Positive Data Control: A Secure Architecture for LLM-Mediated Data Governance

Shames AL Mandalawi
University of Arkansas at Little Rock
Muzakirruddin Ahmed Mohammed
University of Arkansas at Little Rock
Mert Can Cakmak
mccakmak@ualr.edu
University of Arkansas at Little Rock
John R Talburt
University of Arkansas at Little Rock

Research Article

Keywords: Data governance, Large Language Models, Access control, Zero-trust architecture, Policy enforcement, Text-to-SQL security, Prompt injection, Defense-in-depth, Secure system architecture, Positive Data Control

Posted Date: May 7th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9317019/v1

Additional Declarations: No competing interests reported.

Positive Data Control: A Secure Architecture for LLM-Mediated Data Governance

Shames AL Mandalawi · Muzakirruddin Ahmed Mohammed · Mert Can Cakmak · John R. Talburt

Received: date / Accepted: date

Abstract Modern data governance relies on formal access control and documented policies, yet enforcement remains challenging in dynamic environments where users issue natural language data requests. Large Language Models (LLMs) offer semantic interpretation capabilities, but granting them direct access to sensitive data introduces risks such as prompt injection, malicious query generation, and unintended exposure.This paper presents a zero-knowledge governance architecture, where the LLM never sees actual data values, in which requests are interpreted using metadata, policies, and permissions, while deterministic validation and sandboxed execution enforce authorization. The LLM is treated as an untrusted component and does not constitute the enforcement boundary. We formalize the threat model, implement a prototype, and evaluate it across a comprehensive set of governance and adversarial scenarios. Results show correct policy decisions and successful prevention of unauthorized or injection-style access within the evaluated suite, without data-value exposure to the LLM. The findings demonstrate that architectural separation between semantic interpretation and data execution enables secure automated governance consistent with established information security principles.

Keywords Data governance; Large Language Models; Access control; Zero-trust architecture; Policy enforcement; Text-to-SQL security; Prompt injection; Defense-in-depth; Secure system architecture; Positive Data Control

Shames AL Mandalawi
Center for Entity Resolution and Information Quality (ERIQ)
University of Arkansas - Little Rock
E-mail: salmandalaw@ualr.edu

1 Introduction

Modern organizations depend on data for analytics, operations, and decision-making. At the same time, data has become a regulatory and security liability. Regulations such as the General Data Protection Regulation (GDPR) [10], sector-specific compliance mandates, and escalating cyber threats have elevated data governance from an administrative concern to a core security function. However, the scale and complexity of enterprise data now exceed what traditional, human-centered governance processes can reliably manage.

Conventional governance models rely on manual data stewardship, static access control configurations, and retrospective auditing. Formal mechanisms such as role-based access control (RBAC) [11] and attribute-based access control (ABAC) [12] provide structured permission management, but they require explicit configuration and continuous maintenance. As data systems evolve, policies, schemas, and user roles change frequently. Maintaining alignment between written policies and actual enforcement becomes increasingly difficult, creating a gap between governance intent and operational reality.

Positive Data Control (PDC) has been proposed as an architectural approach to address this limitation [26, 27]. Rather than allowing users to interact directly with data systems, PDC introduces a mediation layer that evaluates requests before execution. The concept is similar to fly-by-wire systems in aviation, where pilot inputs are interpreted and validated by a control computer before being translated into physical actions. In a data governance context, the control layer interprets user intent, checks policy compliance, and logs decisions prior to allowing access.

