SENTRY-DB: A Sovereign, Identity-Bound Streaming Gateway with Predictive Runtime Governance for LLM Agents

1st Aisvarya Sampath Kumar
Independent Researcher
Houston, Texas, USA
aisvarya20@gmail.com

2nd Mona Arora
IEEE Senior Member
Independent Researcher
Princeton, New Jersey, USA
Mona.arora@outlook.com

3rd Pravin Barapatre
IEEE Senior Member
Independent Researcher
Boston, Massachusetts, USA
pravin.barapatre@gmail.com

4th Aditya Vardhan Pola
IEEE Senior Member
Independent Researcher
Ashburn, Virginia, USA
polaaditya@ieee.org

5th Arijit Roy
Independent Researcher
Kolkata, West Bengal, India
arijit.roy.professional@gmail.com

6th Rohit Singh
IEEE Member
George Washington University
Washington, DC, USA
rohit.singh1@gwu.edu

***Abstract*—Large Language Model (LLM) agents are increasingly used to query enterprise databases on behalf of end users. Existing agent database tools have three weaknesses. They often execute queries under shared service accounts, obscuring the identity of the actual requester. They may return entire result sets in one block, increasing memory use and data exposure. They also produce inconsistent audit records that complicate compliance. Tool invocation protocols, such as the Model Context Protocol (MCP), standardize tool discovery and invocation but do not define database-specific execution, identity, streaming, or governance requirements. This paper presents SENTRY-DB, a governed streaming database architecture that addresses this gap. SENTRY-DB binds both the calling service and end-user identity to each request, enforces safety limits server-side, streams results in typed cancellable batches, and produces governance artifacts for every execution. A working gateway was implemented for PostgreSQL and Snowflake and evaluated against four baselines. The gateway delivered the first result in 48–61 ms compared with 1,870 ms for buffered approaches. When the agent required only 20 of 5,000 matching rows, data transfer decreased by 99.6%. Of 2,550 security and compliance checks, 2,545 passed (99.8%). A learned model over execution traces detected anomalous query patterns with ROC-AUC 0.91 at partial trace observation, providing a preliminary early predictive signal. Among the approaches evaluated, SENTRY-DB is the only one that satisfies all eight requirements identified for agent-safe database access.**

***Index Terms*—LLM agents, governed database access, identity delegation, streaming, authorization, AI governance, Model Context Protocol, predictive runtime governance, execution traces**

I. Introduction

LLM agents have evolved from passive assistants to autonomous systems that retrieve evidence, reason over structured data, and execute multi-step workflows with minimal supervision [1], [2]. Enterprise deployments span portfolio rebalancing, incident triage, compliance verification, and credit risk assessment. In all these areas, databases represent the highest-value and highest-risk integration boundary.

Neither database client contracts nor agent tool contracts provide a uniform, agent-safe database interface. Current solutions rely on monolithic `run_sql(query)` tools with inconsistent limits, authorization, and logging [3]. This leads to five key failure modes: shared credentials cause confused-deputy vulnerabilities [4]; agent-controlled safeguards are bypassable via prompt injection [5], [6]; non-streaming delivery increases latency and memory pressure [7]; weak governance fails SOC 2 audit requirements [8]; and lack of cancellation increases over-collection and data exposure.

Protocols such as MCP [9] standardize tool discovery but treat databases as generic tools, without requiring streaming, cancellation, identity delegation, or governance. Sovereign data-space architectures enhance access control through policy-governed usage, contract negotiation, and provenance across organizational boundaries [**?**], [10]. The DAET cycle (Discover, Authorize, Execute, Trace) summarizes these governance processes. Agentic-risk studies highlight the need for zero-trust identity and security governance in autonomous agent systems [11].

This paper introduces SENTRY-DB, a governed streaming database architecture behind MCP. Its key contributions are: (i) a four-phase safe query lifecycle (*Prepare→Execute→Stream→Close*); (ii) server-enforced safety profiles and sovereign-trust governance objects; (iii) actor-subject separation with threat analysis against OWASP and MITRE ATLAS; (iv) a PostgreSQL and Snowflake prototype evaluated against four baselines; and (v) execution-trace analytics for predictive governance. SENTRY-DB is novel in unifying these mechanisms into a single agent-safe database governance architecture rather than treating them separately.

II. Background and Related Work

Tool invocation protocols. ReAct [1], Toolformer [2], and Gorilla [12] established tool-using language models. MCP [9] standardizes discovery and invocation but treats databases as generic tools, mandating none of streaming, cancellation, identity delegation, or governance. LangChain SQL Agent [3] executes under shared credentials with inconsistent safeguards.

Database connectivity. ODBC and JDBC [13] target developers, not agents. Apache Arrow Flight [7] streams typed batches with backpressure but lacks identity or governance. Row-Level Security [14], Column-Level Security [15], and dynamic masking provide native access controls but lack agent coordination in current tools.

Identity and sovereign trust. OAuth 2.0 Token Exchange [16], OpenID Connect [17], and SPIFFE [18] provide identity primitives. The W3C PROV model [19] and SOC 2 [8] specify provenance requirements. International Data Spaces [20] provide a foundation for sovereign, policy-governed data exchange across organizational boundaries. Zero-trust extensions for agentic systems require continuous verification of identity, purpose, and capability [11], [21]. Recent work on zero-trust agentic federated learning uses hardware-backed cryptographic attestation to authenticate autonomous agents and establish verifiable trust in distributed security environments [22].

Runtime observability. Distributed tracing captures fine-grained temporal execution information and informs the runtime governance approach adopted here [23]. Complementary work on AgentSOC demonstrates how anticipatory agent reasoning and policy-compliant response planning can support adaptive containment in enterprise security operations [24].

III. Problem Statement and Requirements

Current agent-database integrations exhibit five failure modes that compound risks in enterprise deployments.

P1: Shared credential execution. Agents execute queries under a single service account with broad database permissions. End users inherit these permissions regardless of authorization level, creating confused deputy vulnerabilities [4].

P2: Inconsistent safeguards. Safety constraints, such as row limits and read-only enforcement, are usually implemented in agent-side code. Prompt injection can bypass these controls [5], [6]. Agents working with different databases also face incompatible constraint mechanisms.

P3: Non-streaming delivery. Most agent database tools buffer the entire result set before returning it. This increases latency and memory usage [7], especially for workflows that require only partial results.

P4: Weak governance. Ad hoc logging leads to inconsistent audit trails, and end-user identity is often missing from logs. These gaps weaken the evidence for SOC 2 access control and monitoring criteria [8], increasing compliance risk in multi-agent systems.

