A LangGraph + MCP Workbench for Neo4j-Backed Security Graphs: System Description, Constraints, and Evaluation Protocols

Industry Research Group — Secure Systems Engineering
`contact@example.org`

October 3, 2025

Abstract

This paper documents the current implementation of a graph-centric workbench for cybersecurity analysis and adversary emulation. The system uses LangGraph to orchestrate an LLM-driven agent, the Model Context Protocol (MCP) over `stdio` to invoke Neo4j tools, a FastAPI backend that exposes REST endpoints and a WebSocket for streaming agent events, and a React (Vite) frontend for visualization and control. We describe what is implemented *today* in this repository, the guardrails that actually exist (configuration gating and confirmation), the major omissions (no external policy engine, no persistent checkpoints), and how we evaluate the system via reproducible protocols and tests. We also articulate *challenges for industrialisation* in a regulated banking context (e.g., BNP Paribas) and the technical backlog required to reach production-readiness.

1 Introduction

Security teams benefit from combining graph context (assets, identities, reachability) with adversary emulation to validate controls. LLMs help translate analyst requests into Cypher queries and scenario plans, but safety and traceability matter more than convenience. This workbench aims for a practical baseline: (1) classify user intent; (2) plan and optionally confirm graph mutations; (3) execute read/write operations against Neo4j through MCP; (4) run mock or external platform simulations; (5) stream status to a web UI.

2 System Overview (as built)

2.1 Top-Level Components

Frontend (React + Vite). The UI in `frontend-vite/` connects to the backend's REST endpoints (`/tools/*`) and to a WebSocket at `/ws` for live agent/simulation events. It renders graphs (Sigma.js/Graphology) and provides a Cypher panel and scenario controls.

Backend (FastAPI). The service in `agent/backend/app/` exposes: (i) health (`/api/health`); (ii) graph endpoints (`/tools/load_graph`, `/tools/run_cypher`); (iii) simulation endpoints (`/tools/start_attack`, `/tools/check_attack`, `/tools/fetch_results`); and (iv) a WebSocket (`/ws`). Write-mode Cypher is gated by `GRAPH_ALLOW_WRITE_CYPHER=false` by default.

Agent runtime (LangGraph). The agent graph is compiled in `agent/flow.py`. Nodes include `classify_intent`, `plan_graph_action`, `confirm_graph_action`, `reject_graph_action`, `execute_graph_action`, `plan_scenario`, `execute_scenario`, `monitor_job`, `run_cypher`, `summarise_job`, and `respond`. There is no persistent checkpoint store in the current build.

MCP integration (stdio). `agent/mcp_integration.py` launches the `mcp-neo4j-cypher` server via `uvx` and loads tools with `langchain-mcp-adapters`. If MCP is unavailable, the client falls back to the Neo4j Python driver (or a mock). High-level helpers in `MCPGraphOperations` wrap tool calls such as `get-schema` and `run-cypher`.

Neo4j. A vanilla Neo4j 5.x container is defined in `docker-compose.yml`. Each MCP tool invocation runs in its own transaction at the driver level; graph loads currently iterate node/edge creation rather than one atomic multi-statement transaction.

Simulation engine. `agent/simulation_engine.py` provides a generic, evented mock engine and a Caldera-flavored adapter skeleton. Backend endpoints use this engine to start, poll, and summarize jobs. Real platform integration (Caldera, Metasploit, Atomic Red Team) is intentionally stubbed.

2.2 Data Model and Graph Operations

The working graph payloads match `agent/models.py`: nodes have `id`, optional `labels`, and arbitrary `attrs`; edges have `id`, `source`, `target`, `type`, and optional `attrs`. The backend validates payload shape, not business semantics. The MCP layer exposes:

- **Schema/query**: `get-schema()`, `run-cypher(query, params, mode)` with `mode ¯ read|write`.
- **Helpers**: `add_node`, `add_edge`, `update_node`, `update_edge`, `delete_node`, `delete_edge` implemented as parameterized Cypher templates in `MCPGraphOperations`.