Figure 1 illustrates this distinction. In traditional architectures, users interact directly with data systems

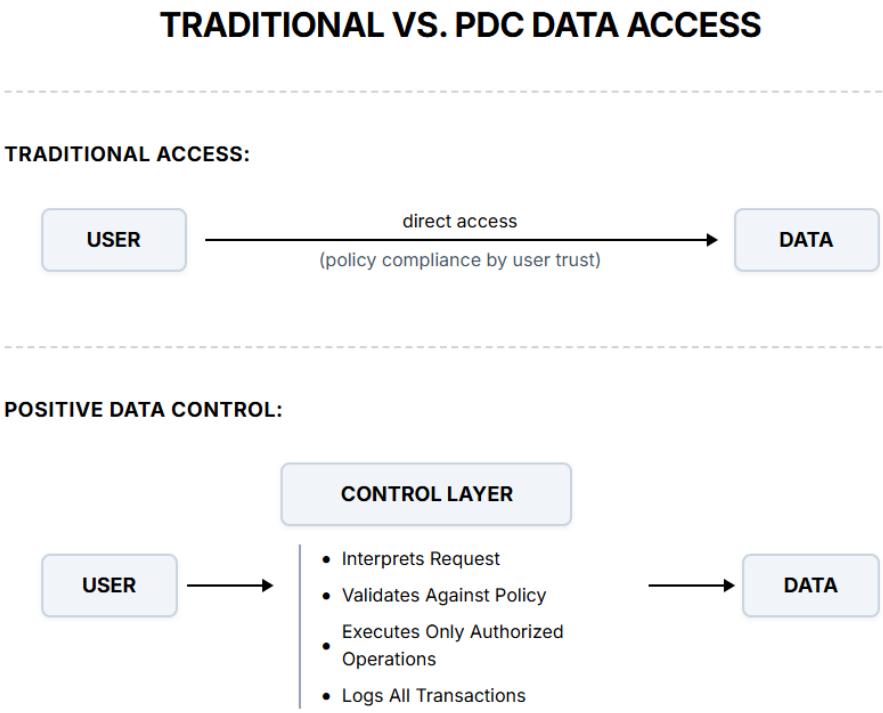

Fig. 1 Traditional versus Positive Data Control (PDC) data access. In traditional architectures, users interact directly with data systems and policy checks are often applied post hoc. In the PDC model, a control layer mediates all requests, interprets user request, validates policy compliance, and logs decisions before any operation reaches the data layer.

and policy violations are often detected after the fact. In contrast, PDC shifts enforcement to the point of access, preventing violations before they occur.

Although the PDC model is conceptually clear, implementing it in practice presents a challenge. Users express data needs in natural language, while enforcement mechanisms operate over structured schemas and formal permission rules. Static rule engines struggle to interpret ambiguous or novel requests. Recent advances in Large Language Models (LLMs) offer a potential solution by enabling semantic interpretation of natural-language inputs and translation into structured queries.

However, introducing LLMs into governance systems raises significant security concerns. Text-to-SQL systems have been shown to be vulnerable to prompt injection and malicious query generation [17,7,8]. Industry guidance has highlighted LLM-specific risks, including policy bypass and unintended data exposure [16]. Granting an LLM direct access to sensitive data therefore expands the attack surface and conflicts with established security principles such as least privilege and separation of duties [15].

This paper addresses this tension by proposing a zero-knowledge governance architecture in which the LLM interprets requests without accessing actual data values. Here, "zero-knowledge" is used in an architectural sense: the model never receives database records or query results. Instead, it operates only on governance artifacts such as schema metadata, sensitivity labels, access policies, and user permissions. Deterministic validation and sandboxed execution enforce authorization within the data domain, ensuring that the LLM does not constitute the enforcement boundary.

The central premise of this work is that policy enforcement does not require exposure to data values. Structural information and clearly defined policies are sufficient to generate compliant queries, while execution control remains in trusted, deterministic components. By separating semantic interpretation from data access, the architecture limits the impact of model misbehavior, prompt injection, or API compromise.

This paper investigates the following research questions:

- **RQ1:** Can an LLM enforce governance policies using only metadata and policy descriptions?
- **RQ2:** How accurately can natural-language requests be translated into policy-compliant queries across different roles and constraints?
- **RQ3:** What security properties result from strict architectural separation between the LLM and the data layer?
- **RQ4:** Does this design provide a practical realization of Positive Data Control in enterprise settings?

