PolicyStrata: Responsibility-Scoped Testing for Cross-Layer Policy Drift in LLM Data Agents

Zachary Roth
Raintree Technology
zacharyroth@pm.me

June 26, 2026

Abstract

LLM data agents translate natural-language requests into semantic plans, SQL queries, and analytical answers. Existing evaluations emphasize text-to-SQL correctness, malformed-output reduction, or model refusal behavior. Deployed agents also replicate authorization, semantic, lineage, version, and release obligations across model-visible manifests, grammars, validators, compilers, database controls, and output filters. These non-equivalent surfaces can drift after policy, schema, or semantic model updates, producing failures that component tests miss.

We present PolicyStrata, a regression-testing framework for cross-layer policy drift in governed data-agent stacks. PolicyStrata generates principals, requests, plans, mixed-version surface configurations, database states, and lowered queries, then checks whether each surface preserves its assigned authorization, semantic, lineage, containment, and release obligations. Its minimized witnesses identify the first violated transition. The artifact includes DataPolicyDriftBench, a deterministic benchmark with three analytics domains; seeded, generated, detector-frozen held-out, and clean-control suites; baseline comparators; ablations; JSONL traces; optional database smoke fixtures; and a brownfield integration check. On deterministic, non-equivalent mutant suites generated from the implemented operators and fixtures, PolicyStrata reports witnesses for all 1720 covered injected faults, with 0 false positives on 80 clean controls. This is v1 fault-model coverage, not recall on unknown production faults.

1 Introduction

LLM data agents connect natural-language interfaces to identity systems, semantic models, metric definitions, query compilers, database authorization, row-level security, result summarization, and audit trails. A generated query can be syntactically valid while still violating tenant scope, metric semantics, lineage requirements, or release policy.

The central claim of this paper is narrow:

> Cross-layer policy drift is an under-measured cause of authorization and semantic failures in LLM data agents.

This is not a claim that constrained decoding, dynamic grammars, or semantic intermediate representations provide security. They reduce malformed actions and simplify validation, but they do not guarantee authorization, confidentiality, semantic correctness, or safe release.

Consider a multi-tenant support analytics agent. A support analyst may see tenant-A ticket aggregates, but not tenant-B data, raw customer emails, or gross-revenue metrics. After a policy update, the validator moves to version 7, the model-visible manifest still exposes a version-6 metric alias, and the SQL compiler still uses a version-5 tenant key. A request for "last month's escalations by region" can be accepted, lowered through the stale alias, and compiled with the wrong tenant predicate. Database RLS may contain the bug, or the result may be released. The useful failure report is a small database-backed witness and the first failed transition.

The right property is not equality between layers. A manifest may intentionally expose less than a validator permits, a grammar may describe expressible plans rather than authorized database operations, and a service account may have privileges no user request should exercise. PolicyStrata instead checks responsibility-scoped contracts over two dimensions: runtime objects that flow through the agent and policy surfaces that expose, accept, reject, transform, execute, or release those objects.

Prior work tests whether models generate correct SQL, whether SQL queries are semantically equivalent, whether policies differ, or whether runtime monitors enforce a policy. PolicyStrata instead tests whether a governed data-agent deployment preserves authorization, semantic, lineage, version, containment, and release obligations across non-equivalent policy-bearing representations. Its output is not only pass/fail; it is a minimized database-backed witness and first-transition attribution for the violated obligation.

Scope. PolicyStrata is a regression-testing framework, not a replacement authorization engine. It assumes a canonical specification exists, treats that specification as an oracle for the tested deployment, and searches for disagreements between that oracle and independently maintained implementation surfaces. Its v1 target is single-request, read-only analytical workflows over governed data. Write actions, adaptive privacy, and history-aware release policies are deliberately outside the deterministic v1 guarantee.

Claims and non-claims.

Claim	Non-claim
Cross-layer policy drift can be tested with responsibility-scoped witnesses	POLICYSTRATA is not an authorization boundary
The artifact covers implemented deterministic mutant operators and fixtures	The 1720/1720 result is not real-world recall
Constrained generation can improve reliability and traceability	Grammar membership is not a security guarantee
Deterministic evidence requires no LLM API key	Model-mediated exploitability is not part of the deterministic result

Contributions. This paper makes four contributions:

- We formalize LLM data agents as sequences of non-equivalent policy-bearing representations and define surface-specific contracts for exposure soundness, declared completeness, authorization-preserving lowering, semantic preservation, database containment, and stateless release conformance.
- We introduce POLICYSTRATA, a policy-guided test generator over principals, latent intents, natural-language requests, semantic plans, version vectors, database states, lowerings, and release decisions.
- We define a witness and attribution procedure that minimizes failures while preserving the violated obligation, the distinguishing database state, and the first observed violated transition.
- We provide DATAPOLICYDRIFTBENCH, a deterministic runnable v1 artifact with three analytics domains, seeded, generated, detector-frozen held-out, and clean-control suites, reproducible traces, summary metrics, minimized witnesses, baseline comparators, ablations, optional database smoke fixtures, imported-trace scanning, and deployment-inventory support.

