MCP Toolbox: A Secure Multi-Tenant Tool Connectivity Layer for Agentic Systems

Hemanth Sai D
UMass Boston
Boston, MA, USA
*

Abstract

Modern agentic LLM applications require reliable access to external tools (databases, object stores, ticketing systems, and internal APIs). While the Model Context Protocol (MCP) standardizes tool discovery and invocation, production deployment additionally demands multi-tenant isolation, least-privilege authorization, sandboxed execution, and end-to-end auditability. We present **MCP Toolbox**, a production-oriented MCP server that centralizes these concerns via an offline-to-online design. Offline, the system ingests heterogeneous tool specifications (e.g., OpenAPI, SQL templates), normalizes them into a versioned tool catalog with typed input/output schemas, binds policies (RBAC/ABAC), and generates contract tests. Online, it authenticates sessions, performs scoped tool discovery, validates tool-call payloads, evaluates policies and rate limits, executes calls through sandboxed connector adapters, and emits structured audit events linked to distributed traces. We detail the full data lifecycle (how tool metadata, secrets, and invocation telemetry are acquired, transformed, and persisted), provide flow diagrams for onboarding and runtime execution, and define an evaluation protocol for correctness, robustness, and tail-latency. The toolbox serves as a reference architecture for building governable tool connectivity for enterprise agents.

1 Introduction

Agentic systems extend LLMs with the ability to query, retrieve, and modify external state. In enterprise settings, practical value arises from interactions with heterogeneous systems: relational warehouses (Postgres/Redshift), object storage (S3), ticketing (Jira/ServiceNow), messaging (Slack/Email), and internal HTTP services. However, tool integration is often implemented as application-specific glue code: each client embeds its own authentication, ad-hoc request shaping, and logging. This fragmentation causes (i) inconsistent authorization and over-privileged credentials, (ii) weak auditability (unclear provenance of actions), and (iii) brittle interfaces that fail under schema drift.

The Model Context Protocol (MCP) provides an open standard for LLM clients to discover and invoke tools exposed by an MCP server (Model Context Protocol Project, 2025; Anthropic, 2024). MCP reduces interface fragmentation, but protocol conformance alone is insufficient for production. Real deployments require deterministic enforcement of multi-tenancy boundaries, least privilege, rate limiting, sandbox constraints, and observability.

This paper proposes **MCP Toolbox**: a secure, multi-tenant MCP server that mediates all tool calls and exports audit/telemetry signals suitable for debugging, evaluation, and compliance. The central design choice is to separate *tool onboarding* (schema normalization, policy binding, test generation) from *runtime execution* (authentication, scoped discovery, authorization, sandboxed dispatch).

Preprint. Work in progress.

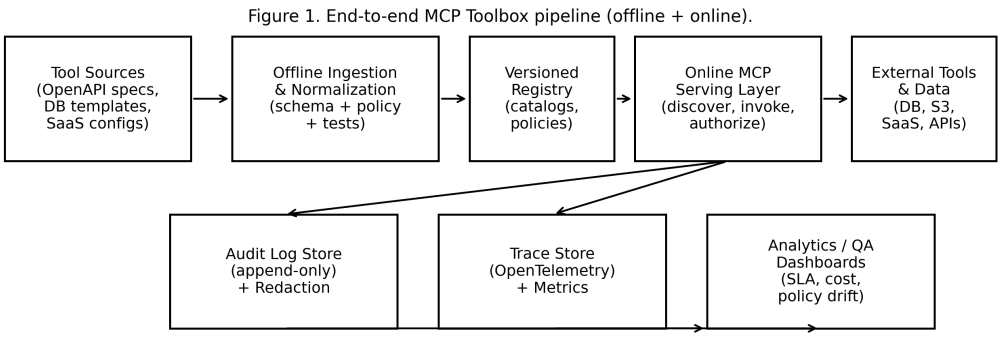

Figure 1: Offline-to-online pipeline in MCP Toolbox. Tool specifications are normalized into a versioned catalog with schemas and policy bundles. Online serving performs scoped discovery, per-call authorization, sandboxed execution through connectors, and audit/telemetry emission.