TABLE I. Problem-to-Requirement Mapping and Standards Grounding

Problem	Derived Requirement	External Standard
P1	R1: Identity preservation (actor + subject)	NIST SP 800-207 § 2.1 (continuous verification); SOC 2 CC6.1
P2	R2: Capability discovery; R3: Safe query lifecycle	OWASP LLM01 (prompt injection); SOC 2 CC6.6
P3	R4: Streaming with backpressure	OWASP LLM02 (insecure output handling); SOC 2 CC6.7
P4	R7: Authorization attestation; R8: Audit and lineage	SOC 2 CC6.1, CC7.2; EU AI Act Art. 12
P5	R5: Cancellation with deadlines	SOC 2 CC6.1; GDPR Art. 5(1)(c) data minimization
Cross-cutting	R6: Typed schemas	OWASP LLM08; SOC 2 CC7.3 (anomalies)

P5: Over-collection without cancellation. All matching rows are transferred even if the agent needs only a subset. No mechanism exists to terminate the transfer early, thereby increasing data exposure and computational costs.

Table I maps these problems to eight requirements grounded in external standards: NIST SP 800-207 zero-trust principles [21], the OWASP Top 10 for LLM Applications [6], SOC 2 trust service criteria [8], GDPR data minimization, and the EU AI Act [25]. This grounding avoids circular reasoning and enables independent verification of completeness.

IV. System Architecture

SENTRY-DB is a layered gateway between the agent runtime and backend databases. Fig. 1 illustrates the topology.

The **Agent Runtime** (1) acts as the *actor*, initiating queries for end users. Each request includes an `IdentityEnvelope` with actor and subject credentials. The optional **MCP DB Tool Wrapper** (2) converts MCP invocations into SENTRY-DB messages to maintain compatibility with existing agent frameworks. The **SENTRY-DB Gateway** (3) serves as the Policy Enforcement Point. It authenticates identities, authorizes requests via the Policy Engine, enforces the `SafetyProfile`, and determines the execution mode.

When multiple policies impose conflicting constraints, SENTRY-DB resolves them conservatively. Deny decisions take precedence over allow decisions, and limits use the strictest value. The resolved policy set and versions are recorded in the authorization attestation, enabling reconstruction of enforcement decisions during audit. This process establishes a sovereign handshake that verifies identity, policy, and execution constraints before execution, drawing on dataspace contract negotiation and usage control principles [**?**], [10].

The **Database Adapters** (4) translate the SENTRY-DB lifecycle into backend-native semantics (SQL, Cypher, MQL). The **Policy Engine** (5) enforces authorization through OPA [26]. The **Identity Provider** (6) supports OAuth 2.0 token exchange [16] and validates actor identity with SPIFFE SVIDs [18]. The **Trace Analyzer** (7) processes runtime telemetry, encoding query patterns, latency, and anomaly signals into predictive features to detect probing and planning

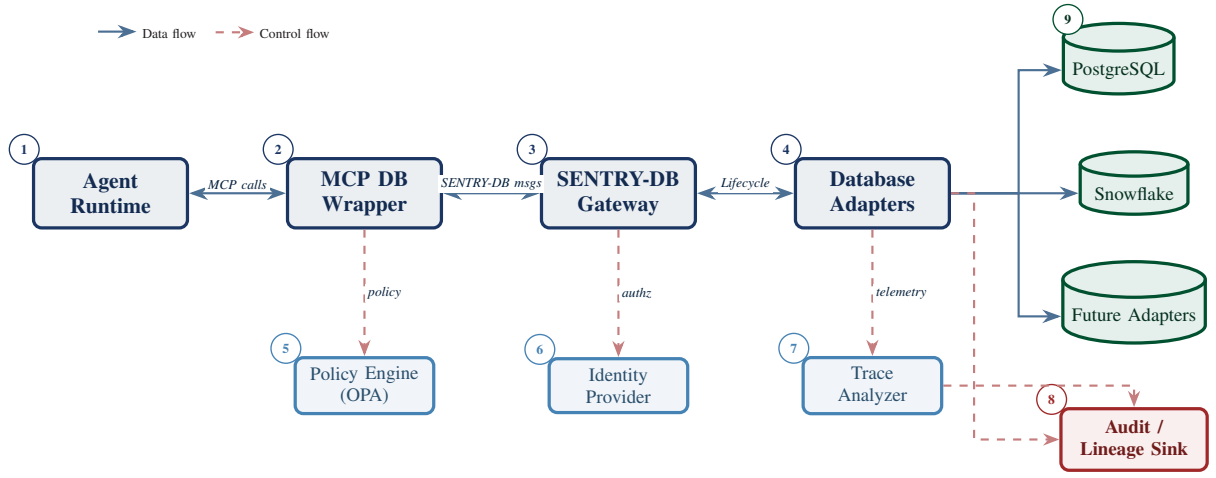


Fig. 1. SENTRY-DB system architecture. The Agent Runtime (1) sends identity-bound requests through the MCP Wrapper (2) to the SENTRY-DB Gateway (3), which enforces policy via the Policy Engine (5), authenticates through the Identity Provider (6), and routes queries through Database Adapters (4) to backend databases (9). The Trace Analyzer (7) processes runtime telemetry. All governance events flow to the Audit Sink (8).

TABLE II. REQUIREMENTS SATISFACTION ACROSS APPROACHES

Requirement	MCP	LangCh.	Arrow	JDBC	SENTRY-DB
R1: Identity pres.	○	×	×	○	●
R2: Cap. discovery	●	○	○	○	●
R3: Safe lifecycle	×	×	○	○	●
R4: Streaming	×	×	●	○	●
R5: Cancellation	×	×	○	○	●
R6: Typed schemas	○	○	●	●	●
R7: Attestation	×	×	×	×	●
R8: Audit + lineage	○	○	×	○	●
Score (of 8)	2	0	3	1	**8**

●=Full, ○=Partial, ×=None.
Table XII extends this comparison with the DAET baseline.

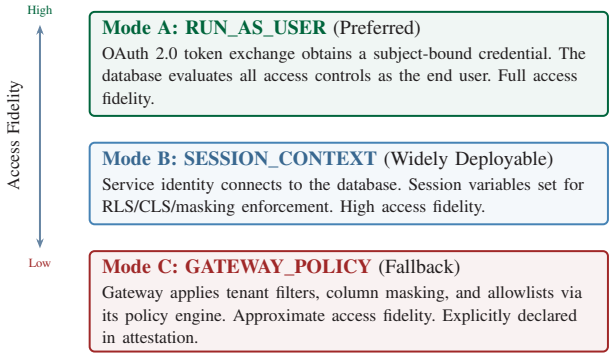


Fig. 2. SENTRY-DB execution modes ordered by access fidelity. Each mode is explicitly declared in the `AuthorizationAttestation`, enabling verifiable governance regardless of enforcement mechanism.

loops. The **Audit/Lineage Sink** (8) collects governance events for SIEM, compliance, and forensic analysis.

V. PROTOCOL DESIGN

A. Capability Discovery and Safe Query Lifecycle

The `Capabilities()` call returns supported query modules, wire encodings, server-enforced maxima, and available enforcement modes, aligning with the *Discover* phase of the DAET cycle defined earlier. SENTRY-DB replaces monolithic `run_sql` with a four-phase lifecycle: Prepare, Execute, StreamFetch, and Cancel/Close.