Neo4j operational notes. For production, enable Bolt+TLS and role-scoped users, restrict APOC procedures, set sensible query timeouts, and add indices/constraints for IDs and common lookups. Establish regular backups and restore drills.

3 Design Choices and Implications

3.1 Intent classification and routing

Messages are classified with a schema-guided decoder (`agent/nodes/intent_classifier.py`) into: graph mutation, scenario request, status update, Cypher query, confirmation/rejection, small talk, or unknown. A simple router (`agent/nodes/router.py`) prioritizes active jobs and routes confirmations only when a pending plan exists. This keeps control flow explicit and debuggable.

3.2 Tool boundary via MCP

MCP gives a narrow, testable boundary between the agent and the database: the agent never imports a Neo4j driver directly and only calls `run-cypher` / `get-schema`. We prefer small, parameterized Cypher templates in `MCPGraphOperations` over free-form string building. When MCP is unavailable, the system degrades to a direct driver or a mock, which allows local development and deterministic tests.

3.3 LLM safety and tool use

To reduce the chance of unsafe actions:

- Use strict JSON-schema decoding for intents and Cypher plans; reject malformed outputs.
- Never interpolate untrusted text into Cypher structure; keep structure (labels/relationship types) on allowlists or generated from schema when possible.
- Require human confirmation for risky changes and prefer dry-run diffs of graph mutations.

3.4 Write safety model (as implemented)

There is no external policy engine in this repository. Instead, writes are gated in two places: (i) API-level configuration `GRAPH_ALLOW_WRITE_CYPHER` (default: `false`); (ii) agent-level confirmation for plans that set `requires_confirmation`. Basic input validation rejects malformed payloads; Cypher requests undergo minimal checks (e.g., empty queries). This is intentionally conservative but far less comprehensive than policy-as-code.

3.5 Execution and state

Each MCP call executes in its own transaction. The current `load_graph` helper iterates node then edge creation; it does not guarantee atomicity across the whole load. The LangGraph application maintains an in-memory state (`agent/state.py`); no checkpoint persistence or audit trail is implemented beyond logs and WebSocket events.

3.6 Production notes (security and privacy)

For deployment in a bank environment, plan for: single sign-on (OIDC/SAML) with role-based permissions; TLS everywhere and short-lived service credentials kept in KMS/HSM; redaction of secrets/PII in logs and traces; and regional processing or on-prem models for sensitive data.

4 Evaluation Protocols (reproducible)

We provide protocols that match the repo's current behavior; results will vary with data and configuration.

- **API smoke tests.** Use the included tests under `agent/backend/tests/` (FastAPI TestClient) to verify `/api/health`, `/tools/load_graph`, `/tools/run_cypher`, and simulation endpoints. These assert write gating and payload validation.
- **End-to-end graph flow.** Load `frontend-vite/public/sample-graph.json` via `/tools/load_graph`, run a read-only Cypher (e.g., `MATCH (n) RETURN n LIMIT 10`), and confirm UI rendering.
- **Agent routing.** Over WebSocket, submit messages that elicit each intent class and verify node transitions (classification → planning → execution).
- **Simulation loop.** Start a mock scenario via `/tools/start_attack`, poll `/tools/check_attack`, and fetch results. Confirm event stream updates in the UI.

Pre-production checks (BNP Paribas). Before go-live, validate: SSO and RBAC enforcement; policy checks on write paths with human approval for destructive changes; immutable audit storage; metrics/logs/tracing with redaction and SIEM forwarding; DR plan with backup/restore drill; container image scanning and signing.

5 Challenges for Industrialisation (BNP Paribas)

To run this workbench safely inside a bank, a small set of controls is non-negotiable. The most important items are below; related details appear throughout the paper in the design notes, protocols, and backlog.