To address these questions, we design and implement a prototype that combines LLM-based interpretation with deterministic SQL validation, sandboxed execution, purpose-aware access control, and audit logging. The system is evaluated using representative governance and adversarial scenarios across multiple organizational roles.

The primary contributions of this paper are:

1. A zero-knowledge governance architecture that separates semantic interpretation from data-value exposure.
2. A defense-in-depth enforcement pipeline combining LLM mediation with deterministic validation and sandboxed execution.
3. An empirical evaluation demonstrating policy compliant behavior and adversarial resilience.
4. An operational realization of Positive Data Control grounded in established information security principles.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 formalizes the security model and trust boundaries. Section 4 presents the system architecture. Section 5 details the prototype implementation. Section 6 reports evaluation results. Section 7 discusses implications and limitations. Section 8 concludes.

2 Related Work

2.1 Data Governance Frameworks

Data governance has evolved from IT-centric data management into structured organizational frameworks integrating policy, accountability, and compliance. Modern governance programs include metadata catalogs, business glossaries, stewardship roles, and formal access control mechanisms [28]. Regulatory mandates such as SOX, GDPR, and HIPAA have further reinforced the need for auditable and systematic governance processes.

International standards such as ISO 8000 formalize requirements for data quality and master data exchange [32]. Research has demonstrated the application of ISO-aligned quality models to enterprise repositories, enabling measurable and repeatable assessments [33]. While these frameworks strengthen documentation and accountability, enforcement often remains dependent on human oversight or static rule-based configurations.

To address these limitations, Talburt proposed Positive Data Control (PDC) as a mediation architecture positioned between users and data systems [26,27]. PDC emphasizes proactive policy enforcement at the point of access rather than retrospective auditing. However, practical implementations that combine automation with strong security guarantees remain limited.

2.2 Access Control and Policy Enforcement

Access control mechanisms operationalize governance rules. Role-Based Access Control (RBAC) provides organizational alignment through role abstraction [11], while Attribute-Based Access Control (ABAC) extends authorization decisions to contextual attributes [12]. Formal policy languages such as XACML offer standardized representations for machine-evaluable policies [29].

More adaptive models incorporate risk awareness into authorization decisions. Atlam and Yang integrate RBAC, ABAC, and risk-based controls to improve adaptability in healthcare environments [39]. Zero-trust architectures further reinforce continuous verification and the assumption of compromise [14].

Despite these advances, traditional access control systems require explicit policy encoding and continuous updates as environments evolve. They do not natively support interpretation of natural-language requests, nor do they address the semantic gap between user intent and structured enforcement logic.

2.3 LLMs and Text-to-SQL Systems

Large Language Models have enabled natural-language interfaces to structured databases. Text-to-SQL research, supported by benchmarks such as Spider [34], has improved semantic parsing across domains. Techniques such as constrained decoding increase syntactic validity of generated SQL [35].

However, generation accuracy does not imply security. Peng et al. demonstrate that production text-to-SQL systems can be exploited to generate malicious queries and are vulnerable to backdoor attacks [17]. Chafik et al. propose secure agent-based designs incorporating malicious prompt detection [18], reducing but not eliminating attack risk.

Recent work has extended LLM applications toward policy-aware mediation. The work [36] introduces a policy-constrained generative controller that interprets natural-language requests against metadata and written policies using staged reasoning and deny-by-default gates. Their evaluation shows improved auditability and decision precision. Similarly, the study [40] proposes a coordinated multi-agent governance architecture in which specialized LLM agents perform cataloging, policy interpretation, and access mediation.

These approaches demonstrate that LLMs can support governance automation. However, they do not formalize a strict architectural separation preventing the language model from accessing sensitive data values during enforcement. In most implementations, the LLM remains logically positioned close to operational data contexts.

Our work differs by treating the LLM as an untrusted interpreter constrained to metadata visibility, with deterministic validation and sandboxed execution forming the enforcement boundary.