Contributions. We make four contributions: (1) an offline-to-online architecture that cleanly decouples tool definition from runtime governance; (2) a versioned tool contract (catalog + schemas + policy bundles + contract tests) that enables reproducible behavior across clients; (3) a security model for multi-tenant MCP servers based on scoped discovery and per-call policy evaluation; and (4) an evaluation blueprint focused on authorization correctness, reliability under failure, and tail-latency.

2 Background and problem setting

2.1 MCP as a tool interface standard

MCP defines an interaction surface between a tool-using LLM client and a tool-providing server (Model Context Protocol Project, 2025). In an MCP session, clients request a list of available tools, then invoke specific tools using typed parameters. MCP thereby standardizes tool discovery and invocation, enabling clients to be decoupled from concrete tool implementations.

2.2 Operational requirements beyond MCP

We consider a production environment with multiple tenants and roles. Each session is associated with a context:

$$C = (\texttt{tenant\_id},\ \texttt{role},\ \texttt{scopes},\ \texttt{client\_id}).$$

A tool is a versioned capability:

$$T = (\texttt{tool\_id},\ S_{in},\ S_{out},\ P,\ L,\ X),$$

where S_{in} and S_{out} are JSON schemas, P is a policy bundle (RBAC/ABAC constraints), L are operational limits (rate limits/timeouts), and X are sandbox constraints (allowlists/readonly modes).

Threat model. We assume adversarial prompts and indirect prompt injection attempts that may induce unauthorized tool calls. Therefore, authorization must be enforced at the toolbox boundary, not delegated to the LLM client. We also assume upstream tool systems can fail (timeouts, throttling, partial outages), and that tool payloads can contain sensitive data. These assumptions motivate fail-closed policies, bounded retries, and log redaction.

3 System overview

Figure 1 summarizes the offline-to-online design. Offline, MCP Toolbox constructs immutable artifacts: a versioned tool catalog with schemas and policies, plus contract tests. Online, it serves MCP sessions, performs scoped discovery, validates and authorizes tool calls, executes them via connector adapters, and exports telemetry.

4 Offline tool onboarding: how tool data is acquired and transformed

The onboarding pipeline converts heterogeneous tool definitions into a governed, testable contract. This pipeline is designed to be deterministic and reproducible: given the same inputs (tool specs and policy configuration), it produces identical catalog artifacts.

4.1 Tool sources

MCP Toolbox supports three common sources of tool definitions:

- **OpenAPI-described HTTP APIs:** endpoints, parameters, authentication requirements, and response schemas are extracted from OpenAPI specifications (OpenAI Initiative, 2024).
- **Database-backed tools:** SQL tools are defined as parameterized templates or stored procedure wrappers. Optionally, database introspection can be used to validate schema references.
- **SaaS connectors:** curated tool definitions for systems such as ticketing/messaging. These are typically constrained by allowlists (project IDs, channels) and mutation gates.

4.2 Normalization into a versioned catalog

For each tool definition, onboarding produces a canonical record with: (i) a stable `tool_id` name, (ii) typed JSON schemas (S_{in}, S_{out}), (iii) a policy surface (required scopes, tenant constraints, approval gates), and (iv) operational controls (timeouts, budgets).

Schema generation and validation. OpenAPI operations are mapped to tools by operation identifier; request/response objects are converted into JSON schema fragments. Database tools specify input schemas for parameters and output schemas for row sets. The pipeline validates that all required fields are typed and that bounds are explicit (e.g., maximum result size).

Policy binding. Policies are attached at onboarding time to avoid ambiguous runtime behavior. We distinguish:

- **RBAC:** role-to-scope assignments (e.g., `db:read`, `ticket:write`).
- **ABAC:** attribute constraints over parameters and targets (e.g., allowlisted schemas, bucket prefixes, project IDs).