2 Background and Related Work

Text-to-SQL correctness is not enough. Benchmarks such as Spider and BIRD advanced evaluation of database-grounded question answering [9, 21], and distilled test-suite accuracy showed how generated databases can better approximate semantic accuracy than exact string or single-database execution matching [22]. Enterprise data-agent benchmarks such as DAB, BEAVER, and DAComp further emphasize that real data work involves heterogeneous systems, business-specific schemas, evolving pipelines, joins, temporal semantics, and open-ended analysis rather than isolated SQL generation [1, 8, 11]. DAB covers multi-database enterprise data questions, BEAVER spans many enterprise schemas and domains, and DAComp covers repository-level data engineering plus open-ended analysis. These benchmarks establish the difficulty of enterprise data agents, but correctness against a gold query or workflow rubric does not imply that an answer is authorized, safe to release, or aligned with enterprise metric definitions.

Constrained generation is useful but limited. PICARD, grammar-constrained decoding, structured-output libraries, and JSON-schema benchmarks show that constraining the decoder can reduce invalid outputs and enforce syntactic membership in a target language [3, 5, 16, 20]. In data agents, that is valuable operationally: fewer malformed actions, cleaner validators, easier audit logs, and lower repair cost. But a syntactically valid query can still access the wrong tenant, compute a forbidden metric, infer sensitive values through differencing, create misleading aggregates, or exceed query budgets.

Policy enforcement is active prior art. Recent systems for agent policy enforcement, including deterministic policy compilers and programmable privilege controls, move beyond prompt-only security [13, 17]. Role-conditioned refusal benchmarks also study whether models or verifier pipelines respect access-control policies over text-to-SQL tasks [7]. Work on data-flow control further argues that query correctness is distinct from safety [18]. These works motivate a different contribution here: not another enforcement layer, but a testing framework that detects drift and attributes the first layer transition where a policy obligation stops holding.

Differential policy and database testing are prior art. Margrave computes semantic differences between access-control policy versions [4], and access-control mutation testing defines fault models, mutation operators, mutant killing, and equivalent-mutant handling for policy test adequacy [12]. Cedar's development process used formal models, differential random testing, and property-based testing to compare a production authorization engine against an independently specified model [2]. Beacon detects DBMS access-control bugs by checking consistency between executable database operations and system-catalog visibility [14]. SQL equivalence and text-to-SQL evaluation systems generate distinguishing databases or prove bounded equivalence for query pairs [6, 19, 22]. POLICYSTRATA builds on these ideas but changes the unit of analysis: it studies conformance relationships among heterogeneous policy-bearing representations in one deployed data-agent pipeline.

Novelty boundary. The contribution is not a new access-control logic, SQL equivalence checker, or agent runtime monitor. POLICYSTRATA targets the boundary where deployed data agents fail: non-equivalent, versioned representations that must preserve authorization, semantic, lineage, and release obligations as a request moves from a model-visible surface to executable behavior.

Constraint drift and LLM-specific surface area. Recent work on constraint drift argues that safety-critical constraints can weaken as they move through memory, delegation, tool I/O, execution, audit, and optimization [10]. POLICYSTRATA is narrower: it makes drift in governed data agents observable through database witnesses, version vectors, and surface-specific contracts. Natural-language ambiguity, stochastic request-to-plan translation, model-visible manifests, repair

loops, raw result data entering an LLM context, and model or prompt updates determine which obligations are reached.

3 Problem Definition

Let a data-agent pipeline have runtime objects:

$$X_0 \to X_1 \to X_2 \to X_3 \to X_4$$

The representations may correspond to a model-visible request, a typed semantic plan, a validated executable plan, a SQL query and execution context, and a released analytical result. The surfaces that handle those objects are related but not interchangeable:

Request or latent intent
↓
Manifest or grammar: expose only authorized capabilities
↓
Semantic-plan validator: accept supported authorized plans and reject unauthorized plans
↓
Compiler or lowering: preserve tenant, purpose, policy version, lineage, and semantics
↓
Database policy and RLS: contain unauthorized executable operations
↓
Release policy: release only allowed result-lineage pairs

The surface-specific contracts used by POLICYSTRATA are summarized in Table 1. The formal versions appear in Appendix A.