2.4 Privacy-Preserving AI

Privacy-preserving machine learning techniques seek to limit data exposure during model training and inference. Differential privacy provides formal guarantees on information leakage from outputs [31]. Federated learning enables distributed model training without centralizing raw data [37]. Surveys highlight trade-offs between privacy guarantees, performance, and complexity [38].

These techniques primarily address training-time privacy and aggregate output protection. Less attention has been given to inference-time exposure in governance systems, where models access operational data during decision-making. The GDPR principle of data minimization underscores the need to restrict unnecessary data access during processing.

Our approach complements cryptographic privacy methods by enforcing architectural data minimization: the LLM operates only on metadata and policy artifacts, never on raw data values.

2.5 Research Gap

The literature provides mature but disconnected foundations. Governance frameworks define roles, policies, and quality standards [28,32], yet enforcement remains partly manual. Formal access control models offer deterministic authorization [11,12], but lack natural language flexibility. Text-to-SQL systems enable semantic interaction [34,35], yet prioritize generation accuracy over policy compliance and exhibit adversarial vulnerabilities [17,18]. Privacy-preserving AI research protects training data and aggregate outputs [31,37], but does not address governance-time architectural exposure.

What remains missing is an integrated architecture that simultaneously:

- Interprets natural-language data requests,
- Applies deterministic authorization independent of model reasoning,
- Performs policy reasoning over structured metadata rather than raw data values, and
- Enforces a strict boundary preventing the language model from accessing confidential records.

Existing systems either grant the LLM operational proximity to data or rely exclusively on static access control without semantic mediation. The absence of a design that combines LLM-assisted interpretation with deterministic, zero-trust enforcement creates a structural vulnerability: flexibility is gained at the expense of security, or security is preserved at the expense of usability.

This work addresses that architectural gap by proposing and empirically validating a zero-knowledge governance pattern in which the LLM acts as an untrusted interpreter constrained by metadata visibility and deterministic enforcement layers. The design aligns with established security principles including separation of privilege and defense in depth [15].

3 Security Model: Threats, Trust Boundaries, and Design Goals

This section clarifies the security foundations of the proposed Positive Data Control (PDC) architecture. Before describing the system in detail, it is necessary to define what security problem the architecture addresses, what types of attackers are considered, and which components are assumed to be trusted. Security claims are meaningful only when evaluated under an explicit threat model. For this reason, we specify the adversarial capabilities, trust boundaries, and security objectives that guide the design of our approach.

The need for a clear security model is particularly important in systems that incorporate Large Language Models (LLMs). While LLMs enable flexible natural-language interaction and dynamic policy interpretation, they also introduce new risks such as prompt injection, unintended query generation, and probabilistic errors in reasoning. Established security engineering principles emphasize that enforcement mechanisms must not rely solely on components that may behave unpredictably or be manipulated by adversarial input [1,2]. Accordingly, our design treats the LLM as a reasoning assistant rather than a trusted enforcement authority. Critical security guarantees are instead anchored in architectural separation, deterministic validation, and constrained execution.

In the following subsections, we first describe the system and trust assumptions, then formalize the adversary model, and finally state the security objectives that the architecture is designed to satisfy.

3.1 System and Trust Assumptions

The proposed architecture separates natural-language policy interpretation from direct data access. The LLM operates exclusively on metadata (schemas, sensitivity annotations, and policy descriptions) together with authenticated user context and declared purpose. The database engine, by contrast, contains the actual data values and executes validated queries within a sandboxed environment.

This separation is conceptually aligned with classical information-flow security models [3,4], in which sensitive data must not influence lower-security domains without explicit authorization. In our design, data values constitute the high-sensitivity domain, while the LLM control layer belongs to a strictly constrained interpretation domain. The architectural requirement is that database values and query result sets are never included in the LLM input context. The LLM therefore cannot memorize, process, or exfiltrate information it never receives.

We assume that authentication correctly binds users to roles and permission sets, and that the validation and execution layers enforce their programmed checks correctly. We do not assume that the LLM is trustworthy; rather, we treat it as a potentially unreliable or manipulable component whose outputs must be independently verified.