Prepare. The gateway validates and classifies the query, enforces limits, and returns an `AuthorizationAttestation` with a `statement_id` (the *Authorize* step of DAET). **Execute.** The gateway initiates the query using the selected execution mode and identity context, returning a `cursor_id`. **StreamFetch.** Typed record batches stream with backpressure, each carrying schema, payload, and incremental lineage. The `SafetyProfile` continuously enforces cumulative row, byte, and time limits. **Cancel/Close.** Resources are released, lineage finalized, and a complete audit event is sent to the Audit/Lineage Sink for trace analytics.

B. Identity Model and Execution Modes

Every request includes an `IdentityEnvelope` carrying both the *actor* (service identity via SPIFFE [18]) and the *subject* (end user via OIDC [17]). No agent entity is trusted by default; trust is established through cryptographic credentials and auditable purpose declarations [11], [21]. The gateway enforces exactly one of three execution modes (Fig. 2).

C. Governance Metadata and Wire Format

Every execution produces two governance artifacts. The **Authorization Attestation** records the effective subject, tenant ID, roles and scopes, enforcement mode, versioned policy IDs, timestamp, and correlation ID, providing a verifiable record of the authorization decision. The **Lineage Record** captures

TABLE III. FEATURE COMPARISON WITH EXISTING APPROACHES

Dimension	MCP Tool	LangChain SQL	Arrow Flight	SENTRY-DB
Scope	Generic tools	SQL only	Data transfer	Database-agnostic
Lifecycle	Not defined	Single call	Stream only	4-phase
Identity	Optional ctx	Shared creds	None	Actor/subject
Safety	Tool-specific	Agent-side	None	Server-enforced
Streaming	Not required	No	Native	Native+ governance
Governance	Tool logs	Ad hoc	None	Attest.+ lineage
Cancellation	No	No	Partial	First-class
Trace analytics	No	No	No	Integrated

resources touched, cumulative rows and bytes, elapsed time, and correlation ID, supporting SOC 2 audit evidence [8] and feeding trace analytics. Table III compares SENTRY-DB with existing approaches across eight dimensions.

The SENTRY-DB schema uses Protocol Buffers. Fig. 3 shows a JSON-like logical schema for a Prepare request, attestation response, and StreamFetch batch under SESSION_CONTEXT. The wire format is binary Protobuf; the listing shows field structure without serialization details. The `correlation_id` links messages for audit reconstruction.

VI. SECURITY MODEL AND GOVERNANCE

A. Threat Model

SENTRY-DB treats the agent as untrusted. The threat model draws from OWASP Top 10 for LLM Applications [6], MITRE ATLAS [27], and systemic risks identified in multi-agent deployments [11].

T1: Prompt injection (OWASP LLM01, ATLAS AML.T0051). A compromised agent submits malicious queries. Mitigation: server-enforced `SafetyProfile` and Prepare-phase validation.

T2: Confused deputy (CWE-441). Service-account permissions are inherited by all end users. Mitigation: mandatory actor/subject binding.

T3: Cross-tenant leakage. Query returns rows from another tenant. Mitigation: `tenant_id` validation with RLS, mode declared in attestation.

T4: Over-collection. All matching rows transferred when a subset suffices. Mitigation: server-enforced `max_rows`, `max_bytes`, `timeout_ms`, and first-class cancellation.

T5: Inference via repeated queries. Benign-looking queries extract sensitive data over time. Mitigation: lineage tracking with correlation IDs and Trace Analyzer probing detection.

T6: Cascading failures. One agent's failure propagates through a workflow. Mitigation: gateway circuit breakers triggered by anomalous runtime patterns.

```
// Prepare request
{
  "identity_envelope": {
    "actor_id": "spiffe://corp/agent/watchtower",
    "subject_token": "eyJhbGc...",
    "tenant_id": "T1",
    "roles": ["sre_viewer"]
  },
  "safety_profile": {
    "read_only": true, "max_rows": 200,
    "max_bytes": 2097152, "timeout_ms": 2000
  },
  "query": { "dialect": "postgresql",
    "text": "SELECT id, severity FROM incidents
           WHERE tenant_id=$1 LIMIT 200",
    "params": ["T1"] }
}
```

```
// Prepare response with attestation
{
  "statement_id": "stmt-42",
  "attestation": {
    "effective_subject": "alice@corp.com",
    "tenant_id": "T1", "roles": ["sre_viewer"],
    "enforcement_mode": "SESSION_CONTEXT",
    "policy_ids": ["rls-incidents-v3", "mask-pii-v2"],
    "correlation_id": "corr-a8f2c1",
    "signature": "MEUCIQD..."
  }
}
```

```
// StreamFetch batch response
{
  "cursor_id": "cur-99",
  "schema": [{"name":"id","type":"int64"},
             {"name":"severity","type":"string"}],
  "payload": [[1001,"SEV1"], [1002,"SEV1"]],
  "lineage": {
    "resources": ["public.incidents"],
    "rows": 10, "bytes": 4200, "elapsed_ms": 45,
    "correlation_id": "corr-a8f2c1"
  }
}
```

Fig. 3. Example SENTRY-DB wire messages. Prepare request, Prepare response with attestation, and one StreamFetch batch.

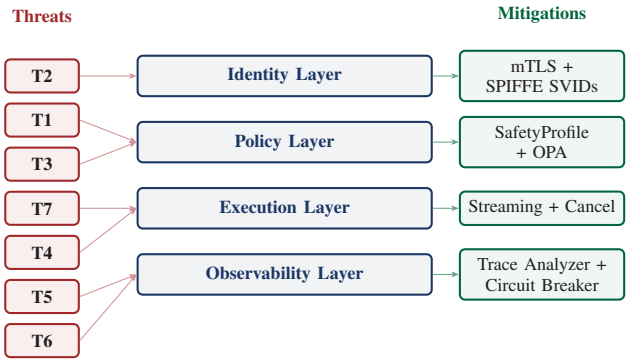


Fig. 4. Threat-to-mitigation mapping across SENTRY-DB defense layers. Each threat (T1–T7) is intercepted by one or more layers before reaching the database.

T7: Agentic drift. The agent diverges from its authorized scope. Mitigation: pre-execution validation of identity, intent, scope, and policy constraints.

Fig. 4 shows the threat-to-mitigation mapping across four defense layers.