Table 1: Policy-bearing surfaces and their responsibility-scoped obligations.

Surface	Responsibility
Manifest or grammar	Do not expose capabilities whose reachable operations are unauthorized; underexposure may be intentional.
Validator	Accept authorized objects inside the declared support envelope and reject unauthorized objects.
Compiler	Preserve principal, tenant, purpose, policy version, lineage, and business semantics during lowering.
Database policy	Contain unauthorized executable operations under the runtime database role.
Release policy	Release only result-lineage pairs allowed to leave the trusted boundary.

A grammar is a language recognizer. A validator is a decision procedure. A compiler is a partial lowering function. RLS is an execution-time reference monitor. A manifest is a projection shown to the model. Treating these as one kind of object obscures the bugs we want to find.

At version t, the canonical policy separates authorization, semantic, and release specifications: $P^t = (P^t_{\mathsf{auth}}, P^t_{\mathsf{sem}}, P^t_{\mathsf{rel}})$. The authorization specification defines who may perform operations over rows, columns, metrics, tenants, and purposes. The semantic specification defines the meanings of metrics, dimensions, aliases, grains, joins, filters, and time windows. The release specification defines whether a result and its lineage may leave a trusted boundary. These are intentionally separate: a correct RBAC decision cannot prove that "net revenue" used the right denominator, and a semantically correct aggregate may still be unsafe to disclose.

Exposure soundness. A model-visible manifest or grammar is exposure-sound if every operation reachable through an exposed capability is authorized for the principal and context. This does not require every authorized capability to be exposed.

Declared completeness. Over-restrictive behavior is a defect only inside the advertised support envelope. This avoids treating intentional underexposure, feature flags, stricter release suppression, or unsupported-but-authorized requests as bugs.

Authorization-preserving lowering. Lowering must also preserve security-relevant context: principal, tenant, purpose, jurisdiction, policy version, release boundary, and time semantics cannot be silently dropped. This property catches compilers that accept a valid plan but emit SQL that reads the wrong tenant, expands a metric into forbidden columns, or uses stale identity keys.

Semantic translation validation. POLICYSTRATA searches for admissible database states where the lowered query and the reference semantic plan produce different observations. The comparison must define bag versus set semantics, ordering, floating-point tolerance, NULL and three-valued logic, errors, timestamp and timezone behavior, and nondeterministic functions. Testing finitely many generated states finds distinguishing witnesses; it does not prove general SQL equivalence unless paired with a verifier such as bounded SQL-equivalence checking [6, 19]. Semantic equality is not sufficient for authorization: two queries can return the same visible result while one scans forbidden rows or columns. POLICYSTRATA therefore checks both denotational behavior and dependency or lineage conformance.

Release conformance. For v1, release is stateless and single-query: a result and its lineage may leave a boundary only if the release policy permits that pair for the principal and context. A release boundary may be the human user, the LLM context, an external model provider, a log, a cache, or another tool. Raw unauthorized rows entering an untrusted model context can be a disclosure even if the final user-facing answer is suppressed.

Witness classes. POLICYSTRATA reports five main witness classes. Over-permissive exposure means a surface exposed an unauthorized capability. Over-restrictive rejection means an authorized object inside the declared support envelope was rejected. A lowering violation means an accepted object was compiled or transformed into unauthorized behavior. Semantic drift means a lowered query disagrees with the reference semantic plan on a distinguishing database state. A release violation means a result-lineage pair crossed a boundary that the release policy should have blocked.

Drift witness. A drift witness records the principal, request, database state, version vector, runtime object, lowered object, lineage, observed result, first violated transition, and violated obligation. Witness minimization preserves the violated obligation and first observed violated transition while shrinking

principal attributes, request tokens, plan nodes, policy clauses, database rows and values, version-vector entries, and SQL structure.

Divergence, drift, and version skew. A static mismatch present from initial deployment is policy divergence. Update-induced drift exists when the pipeline satisfied expected contracts before an update but fails afterward. The common operational case is version skew: the canonical policy may be at version 7 while the manifest is version 7, the grammar is version 6, the compiler rules are version 5, and the database RLS policy is version 7. Version vectors are therefore first-class test inputs rather than trace metadata. POLICYSTRATA can detect divergence, update-induced drift, and stale-artifact skew when an independent reference specification is available. It cannot detect a common-mode error where every surface faithfully implements a bad canonical policy unless an external specification or human-reviewed oracle contradicts that policy.