3.2 Adversary Model

We adopt a conservative adversarial perspective similar to the Dolev–Yao model in distributed security analysis [5], in which the adversary controls all user-provided inputs and attempts to subvert enforcement mechanisms.

First, we consider the authenticated malicious insider. Insider threats are widely recognized as a primary source of enterprise data breaches [6]. An authorized user may attempt to access restricted tables, retrieve sensitive columns such as salaries or social security numbers, exceed row limits to enable bulk extraction, or misuse declared purposes to circumvent policy boundaries. The system must therefore enforce least-privilege access independently of user intent.

Second, we consider prompt injection attacks specific to LLM-based systems. Recent research demonstrates that LLMs can be manipulated by adversarial instructions embedded within user input [7,8], including instructions that attempt to override system prompts or governance constraints. Industry guidance such as the OWASP Top 10 for LLM Applications [16] identifies prompt injection as a critical emerging risk. In our setting, an adversary may attempt to coerce the model into generating policy-violating SQL or ignoring access restrictions.

Third, we consider vulnerabilities documented in text-to-SQL systems. Empirical analyses have shown that such systems can be induced to generate malicious or unintended queries, including injection constructs and unauthorized joins [17,18]. Because the LLM in our architecture performs natural language to SQL translation, it inherits these risks unless properly constrained.

Fourth, we consider the possibility that the LLM service itself may be compromised or behave unpredictably. Foundation models are probabilistic systems and may produce incorrect outputs even absent malicious prompting [9]. Therefore, the LLM cannot be treated as a trusted enforcement authority.

Finally, we consider denial-of-service style attacks in which an adversary attempts to generate excessively expensive queries or unbounded result sets to exhaust system resources.

3.3 Scope and Limitations

The architecture is designed to prevent unauthorized disclosure of data values through the LLM-mediated governance interface. We do not claim protection against direct compromise of the underlying database infrastructure, credential theft outside the PDC interface, or sophisticated side-channel attacks. Furthermore, while the system enforces access constraints, it does not eliminate the possibility of statistical inference from legitimately authorized aggregate outputs; mitigating such inference risks would require differential privacy or query auditing mechanisms beyond the scope of this work.

3.4 Security Objectives

Under the above adversary model, the system is designed to satisfy the following objectives.

First, the LLM must not receive or process sensitive data values. This objective enforces architectural data minimization and aligns with regulatory principles such as those embodied in GDPR [10]. Because the model does not observe data values, risks of memorization or model-based exfiltration are structurally reduced.

Second, policy enforcement must be non-by-passable. Even if the LLM generates an incorrect or adversarially influenced query, deterministic validation must prevent execution of unauthorized operations. This layered enforcement approach follows defense-in-depth principles [2] and zero-trust architectural guidance [14], which recommend assuming component compromise and isolating critical assets.

Third, the system must enforce least-privilege access in accordance with established RBAC and ABAC principles [11,12]. Access decisions incorporate role, department, declared purpose, column sensitivity, and aggregation constraints. The LLM proposes queries, but final authorization is determined by independent validation against these constraints.

Fourth, the architecture must resist injection and dangerous operation attempts. Prior classifications of SQL injection attacks [13] motivate explicit detection of prohibited constructs such as DDL and DML statements, UNION-based exfiltration, and comment-based truncation.

Fifth, output handling must reduce exposure even for authorized users. Post-execution masking provides an additional layer of protection to minimize the impact of potential overexposure.

Finally, all decisions must be auditable. Comprehensive logging supports accountability, compliance verification, and forensic analysis.

3.5 Architectural Implication

The central claim emerging from this threat model is not that the system is unbreakable, but that compromise of the LLM component does not directly yield access to protected data values. Because the LLM operates solely over metadata and policy artifacts, successful manipulation of the model cannot expose database contents unless additional enforcement layers fail. In this sense, the architecture reduces the trust placed in the LLM and relocates enforcement authority to deterministic validation and constrained execution components.