Policy bundles may be implemented via a dedicated policy engine such as OPA (Open Policy Agent Contributors, 2025), or by compiled rule sets.

Contract tests. For each tool, onboarding emits test vectors derived from boundary conditions implied by S_{in} and by policy constraints. These tests are used as regression gates: schema drift or policy regressions become build failures.

5 Online serving: runtime control flow and sandboxed execution

At runtime, MCP Toolbox enforces two invariants: (i) *scoped discovery* reveals only tenant- and role-permitted tools; and (ii) *per-call authorization* must succeed before any connector execution.

5.1 Scoped tool discovery

Upon session initialization, the server computes an allowed tool set by filtering the global catalog by `tenant_id`, `role/scopes`, and optional `section` constraints. This prevents information leakage: clients cannot even learn of tools outside their policy scope.

5.2 Authorize-and-dispatch

Figure 2 depicts the runtime flow. Each `tools/call` request passes through: schema validation, policy decision, constraint application, sandboxed connector execution, output shaping/redaction, and telemetry emission.

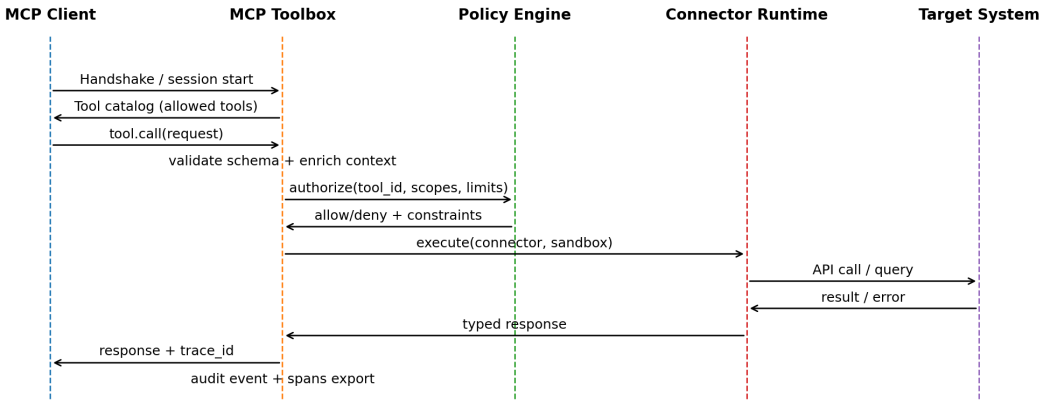

Figure 2: Runtime sequence for a tool call. The toolbox centralizes validation, authorization, execution, output shaping, and audit/telemetry emission.

Algorithm 1 Per-call policy-governed tool execution (fail-closed)

Input: session context C, tool definition T, request payload x
Validate x against S_{in}; **deny** on mismatch
Enforce tenant boundary: **deny** if $C.tenant \neq T.tenant$
Check required scopes; **deny** if missing
Apply rate limits; **throttle** if exceeded
Evaluate ABAC constraints on parameters/targets; **deny** if violated
If mutation requires approval, **return** REQUIRE-APPROVAL
Execute via connector in sandbox X (timeouts, allowlists, readonly mode)
Normalize output to S_{out}; redact sensitive fields
Emit audit event + distributed trace metadata
Return: typed result or typed error

5.3 Sandbox constraints

Sandboxing limits blast radius even for authorized calls:

- **DB sandbox:** readonly roles, statement timeouts, allowlisted schemas/tables, maximum row counts.
- **Object store sandbox:** bucket/prefix allowlists, maximum object sizes, deny-by-default writes.
- **SaaS sandbox:** allowlisted projects/channels, mutation gating, idempotency keys for write operations.

6 Data lifecycle: what happens to data and where it is stored

MCP Toolbox produces and consumes multiple data classes, each with distinct governance requirements.