4 System Design

POLICYSTRATA has five deterministic components.

1. **Canonical specifications**. Separate authorization, semantic, and release specifications for principals, tenants, roles, purposes, metrics, dimensions, columns, row scopes, semantic formulas, join grains, aliases, time windows, lineage, and query budgets.
2. **Independent reference oracles**. Separately implemented interpreters for authorization, business semantics, and release conformance. They label generated objects without sharing production decision code, policy-precedence helpers, tenant expansion, SQL compiler logic, or expected-label generation with the surfaces under test.
3. **Surface and runtime adapters**. Parsers and normalizers for model-visible capability schemas, grammars, semantic IR plans, validator decisions, compiler contracts, compiled SQL, database policies, raw results, and released answers.
4. **Policy-guided generator**. A generator for principals, latent intents, paraphrased requests, semantic plans, SQL programs, database states, implementation version vectors, and mutation operators. Database-state generation is first-class because semantic compiler bugs are often observable only on a distinguishing database instance.
5. **Witness minimizer and localizer**. A reducer that shrinks failing cases and labels the first observed violation: over-permissive exposure, over-restrictive rejection, context-dropping lowering, semantic drift, database containment, or release-policy failure.

Algorithm 1: Responsibility-scoped drift testing.

1. Sample or import $(p, r, D, \vec{v})$: principal, request, database state, and surface-version vector.
2. Derive a reference semantic plan, authorization label, lineage expectation, and release decision from independent policy interpreters.
3. Replay the implementation surfaces: manifest/grammar exposure, validator decision, compiler lowering, database containment, and release decision.
4. Check each transition against the surface responsibility in Table 1, stopping at the first violated obligation.
5. If a violation is found, minimize principal attributes, request tokens, plan nodes, database rows and values, version entries, and SQL structure while preserving the obligation and first-transition label.
6. Emit a JSONL trace plus a witness record; otherwise count the case as clean or intentionally unsupported.

5 Implementation and Reproducibility

POLICYSTRATA is implemented as a Python package with a command-line interface. The full deterministic evidence path runs with:

```
scripts/reproduce-final.sh
```

The implementation keeps the authorization, semantic, and release interpreters independent from the SQL compiler path. The trusted computing base is therefore explicit: reference interpreters, surface adapters, and hand-derived examples are tested artifacts, not hidden benchmark plumbing. Run metadata records mutation operators, suite provenance, evidence level, freeze status, manifest IDs, detector hashes, generator hashes, policy hashes, surface hashes, and suite/task hashes so generated, held-out, clean-control, and externally authored evidence can be reported separately.

A reproducible run requires no LLM API key and no host `psql`. The artifact writes `traces.jsonl`, summaries, minimized witnesses, baselines, ablations, and freeze manifests. Optional Docker services exercise PostgreSQL/RLS and ClickHouse smoke paths; those smoke tests are separate from the deterministic score.

The package also includes scanner and doctor modes as artifact-support and deployment-inventory paths, not as part of the benchmark score. They import application SQL/tool traces, semantic models, optional PostgreSQL fixtures, RLS checks, state assertions, release decisions, prompt/tool manifests, source maps, CI gates, and policy-document signals. The output is wiring inventory, coverage accounting, and remediation guidance. It is not legal review, not production recall evidence, and not an authorization guarantee.

6 DATAPOLICYDRIFTBENCH

DATAPOLICYDRIFTBENCH ships three synthetic but realistic analytics domains:

- `support_saas`: B2B SaaS embedded analytics over tenants, accounts, subscriptions, invoices, support tickets, ticket events, and support agents.

- `finance_saas`: household and advisor financial analytics over firms, advisors, households, accounts, transactions, holdings, and sensitive identifiers.
- `analytics_clickhouse`: product analytics over projects, sessions, events, aggregate-only roles, cohort-release thresholds, ClickHouse-style row policies, timezone windows, and materialized-view lineage.

Across the three domains, fixtures cover tenant and ownership scopes, sensitive-column restrictions, metric aliases, join-path and grain constraints, fiscal/calendar time semantics, raw-record versus aggregate-only access, query budgets, firm/advisor isolation, household-scoped access, small-cohort release suppression, distinct-user/session grain, timezone-bucket preservation, sampled aggregate disclosure, and materialized-view lineage. This reduces single-domain risk, but the suites remain built-in synthetic fixtures rather than independent field evidence.

