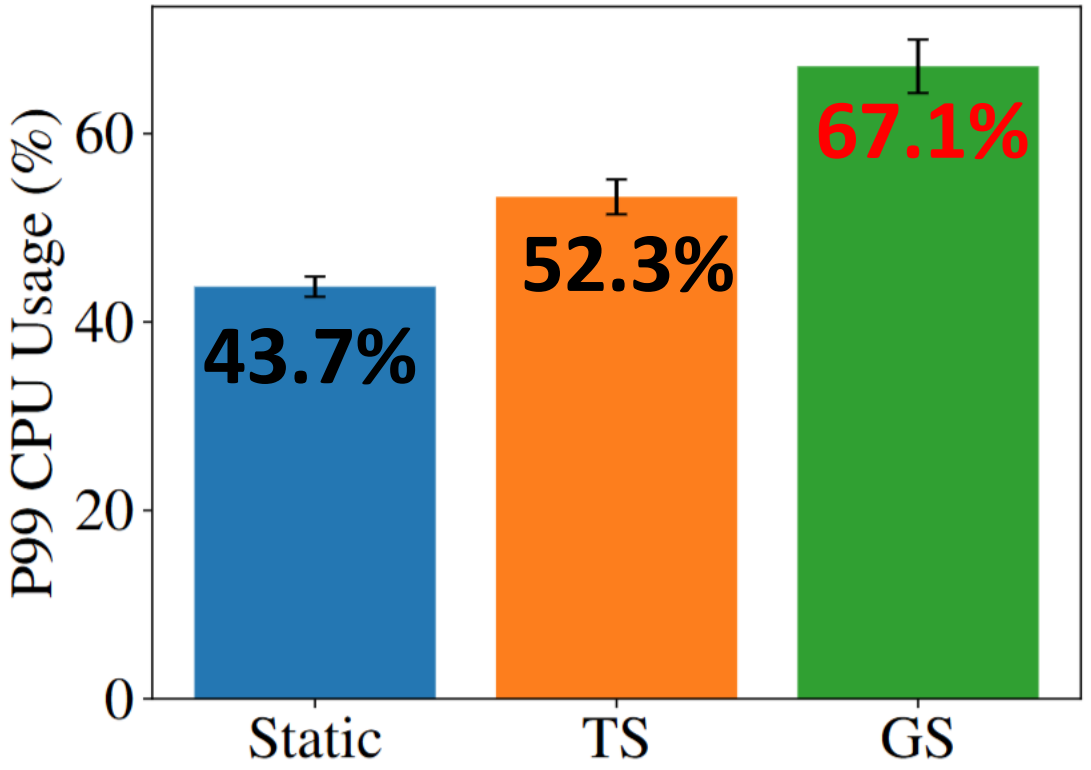




GS Improves Resource Utilization

- GS generally maintains higher CPU utilization
- GS fully utilizing heterogeneous edge resources at high load to 67.1% while staying below predefined threshold 80%





Outline

- Introduction
- Related Work
- System Overview
- Routing
- Deployment
- Experiments
- Conclusion

Conclusion

- Proposed **DOME** for MCP-native edge orchestration
- Designed **SSD** for delay-aware tool-call routing
  - ↓ P50 delay **73%**
  - ↓ P99 delay **82.4%**
  - ↑ Completed tool-calls **4.10x**
- Developed **GS** for adaptive deployment scaling
  - ↑ Calls completed **19%**
  - CPU utilization remained below the threshold
- Implemented and evaluated DOME on a **real-world heterogeneous edge testbed**

Thank you !

Thanks for the help of Prof. Hsu and Tun-Yuan Chang

Future Directions:

- Large-scale edge orchestration
- Real-world agent integration
- Multi-objective orchestration

[1] DOME: Dynamic On-Premise MCP Edge Orchestration for AI Agent, Cheng-Chia Lai, Tun-Yuan Chang, Cheng-Hsin Hsu, Proceedings of the Tenth ACM/IEEE Symposium on Edge Computing, Under review
[2] CARLA-Twin: A Large-Scale Digital Twin Platform for Advanced Networking Research, Zifan Zhou, Cheng-Chia Lai, Bo Han, Cheng-Hsin Hsu, Songqing Chen, Bin Li, Proc. of IEEE International Conference on Computer Communications for Digital Twins over NextG Wireless Networks (DTWin'25), London, United Kindom, May 2025
[3] A Software-Defined Approach to Enabling Network Controllers for Smart Environment Digital Twins, Chih-Chun Wu, Cheng-Chia Lai, Nalini Venkatasubramanian, Cheng-Hsin Hsu, Proc. of IEEE International Conference on Cloud Computing Technology and Science (CloudCom'24) Abu Dhabi, UAE, December 2024
[4] Urban Digital-Twin Planning for Sustainable Smart Cities: System Architecture, Preliminary Experiments, and Open Challenges, Cheng-Chia Lai, Agnibha Sarkar, Phuong Anh Dinh, Suyash Gaikwad, Cheng-Hsin Hsu, Proc. of ACM International Workshop on Middleware for Digital Twins (Midd4DT'24) Hong Kong, China, December 2024
[5] Evaluating subsurface single-hop WiFi and LoRa networks for internet of underground things, Chih-Chun Wu, Rummana Rahman, Charlie Hsu, Cheng-Chia Lai, Nalini Venkatasubramanian, Cheng-Hsin Hsu, Proc. of IEEE International Workshop on Unconventional IoT Applications (Un-IoT'23)