6.1 Artifact stores

Tool catalog store. Versioned tool records, schemas, and policies are stored as immutable artifacts. Each release is identified by a digest and a semantic version.

Secret store. Connector credentials are stored per tenant, ideally via a secret manager, and retrieved at runtime only for authorized executions. Secrets never appear in logs.

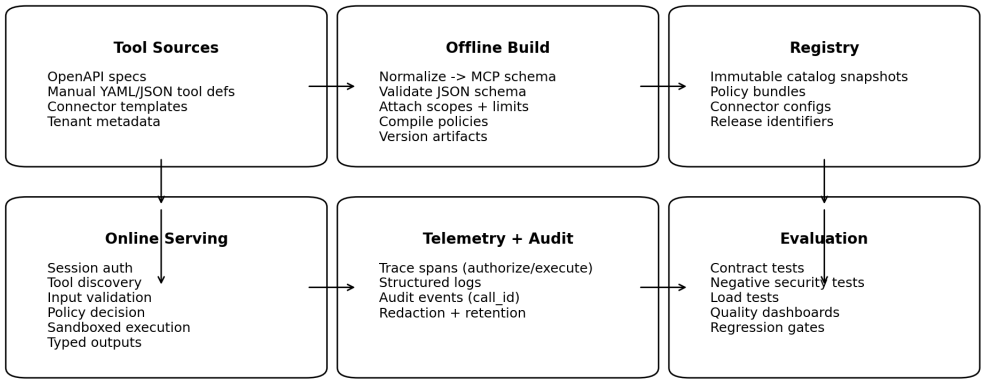

Figure 3: Data lifecycle and governance boundaries. Tool definitions and policies become versioned artifacts; runtime calls produce auditable invocation logs and traces that can be aggregated for evaluation and monitoring.

6.2 Runtime telemetry

Invocation log. Each tool call produces a structured audit record keyed by `call_id`. Required fields include: `tenant_id`, `tool_id`, decision (ALLOW/DENY/THROTTLE/REQUIRE-APPROVAL), status, latency breakdown, and `trace_id`. Payload fields are either redacted or summarized via hashes.

Trace data. The system emits distributed tracing spans to attribute latency across stages (validate, authorize, execute, shape, respond). OpenTelemetry provides a standard representation for traces and exporters (OpenTelemetry Authors, 2025).

7 Observability and auditability

Observability is not a byproduct; it is a design requirement. MCP Toolbox emits: (i) **audit events** for compliance and forensic analysis, (ii) **metrics** for availability and performance (error rates, throttle rates), and (iii) **distributed traces** for debugging tail-latency and upstream dependencies (OpenTelemetry Authors, 2025).

Audit event design. Audit records must support non-repudiation while avoiding data leakage. We recommend separating: (a) immutable metadata (who/when/what tool/decision), from (b) payload summaries (hashes, schema-conformant redacted fields), and applying retention policies per tenant.

8 Evaluation protocol

Because MCP Toolbox is infrastructure, evaluation emphasizes correctness, safety, and reliability.

8.1 Authorization correctness

We propose an oracle-driven test suite of labeled requests covering: cross-tenant attempts, missing scopes, parameter constraint violations, and rate-limit exceedance. Metrics include policy decision accuracy and false-allow rate (must be zero for cross-tenant violations).