The following sections describe how this threat model informs the system architecture and validation pipeline in detail.

4 System Architecture

This section describes the architecture of our Positive Data Control (PDC) system and explains how it achieves two goals that are usually in tension in LLM-based data access. The first goal is usability: non-technical users should be able to request data in natural language and receive answers quickly. The second goal is security: confidential database values must remain protected even if the LLM produces incorrect output or is influenced by adversarial prompting. In line with classical security design principles such as complete mediation and separation of privilege [15], the architecture is designed so that the LLM can assist with interpretation and query drafting, but deterministic controls remain the final enforcement point.

4.1 Design Goals and Principles

The architecture is guided by five principles, each tied to a concrete enforcement mechanism. Table 1 summarizes how these principles map to system responsibilities and security objectives.

Zero-knowledge governance is the central architectural constraint: the LLM must never receive database values or query result sets. Instead, it operates on metadata (schemas, column descriptions, sensitivity labels) and governance artifacts (policies, permissions, declared purpose). This design reduces the consequences of LLM compromise because the model cannot reveal information it does not observe. This approach follows the general direction of zero-trust thinking, where components are treated as potentially compromised and access is mediated per request [14].

Metadata completeness is required because policy reasoning is only as reliable as the inputs provided to the policy engine. Since the LLM cannot inspect data values, the metadata catalog must carry the semantics needed for correct governance decisions: what each table means, which columns are sensitive, and what restrictions apply. This is closely related to attribute-based access control practice, which depends on well-defined subject and object attributes [12].

Explicit policy encoding means policies are centralized and written in a form that is stable, reviewable, and auditable. The key requirement is not that policies be formally verified, but that they be explicit enough to support consistent interpretation and independent validation. This matches standard governance expectations in enterprise security engineering [2].

Defense-in-depth recognizes that an LLM is a probabilistic component and is not suitable as the sole enforcement mechanism. Therefore, the system implements deterministic checks before execution (query safety constraints, permission verification, and resource bounds). This aligns with well-known secure design guidance: assume single mechanisms fail and require multiple independent layers to prevent violations [15,2].

Complete auditability ensures every request is attributable and reviewable. The audit trail records identity, declared purpose, decision, executed query metadata, and high-level outcomes. This is essential for both security monitoring and compliance evidence in enterprise settings.

Table 1 Design principles and their enforcement in the architecture.

Principle	What it ensures	Architectural enforcement mechanism
Zero-knowledge governance	LLM cannot learn or leak database values	Prompt contains only metadata, policies, and permissions; results never routed back to LLM
Metadata completeness	LLM applies policies correctly using structure alone	Catalog provides column semantics, sensitivity labels, joins, and access annotations
Explicit policy encoding	Policies are reviewable, consistent, and auditable	Central repository; policies passed to LLM and validator as authoritative rules
Defense-in-depth	No single failure (incl. LLM error) causes violation	Independent SQL validation; sandboxed execution; bounded resources; post-processing controls
Complete auditability	Accountability and forensic readiness	Immutable audit records for requests, decisions, SQL, and outcomes