TABLE IV. SECURITY INVARIANTS BY EXECUTION MODE

Property	RUN_AS_USER	SESSION_CTX	GW_POLICY
I1: Identity bind	Strict	Strict	Strict
I2: SafetyProfile	Strict (GW+DB)	Strict (GW+DB)	Strict (GW only)
I3: Mode declared	Strict	Strict	Strict
I4: Attestation	Strict	Strict	Strict
I5: Cancel	Strict	Strict	Strict
Access fidelity	Full	High (RLS/CLS)	Approximate
Assumptions	DB accepts delegated tokens	DB supports session vars + RLS	GW has complete policy set
Failure behavior	Auth error (fail-closed)	RLS not applied (fail-open risk)	Incomplete filtering (fail-open risk)

TABLE V. SENTRY-DB FEATURES AND COMPLIANCE CONTROLS

SENTRY-DB Feature	SOC 2 Control	EU AI Act / Other
Actor/subject binding	CC6.1 (logical access)	Art. 14 (human oversight)
SafetyProfile	CC6.6 (system boundaries)	Art. 15 (accuracy, robustness)
Authorization Attestation	CC6.1, CC7.2	Art. 12 (record-keeping)
Lineage Record	CC7.2 (monitoring), A1.2	Art. 12, Art. 13 (transparency)
mTLS + SPIFFE	CC6.7 (transmission)	GDPR Art. 32 (security)
Cancellation + limits	CC6.1, CC6.7	Art. 10 (data governance)
Trace Analyzer	CC7.3 (anomalies)	Art. 73 (incident reporting)
Replay protection	CC6.8 (prevention)	GDPR Art. 32

B. Security Invariants

Five invariants hold across all SENTRY-DB executions:

I1: Every execution binds a verified actor and subject.
I2: The `SafetyProfile` is gateway-enforced, never agent-delegated.
I3: Enforcement mode is explicitly declared, not inferred.
I4: Attestation and lineage accompany every execution and every streamed batch.
I5: Cancellation is first-class, releasing resources atomically with lineage finalization.

Table IV details enforcement strength per execution mode.

C. Transport Security and Compliance

Each `AuthorizationAttestation` includes `effective_subject`, `tenant_id`, `roles_scopes`, `enforcement_mode`, versioned `policy_ids`, a timestamp, and `correlation_id`. Execution is verified when the attestation contains a valid correlation ID, the declared mode matches gateway configuration, policy identifiers reference active policies, and lineage remains within `SafetyProfile` limits. SENTRY-DB enforces mTLS across all components. Actor identity is validated via SPIFFE SVID [18], and attestation objects are signed by the gateway. The Audit Sink rejects duplicate `correlation_id` values for replay protection. Trust-bundle drift during CA rotation is a known cause of mTLS outages in distributed deployments. SENTRY-DB operators should apply the same rigor to gateway CA material.

SENTRY-DB features provide technical controls and audit evidence that align with selected regulatory and assurance requirements. Table V maps these mechanisms to relevant SOC 2 trust service criteria [8], EU AI Act articles [25], and applicable GDPR provisions. This mapping indicates how SENTRY-DB can support compliance evidence, but it does not establish certification or regulatory compliance. Execution attestations and lineage records provide evidence relevant to SOC 2 CC6.1 and CC7.2, while Trace Analyzer outputs can assist with EU AI Act Article 73 incident reporting through timestamped anomaly assessments tied to agent and subject identifiers.

D. Failure Modes and High Availability

The gateway is on the request path and, without additional deployment measures, is a single point of failure. Four failure classes are identified, each with prescribed responses.

- **Gateway unavailable.** SENTRY-DB operates as a stateless replica set behind a load balancer with `/ready` health probes. Clients retry with exponential backoff, using the `correlation_id` as an idempotency key. The Audit Sink deduplicates retried executions on that key.
- **Policy engine (OPA) unavailable.** By default, the gateway fails closed. Requests are rejected with `POLICY_UNAVAILABLE` instead of using stale policy. An optional bounded-staleness cache (60s TTL, signed by the policy controller) trades strict freshness for availability. The mode used is recorded in the attestation.
- **SESSION_CONTEXT fail-open.** If the backend accepts session variables without the expected RLS policy, tenant filtering may fail silently. Three controls mitigate this risk: (i) a startup-time schema probe that rejects `SESSION_CONTEXT` for tables lacking the expected RLS policy, (ii) per-query canary rows that verify `tenant_id` scoping, and (iii) a deployment policy that enforces `GATEWAY_POLICY` fallback if either check fails. The attestation records the applied mode. Residual risk remains if authorization state changes after startup validation or canary checks miss a misconfiguration. These controls reduce, but do not eliminate, SESSION_CONTEXT fail-open risk.

E. End-to-End Security and Governance Example

An SRE asks an agent: *"Show the top 20 Sev-1 incidents for tenant T1 in the last 24 hours."* Fig. 5 traces the resulting exchange; four properties are guaranteed by construction: no agent credential exposure, cancellation-bounded data transfer, per-phase verifiable artifacts linked by `correlation_id`, and normal classification by the Trace Analyzer.

VII. PROTOTYPE IMPLEMENTATION

Gateway. The gateway is implemented in Python 3.11 with `asyncio` and consists of four modules: a protocol layer (request parsing, schema validation, attestation signing), policy-enforcement middleware (identity verification,

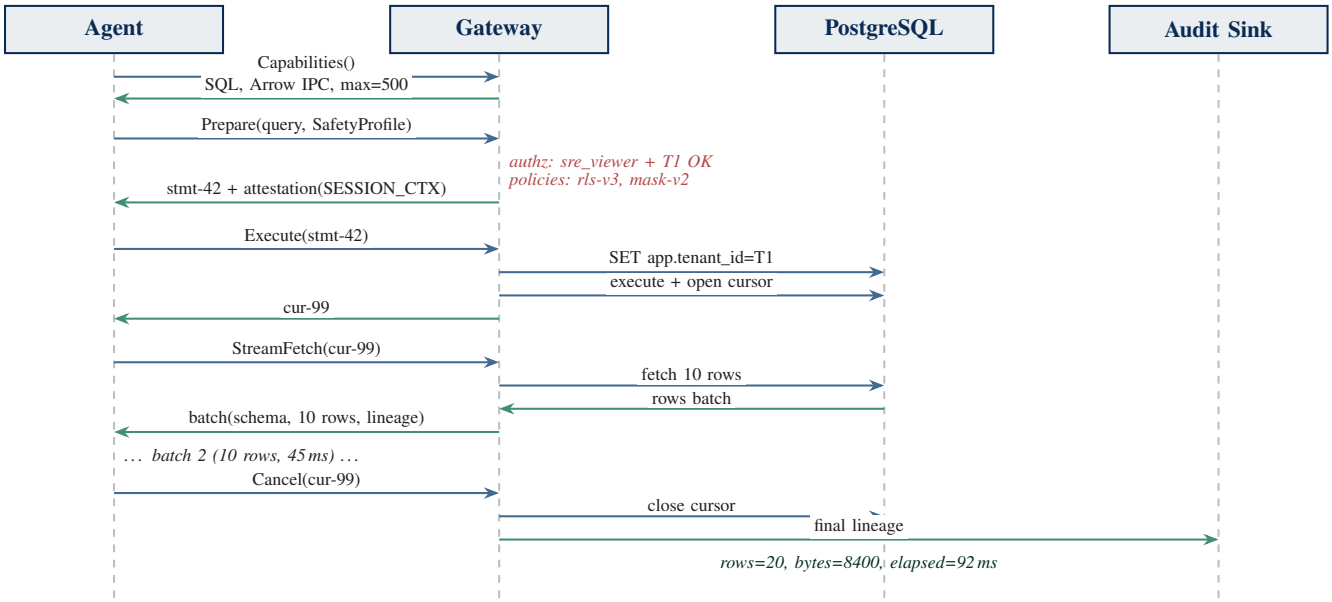


Fig. 5. SRE scenario with capability discovery and the four-phase lifecycle across agent, gateway, database, and audit sink. Blue arrows are requests, green are responses, and dashed lines are lifelines. Gateway actions appear beside its lifeline. Final lineage reaches the Audit Sink on Cancel.