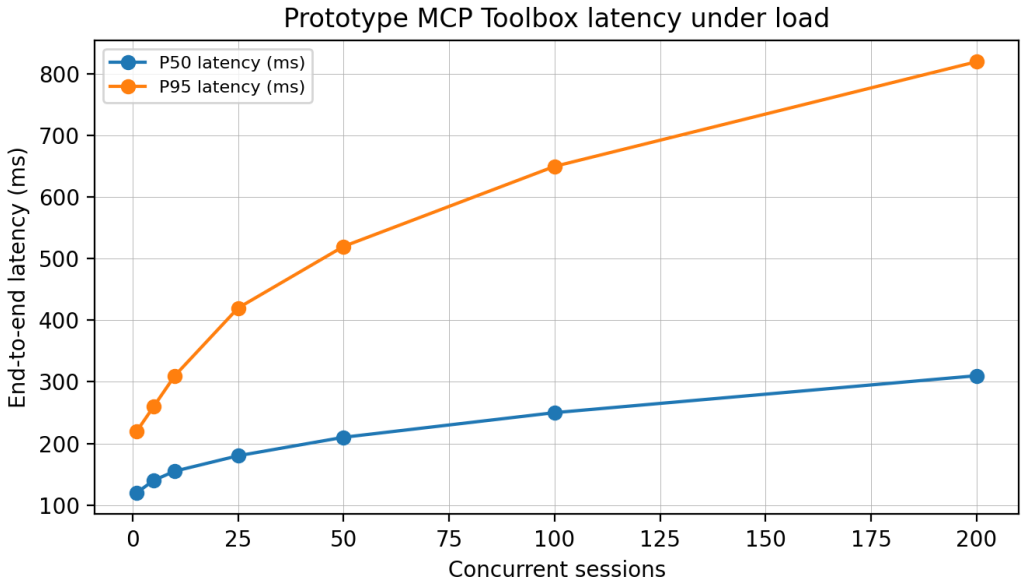

Figure 4: Representative end-to-end latency results (example profile). Tail latency is typically dominated by upstream dependencies; toolbox overhead should remain bounded and observable.

8.2 Contract and regression testing

For each tool, schema-derived contract tests validate that runtime validation rejects malformed payloads and that output shaping conforms to S_{out}. Tool version bumps require passing the entire contract suite.

8.3 Reliability and tail-latency

We recommend load tests across a representative tool mix (DB reads, object reads, HTTP calls) with fault injection (timeouts, upstream 5xx, throttling). Report p50/p95/p99 latencies and attribute time across stages.

Figure 4 illustrates a latency profile for a representative workload.

9 Implementation and development plan

We recommend an incremental development strategy that yields demonstrable artifacts at each stage:

1. **M1 (Core MCP server):** session handling, tool discovery, one safe "hello" tool.
2. **M2 (Catalog builder):** tool ingestion, schema normalization, versioned artifact publishing.
3. **M3 (Policy enforcement):** RBAC scopes, ABAC allowlists, rate limiting, deny-by-default.
4. **M4 (Connector adapters):** Postgres readonly queries, S3 fetch, HTTP OpenAPI tool.
5. **M5 (Telemetry):** audit log schema, trace spans, latency breakdown dashboard.
6. **M6 (Hardening):** circuit breakers, idempotency for writes, approval workflow for mutations.

10 Limitations

First, policy configuration complexity can increase operational burden; however, this reflects an intentional safety tradeoff. Second, schema drift in upstream systems can invalidate tools unless onboarding is automated and continuously validated. Third, prompt injection remains a persistent risk;

MCP Toolbox mitigates by enforcing policies at the boundary, but cannot eliminate social-engineering risk without additional human approval for sensitive mutations.

11 Broader impacts and safeguards

Positive impacts. A governed tool layer reduces duplicated integration work, improves auditability, and enables safer deployment of tool-using agents.

Potential negative impacts. If misconfigured, a tool layer could amplify harm by enabling rapid automated actions. Therefore, safeguards include: least-privilege policies, deny-by-default discovery, approval gates for mutations, redaction of sensitive data in logs, and anomaly detection over invocation patterns.

12 Conclusion

MCP Toolbox operationalizes MCP for production environments by pairing protocol conformance with governance: versioned tool contracts, scoped discovery, per-call authorization, sandboxed execution, and comprehensive audit/telemetry. The proposed architecture and evaluation blueprint provide a practical foundation for deploying enterprise agents that are secure, observable, and reliable under real-world failure modes.

References

References

Anthropic. Introducing the Model Context Protocol. 2024. Available at `https://docs.anthropic.com/en/docs/mcp`.