A representative minimized witness contains two tenants, one support analyst principal scoped to tenant A, one tenant-A ticket, one tenant-B ticket with a distinguishing escalation flag, a semantic request for escalations by region, and a mixed-version surface vector. The validator accepts the request under the version-7 policy, but the compiler lowers through a stale tenant key and emits SQL whose lineage includes tenant B. If database RLS blocks the row, the witness is a contained lowering violation; if the result is released, the same witness becomes an external disclosure. The output is a minimized reproducer rather than only a wrong answer.

Principal
Support analyst scoped to tenant A.
Request
"Escalations by region."
Canonical policy
Tenant-A aggregates allowed; tenant B and raw emails denied.
Version vector
Validator v7; manifest v6 alias; compiler v5 tenant key.
Lowered lineage
SQL dependencies include tenant A and tenant B rows.
First violation
Compiler lowering failed to preserve tenant scope.
Outcome
RLS containment yields a contained lowering violation; release yields disclosure.

The seeded mutation families cover representative cross-layer faults rather than an exhaustive catalog:

Family	Representative examples
Authorization and identity drift	stale role membership; tenant-id/account-id migration; default-tenant fallback
Manifest and validator drift	retired metric remains visible; grammar admits validator-rejected dimensions; sensitive column omitted from deny list
Semantic compiler drift	tenant predicate dropped; gross/net metric skew; fiscal/calendar time mismatch; join fan-out
Database containment drift	stale RLS ownership field; bypassing role; application deny rule not propagated to database
Deployment/version skew	partial rollback; stale distributed cache; validation and execution under different policy versions
Budget and release drift	compiled query exceeds budget; raw unauthorized rows enter LLM context; stale cached answer released

PostgreSQL RLS deserves explicit adversarial configuration cases. Superusers and roles with `BYPASSRLS` bypass row security, table owners normally bypass row security, and `FORCE ROW LEVEL SECURITY` changes owner behavior [15]. A fixture can therefore appear protected while actually executing through a bypassing role. These cases are useful because they distinguish application-layer drift from database-layer containment failures.

7 Deterministic Artifact Results

The runnable Python artifact writes deterministic traces, minimized witnesses, baseline reports, and artifact-usability reports without depending on Inspect, BenchFlow, or an LLM API key.

The v1 deterministic snapshot contains seeded, generated, detector-frozen held-out, and clean-control suites:

Suite	N	Killed	Surv.	Ctrl/FP	Frozen
`support` seeded	50	50	0	0/0	no
`support` generated	500	500	0	0/0	yes
`support` heldout-v1	500	500	0	0/0	yes
`finance` seeded	20	20	0	0/0	no
`finance` heldout-v1	250	250	0	0/0	yes
`analytics` seeded	100	100	0	0/0	no
`analytics` generated	300	300	0	0/0	yes
`clean-controls`	80	–	–	80/0	yes
Non-clean total	**1720**	**1720**	**0**	–	mixed

Baseline comparators are evaluated over the same non-clean traces:

Comparator	Failures caught
Grammar only	121/1720
Semantic validator only	573/1720
SQL AST policy checker	1043/1720
DB policy only	326/1720
Release filter only	364/1720
Lineage only	239/1720
Random data generation	1246/1720
Naive surface equality	573/1720
Defense-in-depth stack	1561/1720

Baselines are intentionally simple point controls, not optimized competing systems. They represent common deployment checks: grammar membership, validator decisions, SQL snapshot inspection, DB containment, release filtering, lineage checks, final-answer checks, random database generation, and unions of these checks. The defense-in-depth baseline approximates a plausible layered production stack by unioning validator-only, SQL-snapshot, DB/RLS-only, and final-answer checks. Its remaining 159 misses are cases where stacked point controls lack POLICYSTRATA's responsibility-scoped contracts or witness localization. These are controlled deterministic-mutant results, not a claim of 100% recall on unknown faults.

Baseline information access. All baselines run over the same non-clean traces and receive only the inputs their label implies. No baseline receives POLICYSTRATA's first-transition labels or minimized witnesses.

Grammar only
Manifest/grammar membership only; no validator, SQL, database state, lineage, or release result.

Validator only
Semantic object and validator decision; no compiled SQL, DB containment, lineage, or release channel.

SQL AST policy checker
Compiled SQL and row/column/metric checks; no manifest intent, database state, or release decision.

DB/RLS only
Executable database role and containment result; no manifest, validator, compiler context, or release lineage.

Release only
Released result and lineage only; cannot inspect earlier exposure, validation, lowering, or DB containment.

Random data generation
Semantic comparison over generated states without policy-guided transition obligations.

Naive surface equality
Direct representation comparison; cannot model intentional underexposure or stricter downstream controls.