`SafetyProfile` enforcement, mode selection), a database adapter interface, and a trace emitter. gRPC is the transport and Protocol Buffers define the SENTRY-DB schema. All lifecycle phases enforce the `SafetyProfile` server-side through middleware. Open Policy Agent (OPA) [26] runs as a sidecar, with Rego policies versioned in Git. Policy evaluation adds 2.1 ms overhead per request.

Adapters. The *PostgreSQL* adapter uses `asyncpg` with a 20-connection pool. For RUN_AS_USER, the validated subject identity maps to a PostgreSQL role and is applied through `SET ROLE`. SESSION_CONTEXT uses `SET app.tenant_id` for RLS [14]. Data is streamed using server-side cursors with a default batch size of 50. The *Snowflake* adapter uses the Snowflake Python Connector with session-parameter injection for RLS, supports SESSION_CONTEXT, and streams data via chunked fetch. In GATEWAY_POLICY mode, tenant filtering and column masking are enforced at the gateway.

Identity and Traces. SPIFFE SVIDs validate actor identity, OIDC tokens validate subject identity [17], and token exchange follows RFC 8693 [16]. A local SPIRE server issues workload identities, and Keycloak serves as the OIDC provider. The Trace Analyzer collects per-phase events (agent ID, action, query hash, latency, row count, correlation ID) and uses an XGBoost [28] model to score anomalies after each StreamFetch batch.

Observability. The gateway emits Prometheus metrics (request rate per mode, phase latency histograms, OPA and token-exchange latency, cursor count, violation counters), structured JSON logs per phase (PII redacted, subject tokens hashed), and OpenTelemetry spans. Operational telemetry flows to the SRE stack, while execution traces flow to the Trace Analyzer.

Test Environment. Tests ran on an AMD EPYC 7763 host (8 vCPUs, 32 GB RAM) with Ubuntu 22.04 LTS. Backends included PostgreSQL 16.2 (co-located, < 0.5 ms) and Snowflake enterprise (US-East, 15–40 ms). Workloads were driven by a LangChain ReAct agent with GPT-4o. The gateway, OPA, SPIRE, and Keycloak operated under Docker Compose.

VIII. Experimental Evaluation

The prototype was evaluated across latency, memory usage, concurrent throughput, batch-size sensitivity, per-mode overhead, security enforcement, and anomaly detection. Four baselines were compared: **MCP-style `run_sql`** (buffered), **Direct `asyncpg fetch()`** (buffered, no governance), **Arrow Flight** [7] (streaming, no governance), and a **governed non-streaming connector** (identity and policy enforcement with attestation and audit, but buffered results). Latency, memory, and throughput experiments were repeated 30 times, with results reported as mean $\pm$ standard deviation. Latency distributions showed no significant departure from normality under the Shapiro–Wilk test ($p > 0.05$). Security and ablation experiments report pass/fail counts with Wilson 95% binomial confidence intervals where applicable.

A. Latency, Memory, and Data Transfer

Four workloads were run against a 50,000-row PostgreSQL incidents table: Top-10, Top-50, Top-100, and Full Scan. The primary latency metric is time to first evidence (TTFE). Fig. 6 shows that SENTRY-DB delivers the first batch in 48–61 ms, comparable to Arrow Flight (45–55 ms). The 3–6 ms overhead reflects identity validation, OPA evaluation, and attestation

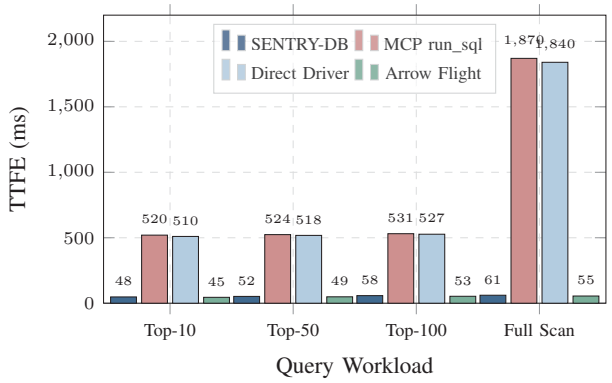


Fig. 6. Time to first evidence across query workloads (PostgreSQL 16.2, batch size 50, mean of 30 runs). SENTRY-DB and Arrow Flight deliver the first batch in under 65 ms, while buffered approaches block until the full result set is materialized.

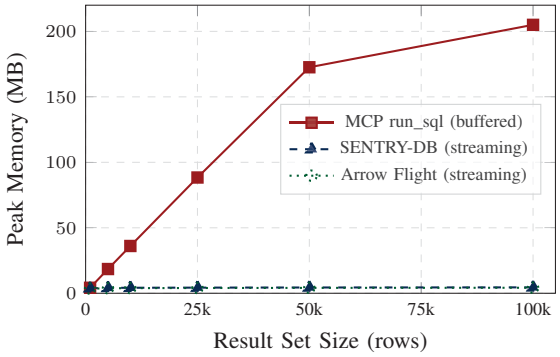


Fig. 7. Peak agent-side memory versus result set size (PostgreSQL 16.2, batch size 50, mean of 30 runs). Buffered approaches grow linearly; SENTRY-DB streaming holds constant at ∼4.4 MB, bounded by the batch buffer.

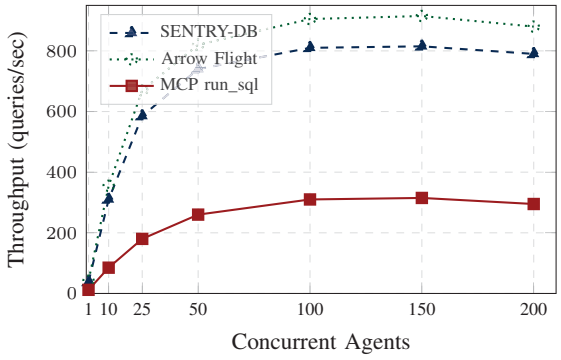


Fig. 8. Sustained throughput under concurrent agent load (PostgreSQL 16.2, Top-100 workload, batch size 50, 8-vCPU gateway). SENTRY-DB reaches 89% of Arrow Flight throughput with full governance; throughput saturates above 100–150 concurrent agents, while MCP `run_sql` saturates earlier under buffered execution.

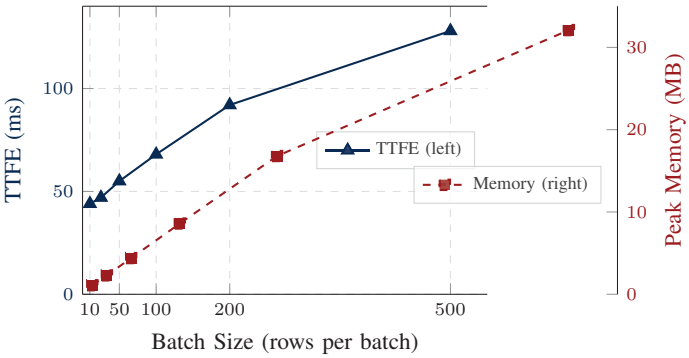


Fig. 9. Batch-size sensitivity (PostgreSQL 16.2, 50k-row table, Top-100 workload, 30 runs). A batch size of 50 balances TTFE and memory use; smaller batches of 10–25 further reduce TTFE and memory.

generation; Table VII provides the per-mode breakdown. On Full Scan, buffered baselines block for 1,870 ms, yielding about 30× higher TTFE.