Model Context Protocol Project. Model Context Protocol (MCP) Specification. 2025. Available at `https://modelcontextprotocol.io/specification`.

OpenAPI Initiative. OpenAPI Specification. 2024. Available at `https://spec.openapis.org/oas/latest.html`.

Open Policy Agent Contributors. Open Policy Agent (OPA). 2025. Available at `https://www.openpolicyagent.org/`.

OpenTelemetry Authors. OpenTelemetry Specification. 2025. Available at `https://opentelemetry.io/docs/specs/`.

A NeurIPS paper checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?
Answer: [Yes]
Justification: The abstract and introduction state the system goal (governed MCP tool connectivity), the offline-to-online architecture, and the evaluation blueprint; these match Sections 3–9.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?
Answer: [Yes]
Justification: Limitations are discussed in the dedicated **Limitations** section, including policy complexity, schema drift, and residual prompt injection risk.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete proof?
Answer: [N/A]
Justification: The contribution is a systems architecture and evaluation blueprint; no formal theorems are presented.

4. Experimental result reproducibility

Question: Does the paper disclose all information needed to reproduce the main experimental results?
Answer: [Yes]
Justification: Section **Evaluation protocol** specifies the test suites, metrics, and reporting requirements; the tooling can be reproduced from the catalog artifacts and contract tests described in Sections 4–6.

5. Open access to data and code

Question: Does the paper provide open access to the data and code?
Answer: [N/A]
Justification: This is a project paper describing an architecture; an open-source release is optional and not required for understanding the method.

6. Experimental setting/details

Question: Does the paper specify all experimental details necessary to understand the results?
Answer: [Yes]
Justification: Section **Evaluation protocol** defines workloads, fault injection, latency percentiles, and attribution spans needed to interpret results.

7. Experiment statistical significance

Question: Does the paper report error bars or significance information?
Answer: [N/A]
Justification: The paper proposes an evaluation framework; if multiple runs are performed, standard deviation / confidence intervals can be added in reporting.

8. Experiments compute resources

Question: Does the paper provide sufficient compute resource details?
Answer: [No]

Justification: Compute specifications are not included because the paper focuses on system design; future benchmarking will report CPU/GPU, memory, and run times.

9. Code of ethics

Question: Does the research conform to the NeurIPS Code of Ethics?
Answer: [Yes]
Justification: The work is infrastructure-focused and includes safeguards for privacy, authorization, and misuse mitigation in the **Broader impacts and safeguards** section.

10. Broader impacts

Question: Does the paper discuss both positive and negative societal impacts?
Answer: [Yes]
Justification: Discussed in the **Broader impacts and safeguards** section, including misuse risks and mitigations (approval gates, least privilege).

11. Safeguards

Question: Does the paper describe safeguards for responsible release of high-risk assets?
Answer: [Yes]
Justification: Safeguards include deny-by-default discovery, least-privilege policies, approval gating, sandbox constraints, redaction, and anomaly monitoring.

12. Licenses for existing assets

Question: Are existing assets properly credited and licenses respected?
Answer: [Yes]
Justification: The paper cites MCP, OpenAPI, OAuth2, OpenTelemetry, and OPA as external specifications/assets; no proprietary datasets are redistributed.

13. New assets

Question: Are new assets introduced well documented?
Answer: [N/A]
Justification: No new dataset/model is released as part of this paper; the main asset is an architectural design and contract format.

14. Crowdsourcing and human subjects

Question: Does the paper include details for crowdsourcing or human subjects research?
Answer: [N/A]
Justification: No human subjects or crowdsourcing studies are conducted.

15. IRB approvals

Question: Does the paper describe IRB approvals or equivalent?
Answer: [N/A]
Justification: No human subjects research is conducted.

16. Declaration of LLM usage

Question: Does the paper describe usage of LLMs if important to the core methods?
Answer: [N/A]
Justification: LLMs are consumers of the tool interface; the core contribution is the MCP server architecture and governance, not a new LLM method.