Defense-in-depth
Union of point checks above; still lacks responsibility-scoped transition contracts and witness attribution.

Ablations over the same non-clean traces show which obligations matter most:

Ablation	Still caught	Missed
Without database containment	1394/1720	326
Without independent oracle	1357/1720	363
Without lineage	1657/1720	63
Without release policy	1682/1720	38
Without transition obligations	363/1720	1357

Policy-version and minimization ablations affect provenance and witness usability rather than the deterministic kill count; they should not be read as detection-rate ablations.

Runtime and artifact size. In one artifact-usability run on a local Apple laptop with an already prepared `uv` environment, `scripts/reproduce-final.sh` completed in 4.27 seconds wall-clock time. The run generated 1800 traces, including 1720 non-clean cases and 80 clean controls; 1720 minimized JSON witnesses; 5 freeze manifests; baseline, ablation, export-adapter, and artifact reports; 1760 files total; and a 17 MB run directory. Emitted witness files were 3080–3602 bytes, with a median of 3302 bytes. The current artifact reports emitted witness size rather than pre-reduction size.

Brownfield wiring case study. A separate household-finance application integration exercises real repository surfaces with synthetic fixture data. It does not contain production customer rows, credentials, signed agreements, or private support tickets. The integration maps policy, terms-of-service, and internal-control documents, a prompt/tool manifest, source map, semantic model, SQL/tool traces, Docker PostgreSQL RLS checks, release decisions, and a CI workflow into POLICYSTRATA.

The purpose of this case is integration coverage, not fault-discovery measurement.

Tenant boundary
Household-scoped `household_id` predicate.

Prompt manifest
8 metrics, 13 dimensions, 18 tool definitions.

Sensitive dimensions
Account name, account mask, institution, merchant, transaction note.

Database fixture
10 tables, 10 RLS policies, 10 tenant columns.

Trace evidence
6 SQL/tool records, all with semantic IR; 2 denial cases.

Database checks
2 RLS checks plus 2 state assertions.

Result
CI gate pass; 0 findings; `ci-gate-ready`.

Interpretation. This is wiring evidence: POLICYSTRATA can ingest a brownfield data-agent repository and account for policy documents, exposed tools, semantic models, source mappings, traces, RLS, release decisions, and CI gates. It is not part of the 1720-case deterministic benchmark, not production customer-data evidence, and not independent production-recall evidence.

8 Evaluation Methodology

The evaluation treats mutant killing as coverage over named deterministic operators and fixtures. A mutant is killed when POLICYSTRATA emits a non-clean witness that preserves the violated obligation and records the first observed failing surface or transition. Reports separate killed, survived, equivalent, invalid, clean-control, and false-positive counts. Public generated suites are reported separately from detector-frozen, externally authored, blinded, or reconstructed real-fault evidence.

Baseline comparison. All deterministic baselines are evaluated over the same trace set as POLICYSTRATA. The artifact includes point baselines for grammar, validation, SQL policy checks, DB/RLS containment, release filters, lineage, final answers, SQL snapshots, random data generation, naive surface equality, and defense-in-depth unions. Future baselines should add component unit tests, distilled test-suite-style database generation, bounded SQL-equivalence checking where supported, and LLM red-team generation under matched identity context, retry budget, refusal channel, and model settings.

Freeze protocol. The freeze command hashes detector source, mutation operators, generator source, policy and surface YAML, suite inputs, materialized tasks, package version, and git commit when available. The run command verifies that manifest before producing traces and copies it into the run directory. This protocol does not make generated cases externally authored, but it prevents post-hoc detector or suite tuning from being silently mixed into held-out evidence.

Localization and minimization. First-transition attribution is useful for debugging, but it is not root-cause localization without counterfactual repair experiments. Witness minimization reports reductions in principal attributes, request tokens, plan nodes, policy clauses, database rows and values, version-vector entries, and SQL structure while preserving the same obligation. Delta-debugging-style reducers generally produce 1-minimal witnesses rather than globally minimum witnesses.

Intentional asymmetry and scale. False-positive evaluation must include legitimate differences among surfaces: under-exposed manifests, feature flags, stricter RLS, service-account ambient authority, staged rollouts, release suppression, and authorized but unsupported requests. Scalability should be reported against principals, roles, rules, schema objects, metrics, aliases, version combinations, plan depth, database-state size, and policy surfaces. Version-vector exploration should use deployment-aware generation or covering arrays rather than enumerating every Cartesian combination.