When the agent cancels after 20 rows from a 5,000-row result set, SENTRY-DB transfers 8.4 KB versus 2,100 KB for buffered baselines, a 99.6% reduction. Fig. 7 shows that peak agent memory grows with result size for buffered approaches, reaching 172 MB at 50,000 rows, while SENTRY-DB remains bounded at 4.4 MB by the batch buffer.

The same workloads were evaluated against the Snowflake backend in the US-East region, with typical round-trip latency of 15–40 ms. TTFE increases to 68–88 ms, while gateway overhead remains 3.9 ± 1.0 ms, statistically indistinguishable from the co-located PostgreSQL case (3.8 ± 0.9 ms; Mann–Whitney U, p=0.71). Peak memory remains bounded at 4.5 ± 0.2 MB. These results indicate that gateway overhead and memory usage remain consistent across both backends.

B. Concurrency, Batch Size, and Per-Mode Overhead

Fig. 8 shows sustained throughput as concurrent agents scale from 1 to 200. SENTRY-DB reaches 810 queries/sec at 100 agents, within 89% of Arrow Flight's performance. When concurrent agents exceed 150, SENTRY-DB throughput plateaus, indicating gateway-side resource saturation. MCP `run_sql` plateaus earlier, reaching approximately 315 queries/sec under buffered execution.

Fig. 9 illustrates that a batch size of 50 keeps TTFE and memory low; smaller batches of 10–25 reduce both further.

Table VII breaks overhead down per mode. SESSION_CONTEXT adds 17% overhead relative to Direct Driver, while RUN_AS_USER and GATEWAY_POLICY add 38% and 34%, respectively.

C. Security, Compliance, and Threat Verification

Eleven security and compliance scenarios evaluated the threats and governance properties defined in Section VI. Table VIII reports raw block rates. Identity tests (missing token, expired SVID, replay) blocked 100%. SafetyProfile violations (write under read_only, cross-tenant, row/byte limits) also blocked 100%. Timeout enforcement reached 98.7%, with 2/150 trials terminating 50–120 ms past the deadline. SQL injection was blocked in 297 of 300 trials (99.0%). The remaining three passed Prepare-phase validation, but downstream RLS still confined them to the authorized tenant scope.

TABLE VI. LATENCY AND DATA TRANSFER RESULTS (30 RUNS PER CONFIGURATION)

Metric	MCP (PG)	Direct (PG)	Arrow Flight (PG)	SENTRY-DB (PG)	SENTRY-DB (SF)
TTFE Top-10 (ms)	520 ± 22	510 ± 19	45 ± 3	48 ± 4	68 ± 7
TTFE Top-100 (ms)	531 ± 24	527 ± 21	53 ± 3	58 ± 4	82 ± 9
TTFE Full Scan (ms)	1870 ± 96	1840 ± 88	55 ± 3	61 ± 4	88 ± 10
Rows transferred (of 5000)	5000	5000	5000	20	20
Bytes transferred (KB)	2100	2100	2100	8.4	8.4
Peak agent memory (MB)	172.6 ± 3	168.3 ± 3	4.1 ± 0.2	4.4 ± 0.2	4.5 ± 0.2
Gateway overhead (ms)	—	—	—	3.8 ± 0.9	3.9 ± 1.0

Values are mean $\pm$ std. dev. Rows transferred measured when the agent cancels after 20 of 5000 matching rows. Peak memory is measured agent-side on a 50k-row Full Scan. Snowflake TTFE is higher because of cloud network latency, not gateway overhead: the gateway contribution stays within 3.8–3.9 ms across both backends.

TABLE VII. PER-MODE GATEWAY OVERHEAD BREAKDOWN (MEAN $\pm$ STD. DEV.).

Phase	RUN_AS_USER	SESSION_CTX	GW_POLICY
Identity valid. (ms)	1.8 ± 0.4	1.2 ± 0.3	1.2 ± 0.3
Policy eval (ms)	2.1 ± 0.6	2.1 ± 0.5	2.4 ± 0.6
Token exchange (ms)	4.3 ± 1.1	0.0	0.0
Attestation (ms)	0.5 ± 0.1	0.5 ± 0.1	0.5 ± 0.1
Filter rewriting (ms)	0.0	0.0	3.6 ± 0.9
Total overhead (ms)	8.7 ± 1.3	3.8 ± 0.7	7.7 ± 1.1
vs. Direct Driver	+38%	+17%	+34%

Token exchange applies only when RUN_AS_USER delegates to the DB. Filter rewriting applies only when the gateway enforces tenant/mask policy.

TABLE VIII. SECURITY AND COMPLIANCE TEST RESULTS

Test Scenario	Trials	Pass	Rate (%)
Write attempt under read_only	200	200	100.0
Cross-tenant query (T3)	200	200	100.0
Row limit exceeded	200	200	100.0
Byte limit exceeded	200	200	100.0
Timeout enforcement	150	148	98.7
Prompt injection via SQL	300	297	99.0
Missing subject token	100	100	100.0
Expired actor SVID	100	100	100.0
Replay (duplicate corr. ID)	100	100	100.0
Attestation completeness	500	500	100.0
Lineage completeness	500	500	100.0
Aggregate	**2550**	**2545**	**99.8**

This reflects defense in depth, with gateway validation and database-native authorization as independent protection layers.

Of the 2,550 checks in Table VIII, 1,550 evaluated direct security and safety enforcement, with 1,545 passing (99.7%). The remaining 1,000 checks evaluated governance-artifact completeness, comprising 500 attestation and 500 lineage checks, all of which passed (100%). Overall, 2,545 of 2,550 checks passed (99.8%), reflecting both enforcement and governance-artifact completeness outcomes.

Table IX maps each threat to its verifying test. Direct-enforcement threats (T1–T4, T7) achieve 99.3–100% block rates. Detection-based threats (T5, T6) reach 88.0% and 92.5% at 60% trace prefix length with a 5% false-positive ceiling.

TABLE IX. THREAT-TO-TEST VERIFICATION MATRIX

Threat	Test Scenario	Defense Layer	Rate
T1: Prompt injection	SQL injection, write under read_only	Policy + Exec.	99.3%
T2: Confused deputy	Missing subject token, expired SVID	Identity	100.0%
T3: Cross-tenant leak	Cross-tenant query	Policy	100.0%
T4: Over-collection	Row/byte/timeout limits	Execution	99.6%
T5: Inference probing	Repeated filter variants	Observability	88.0%
T6: Cascading failure	Recursive loop injection	Observability	92.5%
T7: Agentic drift	Out-of-scope action attempt	Policy	100.0%
Aggregate (direct enforcement threats)			99.8%
Aggregate (detection-based threats)			90.3%

Detection-based threats (T5, T6) use Trace Analyzer outputs at 60% trace prefix length. Rates are block rate for T1–T4, T7 and detection rate at 5% false-positive ceiling for T5, T6.