- **Regulatory fit.** Map data flows and controls to GDPR, DORA, NIS2 and relevant EBA guidance; assign accountable owners.
- **Identity and access.** Enforce SSO via the corporate IdP and role-based permissions; require approval for destructive actions.
- **Data protection.** Minimise data, keep processing in-region, encrypt in transit and at rest, and redact sensitive fields from logs.
- **Policy and audit.** Put policy-as-code on the write path (beyond an env flag) and store tamper-evident audit records for every change.
- **Platform security.** Segment networks with mTLS, harden and scan containers, manage dependencies and secrets centrally.
- **Reliability.** Define SLOs, implement HA/DR and backup-restore drills, and apply backpressure/supervision to the MCP subprocess.
- **Observability.** Ship metrics, logs and traces (with redaction) to the bank SIEM/SOAR for alerting and correlation.
- **Operations.** Provide runbooks and on-call, follow change control, and separate dev/test/pre-prod/prod environments.

6 Current Technical Backlog (short-term)

This section lists engineering tasks intentionally left out of the MVP and complements the enterprise controls in Section *Challenges for Industrialisation*.

- **SSO and RBAC.** Integrate with the corporate IdP and enforce role-based permissions across API/WebSocket and agent actions.
- **Policy and audit store.** Add OPA/Rego policy evaluation on write paths and persist tamper-evident audit logs.
- **Atomic bulk loads.** Implement a transaction-wrapped bulk loader for `/tools/load_graph` with rollback semantics and partial-failure reporting.
- **Schema-aware validation.** Add label/type allowlists synchronised from `get-schema`, typed payload validation, and safer Cypher pre-flight checks.
- **Observability hooks.** Add metrics and tracing (e.g., OpenTelemetry) with redaction; integrate with existing logging.
- **Production adapters.** Complete Caldera adapter and add additional platform adapters (Metasploit, Atomic Red Team, Custom BNPP platform..) behind feature flags and test harnesses.
- **UI consolidation.** Remove or archive the legacy Next.js UI in `frontend/`; standardise on `frontend-vite/`.
- **Hardening defaults.** Make secure-by-default settings explicit (TLS everywhere, restricted APOC, timeouts, conservative limits) and document them.

7 Conclusion

The current workbench implements a pragmatic, testable stack: LangGraph for intent-driven orchestration; MCP (stdio) for Neo4j operations; a FastAPI service with REST+WebSocket; and a Vite/React UI. Safety today is configuration- and confirmation-based, not policy-driven. Near-term priorities are: transactional graph loads, stronger validation and schema awareness, optional policy-as-code, adapter-complete simulation, and basic observability.

References

1. Open Policy Agent (OPA) and Rego policy language. https://openpolicyagent.org/docs/.
2. OpenTelemetry semantic conventions. https://opentelemetry.io/docs/specs/otel/semantic-conventions/.
3. ISO/IEC 27001 (information security management). https://www.iso.org/standard/27001.html.
4. NIST SP 800-53 (security and privacy controls). https://csrc.nist.gov/publications/sp800-53.

5. GDPR (EU General Data Protection Regulation). https://gdpr.eu/.
6. NIS2 Directive (EU network and information security). https://digital-strategy.ec.europa.eu/.
7. DORA (EU Digital Operational Resilience Act). https://finance.ec.europa.eu/digital-finance/digital-operational-resilience-dora_en.
8. EBA Guidelines on outsourcing arrangements and ICT/security risk management. https://www.eba.europa.eu/.
9. OWASP ASVS and OWASP Top 10. https://owasp.org/www-project-application-security-
https://owasp.org/www-project-top-ten/.
10. FastAPI documentation. https://fastapi.tiangolo.com/.
11. Model Context Protocol (specification and versioning). https://modelcontextprotocol.io/specification/latest, https://spec.modelcontextprotocol.io/.
12. LangGraph documentation: state graphs and compilation. https://langchain-ai.github.io/langgraph/.
13. langchain-mcp-adapters (loading MCP tools). https://github.com/langchain-ai/langchain/tree/master/libs/mcp-adapters.
14. mcp-neo4j-cypher tool. https://github.com/modelcontextprotocol/servers/tree/main/servers/neo4j-cypher.
15. Neo4j Cypher parameters and driver transactions. https://neo4j.com/docs/cypher-manual/current/syntax/parameters/.
16. MITRE CALDERA (overview). https://caldera.mitre.org/.