9 Artifact and Data Availability

The submission kit contains a reviewer guide, expected outputs, freeze manifests, Docker Compose services for optional database smoke tests, public and anonymous paper builds, scanner examples, doctor/audit guidance, and generated result artifacts. The benchmark data are synthetic, seeded, and included with the repository so deterministic traces can be regenerated rather than downloaded from an external service.

10 Evidence Ladder and Future Work

The reported result is controlled deterministic artifact-suite evidence, including detector-frozen generated held-out suites but not externally authored suites. The next evidence levels are blind mutations authored independently of the detector, reconstructed real faults from issue trackers or customer incidents, and model-in-the-loop reachability studies over fixed latent drift cases. Architecture-arm comparisons among raw SQL, constrained SQL, typed semantic IR, deterministic compilers, database RLS, and independent runtime enforcement should be reported only after those arms are run under controlled model and retry settings.

Sequential release, aggregation privacy, adaptive differencing, history-aware release policies, counterfactual repair, and minimal repair-set accuracy require additional semantics and should not be read into the deterministic artifact claim.

11 Threats to Validity

- Seeded and generated mutants may overstate benchmark control and understate real-world messiness. Detector-frozen suites reduce tuning risk but are still built-in deterministic fixtures rather than externally authored or blinded benchmarks.
- Independent reference interpreters reduce common-mode implementation errors, but they cannot detect a uniformly bad canonical policy without an external specification.
- Surface adapters can hide or invent drift, so they are part of the trusted computing base and require unit, metamorphic, and hand-example tests.
- Synthetic schemas may miss organizational policy nuance, while customer-specific schemas may limit reproducibility.
- Generated database states can miss semantic differences outside the searched state space, and equal visible results do not prove authorized lineage.
- PostgreSQL/RLS and ClickHouse fixtures are smoke evidence, not part of the 1720-case deterministic benchmark. Model-in-the-loop exploitability depends on model choice, prompt design, and retry budget and should be reported separately.

12 Conclusion

POLICYSTRATA frames governed LLM data-agent safety as a cross-layer conformance problem: whether authorization, semantic, and release obligations survive translation across representations that are independently reasonable but not equivalent. It makes that problem testable with minimized database-backed witnesses, first-transition attribution, and containment measurements across the layers where policy drift occurs.

A Contract Formalization

The formalization fixes terminology for the implemented contracts rather than providing a soundness theorem for arbitrary data-agent deployments.

Let p be a principal and c be runtime context containing request metadata, session state, and deployment versions. Let X_i be the object language at surface i, and let $\gamma_i(x)$ interpret object $x \in X_i$ as the concrete operations, data dependencies, or observations it may induce. Let $A_P(p, a, c)$ mean operation or dependency a is authorized by P^t_{auth}. Let Expose_i, Accept_i, and Release describe actual implementation behavior. Let $E_i(p, x, c)$ be the declared support envelope for surface i, and let $L_i : X_i \rightharpoonup X_{i+1}$ be a partial lowering.

Exposure soundness.

$$\mathsf{Expose}_i(p, x, c) \Rightarrow \forall a \in \gamma_i(x), A_P(p, a, c)$$

Declared completeness.

$$E_i(p, x, c) \wedge \forall a \in \gamma_i(x), A_P(p, a, c) \Rightarrow \mathsf{Accept}_i(p, x, c)$$

Authorization-preserving lowering.

$$\mathsf{Accept}_i(p, x, c) \wedge L_i(x) = y \Rightarrow \forall a \in \gamma_{i+1}(y), A_P(p, a, c)$$

Semantic translation validation.

$$\exists D : \mathsf{Obs}(\mathsf{Exec}(L_i(x), D)) \not\approx \mathsf{SpecEval}(x, D)$$

Release conformance.

$$\mathsf{Release}(p, y, \ell, c) \Rightarrow R_P(p, y, \ell, c)$$

References

[1] Peter Baile Chen, Devin Yang, Weiyue Li, Fabian Wenz, Yi Zhang, Nesime Tatbul, Michael Cafarella, Çağatay Demiralp, and Michael Stonebraker. BEAVER: An enterprise benchmark for text-to-SQL. *arXiv preprint arXiv:2409.02038*, 2024. arXiv v3 reports 9128 question-SQL pairs over 812 tables across 19 domains.

[2] Craig Disselkoen, Aaron Eline, Shaobo He, Kyle Headley, Michael Hicks, Kesha Hietala, John Kastner, Anwar Mamat, Matt McCutchen, Neha Rungta, Bhakti Shah, Emina Torlak, and Andrew Wells. How we built cedar: A verification-guided approach. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, 2024.

[3] Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. Xgrammar: Flexible and efficient structured generation engine for large language models. *arXiv preprint arXiv:2411.15100*, 2024. MLSys 2025.