D. Predictive Anomaly Detection

The Trace Analyzer was evaluated on 2,000 prototype execution traces (1,500 training, 500 test) under mixed workloads. Per-phase events described in Section VII were aggregated into behavioral features: cumulative latency, query-hash repetition, and returned rows relative to the configured SafetyProfile limit. The test set comprised 18% anomalous and 82% benign traces. A majority-class baseline yielded ROC-AUC 0.50. Labels covered three patterns: **inference probing** (repeated queries with incremental filter changes), **boundary probing** (queries exceeding 80% of SafetyProfile limits), and **planning loops** (recursive query patterns).

Two weak baselines contextualize the supervised models: a fixed threshold on the rows-to-limit ratio and an unsupervised Isolation Forest [29] using the same features. Table X presents the results. Gradient Boosted Trees at 80% trace prefix achieved a ROC-AUC of 0.91 (precision 0.85, recall 0.83), outperforming Isolation Forest by 0.12 and the threshold baseline by 0.29. Fig. 10 shows predictive signal at 30% completion (AUC 0.72), rising to 0.84 at 50%, indicating potential for early runtime containment. The three behavioral features above account for 61% of total feature importance. Because the traces come from a single prototype environment, these results represent an upper bound. Generalization is discussed in Section IX.

TABLE X. ANOMALY DETECTION PERFORMANCE

Model	Prec.	Recall	F1	ROC-AUC
Majority baseline	0.00	0.00	0.00	0.50
Threshold on rows/limit	0.58	0.42	0.49	0.62
Isolation Forest [29]	0.71	0.64	0.67	0.79
Logistic Regression	0.69	0.65	0.67	0.76
Random Forest	0.81	0.78	0.79	0.88
Gradient Boosted Trees	**0.85**	**0.83**	**0.84**	**0.91**

Evaluated at 80% trace prefix length. Class prior is 18% positive.
Threshold baseline flags traces exceeding 80% of `max_rows`.
Isolation Forest and Gradient Boosted Trees use the same features.

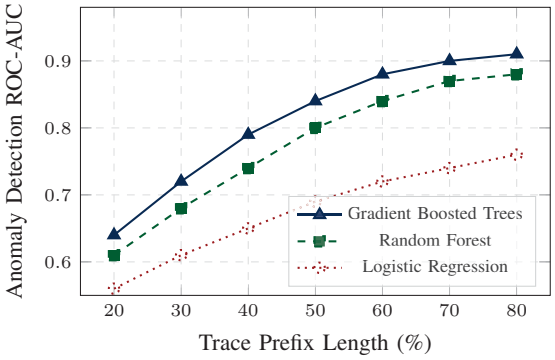


Fig. 10. Anomaly detection AUC versus trace prefix length (2,000 traces, 18% anomalous). AUC reaches 0.84 at 50% completion and 0.91 at 80%, enabling early containment.

TABLE XI. ABLATION STUDY: CONTRIBUTION OF EACH COMPONENT

Configuration	TTFE (ms)	Sec. (%)	Audit	Detect
Full SENTRY-DB	58 ± 4	99.8	Full	Full
− Trace Analyzer	57 ± 4	99.8	Full	None
− SafetyProfile	54 ± 3	36.2	Full	Full
− Attestation signing	57 ± 4	99.8	Partial	Full
− Identity binding	51 ± 3	18.4	Broken	Full
− Streaming / Cancel	$1{,}840 \pm 96$	99.8	Full	Full
Baseline (Direct Driver)	527 ± 21	0.0	None	None

Sec. % = aggregate security enforcement rate across 11 scenarios.
TTFE measured on Top-100 workload over 30 runs,
except the Streaming/Cancel row, which uses Full Scan.

TABLE XII. REQUIREMENTS SATISFACTION ACROSS APPROACHES

Requirement	MCP	LangCh.	Arrow	JDBC	DAET	SENTRY-DB
R1: Identity pres.	○	×	×	○	●	●
R2: Cap. discovery	●	○	○	○	●	●
R3: Safe lifecycle	×	×	○	○	○	●
R4: Streaming	×	×	●	○	×	●
R5: Cancellation	×	×	○	○	×	●
R6: Typed schemas	○	○	●	●	×	●
R7: Attestation	×	×	×	×	●	●
R8: Audit + lineage	○	○	×	○	●	●

●=Full, ○=Partial, ×=None.
DAET = dataspace-governance baseline derived from [**?**], [10].

E. Ablation and Comparative Analysis

Table XI shows the contribution of each component. Removing the SafetyProfile drops security from 99.8% to 36.2% while saving only 4 ms. Removing identity binding drops security to 18.4%. Removing streaming raises TTFE to 1,840 ms. Removing the Trace Analyzer leaves direct enforcement intact but eliminates detection of T5 and T6.

Table XII compares SENTRY-DB against five reference approaches across the eight requirements from Section III. SENTRY-DB is the only approach meeting all eight. A dataspace-governance baseline derived from the Dataspace Protocol and IDS usage-control principles [**?**], [10] scores 4/8 (strong governance, no streaming or cancellation). Arrow Flight [7] provides streaming without identity or governance. The Agentic Trust Framework [11] adds attestation but not database-specific lifecycle or streaming semantics.

IX. DISCUSSION

Key Findings. Streaming with cancellation improves agent-database efficiency. When only part of the results is needed, early cancellation can substantially reduce data transfer and exposure by 99.6% in the evaluated workload. Buffered baselines are structurally disadvantaged on TTFE and memory, and Arrow Flight does not address governance. This comparison therefore isolates the governance cost in streaming, which is about 3.8 ms per request, rather than claiming general superiority. Partial-trace anomaly detection achieves a ROC-AUC of 0.84 at 50% completion, providing an early signal for intervention before a suspicious sequence completes.

Limitations. Although the adapter abstraction is database-agnostic, the prototype validates only PostgreSQL and Snowflake. All experiments were conducted on a single 8-vCPU host with single-table workloads. Multi-replica deployments and multi-query agent workflows with joins and dependent queries were not evaluated. The SPIRE + Keycloak identity setup increases deployment complexity, consistent with zero-trust requirements [21]. GATEWAY_POLICY offers only approximate access fidelity, and its filter rewriting was not tested on nested subqueries, recursive CTEs, or window functions. Lineage tracking does not implement differential privacy, so the Trace Analyzer identifies inference patterns without formal bounds. Security testing used controlled attack patterns, which may not reflect production adversarial diversity. The anomaly model was trained on 1,500 prototype traces and may require retraining under different agent architectures or query distributions.