[4] Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th International Conference on Software Engineering*, pages 196–205, 2005.

[5] Saibo Geng, Hudson Cooper, Michał Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert West, Eric Horvitz, and Harsha Nori. Jsonschemabench: A rigorous benchmark of structured outputs for language models. *arXiv preprint arXiv:2501.10868*, 2025.

[6] Yang He, Pinhan Zhao, Xinyu Wang, and Yuepeng Wang. Verieql: Bounded equivalence verification for complex sql queries with integrity constraints. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA2), 2024. Also available as arXiv:2403.03193.

[7] Đorđe Klisura, Joseph Khoury, Ashish Kundu, Ram Krishnan, and Anthony Rios. Role-conditioned refusals: Evaluating access control reasoning in large language models. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 6018–6034, Rabat, Morocco, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-eacl.316. URL https://aclanthology.org/2026.findings-eacl.316/. Also available as arXiv:2510.07642.

[8] Fangyu Lei, Jinxiang Meng, Yiming Huang, Junjie Zhao, Yitong Zhang, Jianwen Luo, Xin Zou, Ruiyi Yang, Wenbo Shi, Yan Gao, Shizhu He, Zuo Wang, Qian Liu, Yang Wang, Ke Wang, Jun Zhao, and Kang Liu. DAComp: Benchmarking data agents across the full data intelligence lifecycle. *arXiv preprint arXiv:2512.04324*, 2025.

[9] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Min Zhou, Fei Zhang, Guoliang Li, Kevin Chen-Chuan Chang, Xuanjing Huang, Reynold Cheng, and Yongbin Li. Bird: A big bench for large-scale database grounded text-to-sql evaluation. *arXiv preprint arXiv:2305.03111*, 2023.

[10] Tianxiao Li, Yixing Ma, Haiquan Wen, Zhenglin Huang, Qianyu Zhou, Zeyu Fu, and Guangliang Cheng. Safe multi-agent behavior must be maintained, not merely asserted: Constraint drift in LLM-based multi-agent systems. *arXiv preprint arXiv:2605.10481*, 2026.

[11] Ruiying Ma, Shreya Shankar, Ruiqi Chen, Yiming Lin, Sepanta Zeighami, Rajoshi Ghosh, Abhinav Gupta, Anushrut Gupta, Tanmai Gopal, and Aditya G. Parameswaran. Can AI agents answer your data questions? a benchmark for data agents. *arXiv preprint arXiv:2603.20576*, 2026.

[12] Evan Martin and Tao Xie. A fault model and mutation testing of access control policies. In *Proceedings of the 16th International Conference on World Wide Web*, pages 667–676, 2007.

[13] Nils Palumbo, Sarthak Choudhary, Jihye Choi, Guy Amir, Prasad Chalasani, and Somesh Jha. Formal policy enforcement for real-world agentic systems. *arXiv preprint arXiv:2602.16708*, 2026. FORGE; arXiv v3 describes a Datalog reference monitor, observability service, and aspect weaver.

[14] Zongrui Peng, Jingzhou Fu, Zhiyong Wu, Jie Liang, Xiangdong Huang, Dalong Shi, and Yu Jiang. Beacon: Detecting broken access control vulnerabilities in dbmss via system catalog consistency validation. *Proceedings of the ACM on Programming Languages*, 10 (OOPSLA1):905–933, 2026. doi: 10.1145/3798232.

[15] PostgreSQL Global Development Group. Postgresql documentation: Row security policies. https://www.postgresql.org/docs/current/ddl-rowsecurity.html, 2026. Accessed 2026-06-24.

[16] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093*, 2021.

[17] Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing ai agents with privilege control. *arXiv preprint arXiv:2504.11703*, 2025.

[18] Charlie Summers and Eugene Wu. Data flow control: Data safety policies for ai agents. *arXiv preprint arXiv:2606.05679*, 2026.

[19] Andrew Tremante, Yang He, Rocky Klopfenstein, Yuepeng Wang, Nina Narodytska, and Haoze Wu. Spotit+: Verification-based text-to-SQL evaluation with database constraints. *arXiv preprint arXiv:2603.04334*, 2026.

[20] Brandon T. Willard and Rémi Louf. Efficient Guided Generation for Large Language Models. arXiv preprint arXiv:2307.09702, 2023.

[21] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

[22] Ruiqi Zhong, Tao Yu, and Dan Klein. Semantic evaluation for text-to-SQL with distilled test suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 396–411, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.29. URL https://aclanthology.org/2020.emnlp-main.29/.