Future Work. Implementing signed attestations with JWS [30] would enable third-party audits without gateway access. Adding differential privacy to aggregate query controls would establish formal inference bounds. Introducing a parameterized-query-only mode or AST-level validation could close the Prepare-phase gap observed in 3 of 300 SQL injection trials. Developing a conformance test suite would

help database vendors verify SENTRY-DB compatibility. Integrating HSMs could provide hardware-backed protection for attestation-signing keys. Online learning would allow the Trace Analyzer to adapt to evolving agent behavior without full retraining. Additional graph and document-store adapters would extend validation beyond the relational and data-warehouse backends evaluated here.

X. Conclusion

This paper presented SENTRY-DB, a protocol for governed database access by LLM agents. The design addresses problems that current agent database tools do not handle well. Every query carries both the calling service and the end user. Safety limits are enforced by the server, not the agent. Results are streamed in small batches that the agent can stop at any point. A signed governance record is produced for every query.

A working gateway was built for PostgreSQL and Snowflake and compared against four existing approaches. The first result arrives in about 50 ms instead of waiting for the full result set. When an agent needs only a small portion of data, transfer drops by 99.6%. Across 2,550 security and compliance checks, 99.8% passed. A model over execution traces detected anomalous query patterns before trace completion, providing a preliminary early predictive signal.

Among the approaches compared, SENTRY-DB is the only one that meets all eight requirements identified for safe agent database access. It does not replace the Model Context Protocol. It sits behind it and handles the parts that MCP was never designed to handle. As agents take on more autonomous work inside organizations, dedicated database governance for agents warrants further study.

References

[1] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023. [Online]. Available: https://openreview.net/forum?id=WE_vluYUL-X

[2] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 68 539–68 551.

[3] LangChain, "Build a SQL agent," LangChain, 2026, langChain Documentation. [Online]. Available: https://docs.langchain.com/oss/python/langchain/sql-agent

[4] N. Hardy, "The confused deputy: (or why capabilities might have been invented)," *ACM SIGOPS Operating Systems Review*, vol. 22, no. 4, pp. 36–38, Oct. 1988.

[5] Y. Liu, G. Deng, Y. Li, K. Wang, Z. Wang, X. Wang, T. Zhang, Y. Liu, H. Wang, Y. Zheng, L. Y. Zhang, and Y. Liu, "Prompt injection attack against llm-integrated applications," *arXiv preprint arXiv:2306.05499*, 2023.

[6] OWASP, "Top 10 for LLM applications," 2025. [Online]. Available: https://owasp.org/www-project-top-10-for-large-language-model-applications/

[7] Apache Arrow, "Arrow flight RPC," Apache Software Foundation, 2024, apache Arrow Documentation. [Online]. Available: https://arrow.apache.org/docs/format/Flight.html

[8] AICPA, "2017 trust services criteria (with revised points of focus–2022)," American Institute of Certified Public Accountants, 2022, trust Services Criteria. [Online]. Available: https://www.aicpa-cima.com/resources/download/2017-trust-services-criteria-with-revised-points-of-focus-2022

[9] Model Context Protocol Contributors, "Model context protocol specification," Model Context Protocol, 2025, specification, version 2025-06-18. [Online]. Available: https://modelcontextprotocol.io/specification/2025-06-18

[10] Eclipse Foundation, "Dataspace protocol 2025-1," Eclipse Foundation, 2025, stable Specification, Version 2025-1. [Online]. Available: https://eclipse-dataspace-protocol-base.github.io/DataspaceProtocol/

[11] M. Leo, F. Tan, T. Miao, and G. Anand, "From threat to trust: Assessing security risks of agentic ai systems," *International Journal of Information Security*, vol. 25, no. 1, p. 23, 2026.

[12] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, "Gorilla: Large language model connected with massive APIs," in *Advances in Neural Information Processing Systems*, vol. 37. Curran Associates, Inc., 2024, pp. 126 544–126 565.

[13] Oracle, "JDBC database access," Oracle, 2024, the Java Tutorials. [Online]. Available: https://docs.oracle.com/javase/tutorial/jdbc/

[14] PostgreSQL Global Development Group, "Row security policies," PostgreSQL, 2024, postgreSQL 16 Documentation. [Online]. Available: https://www.postgresql.org/docs/16/ddl-rowsecurity.html

[15] Google Cloud, "Introduction to column-level access control in bigquery," Google Cloud, 2024, bigQuery Documentation. [Online]. Available: https://cloud.google.com/bigquery/docs/column-level-security-intro

[16] B. Campbell, J. Bradley, and H. Tschofenig, "OAuth 2.0 token exchange," Internet Engineering Task Force, Tech. Rep. RFC 8693, Jan. 2020.

[17] N. Sakimura, J. Bradley, M. B. Jones, B. de Medeiros, and C. Mortimore, "OpenID connect core 1.0," OpenID Foundation, Feb. 2014, final Specification. [Online]. Available: https://openid.net/specs/openid-connect-core-1_0-final.html

[18] SPIFFE Project, "Secure production identity framework for everyone," Cloud Native Computing Foundation, 2026, sPIFFE Standard. [Online]. Available: https://spiffe.io/docs/latest/spiffe-specs/spiffe/

[19] L. Moreau and P. Missier, "PROV-DM: The PROV data model," World Wide Web Consortium, Apr. 2013, w3C Recommendation. [Online]. Available: https://www.w3.org/TR/2013/REC-prov-dm-20130430/

[20] International Data Spaces Association, "The IDS reference architecture model (IDS-RAM)," International Data Spaces Association, rAM 4 is the stable and official reference model; accessed September 5, 2026. [Online]. Available: https://internationaldataspaces.org/offers/reference-architecture/

[21] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero trust architecture," National Institute of Standards and Technology, Tech. Rep. NIST Special Publication 800-207, Aug. 2020.

[22] S. K. Singh, J. Roy, and M. So, "Zero-trust agentic federated learning for secure internet of things (iot) defense systems," in *2026 IEEE 2nd International Conference on Secure IoT, Assured and Trusted Computing (SATC)*, 2026, pp. 1–10.

[23] A. Parker, D. Spoonhower, J. Mace, B. Sigelman, and R. Isaacs, *Distributed Tracing in Practice: Instrumenting, Analyzing, and Debugging Microservices*. O'Reilly Media, Apr. 2020. [Online]. Available: https://www.oreilly.com/library/view/-/9781492056621/

[24] J. Roy and S. K. Singh, "Agentsoc: A multi-layer agentic ai framework for security operations automation," in *2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC)*, 2026, pp. 1–7.

[25] European Parliament and Council of the European Union, "Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act)," Jun. 2024, official Journal of the European Union, 12 July 2024. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2024/1689/oj

[26] Open Policy Agent, "Open policy agent documentation," Cloud Native Computing Foundation, 2024, official Documentation. [Online]. Available: https://www.openpolicyagent.org/docs/

[27] MITRE, "Navigate threats to ai systems through real-world insights," MITRE, 2024, adversarial Threat Landscape for Artificial-Intelligence Systems. [Online]. Available: https://atlas.mitre.org/

[28] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 785–794.

[29] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[30] M. Jones, J. Bradley, and N. Sakimura, "JSON web signature (JWS)," Internet Engineering Task Force, RFC 7515, May 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7515