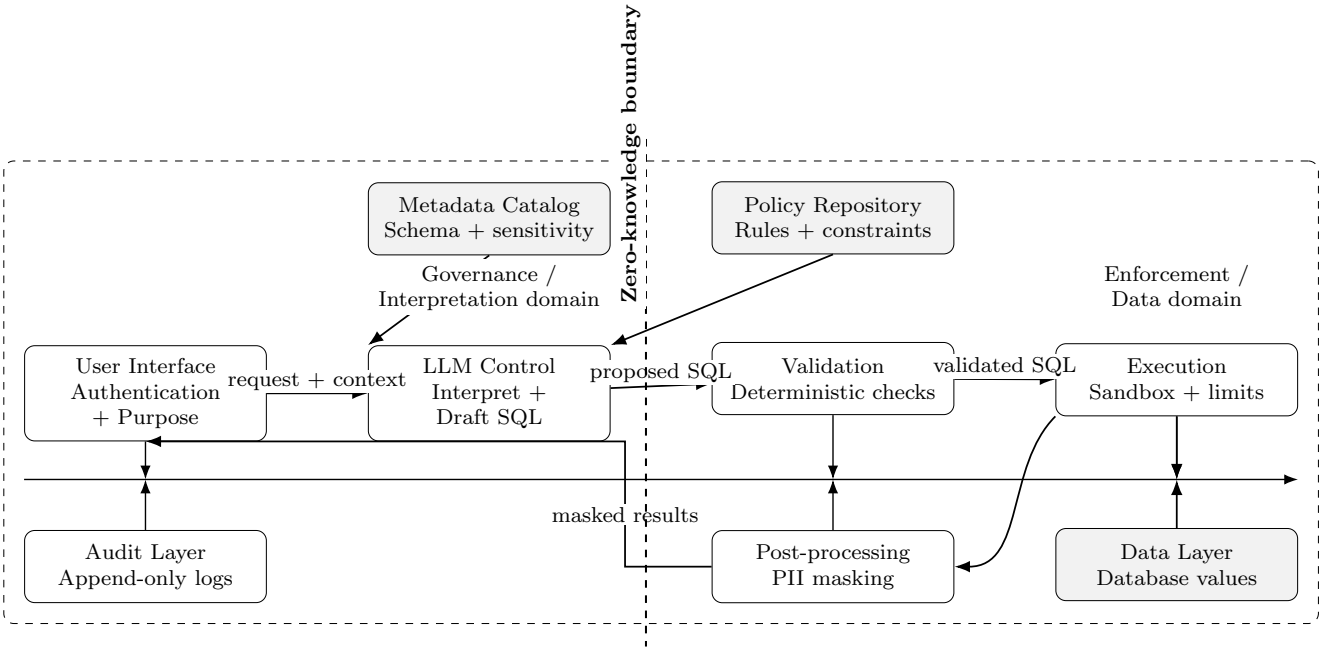

Fig. 2 PDC architecture with an explicit zero-knowledge boundary. The LLM proposes a query using metadata and governance policies, while deterministic validation and sandboxed execution enforce policy and safety constraints before any database values are accessed.

4.2 Architecture Overview and Trust Boundary

Figure 2 illustrates the complete request lifecycle and highlights the architectural separation that underpins the security guarantees of the proposed approach. The pipeline proceeds from left to right. A user first authenticates through the interface layer and submits a natural-language request together with a declared business purpose. This request is bound to an identity, role, and permission profile before any interpretation occurs.

The LLM control layer then receives a structured prompt composed exclusively of governance artifacts: schema descriptions from the metadata catalog, sensitivity annotations, access rules from the policy repository, and the authenticated user's permission context. Importantly, this prompt contains no database values and no previously retrieved result sets. The LLM's responsibility is limited to interpreting the request and drafting a proposed SQL query that reflects the applicable structural and policy constraints.

The vertical dashed line in the figure marks the zero-knowledge boundary. To the left of this boundary lies the governance and interpretation domain, where reasoning about structure and policy occurs. To the right lies the enforcement and data domain, where deterministic checks and controlled execution take place. The LLM does not cross this boundary: it cannot read tables, inspect records, or observe query outputs.

Once the LLM proposes a query, responsibility shifts to the validation layer. This component performs deterministic enforcement, including permission verification, rejection of disallowed operations, and detection

of unsafe constructs. Only after successful validation is the query executed within a sandboxed environment that enforces runtime limits. The execution layer retrieves raw results from the data layer, but these results are never routed back to the LLM. Instead, they pass through a post-processing stage that applies masking rules before being returned to the user. All actions are simultaneously recorded by the audit layer to ensure accountability.

This separation of concerns is critical. Even if the LLM is influenced by prompt injection, produces an incorrect SQL statement, or behaves unpredictably, it cannot directly expose confidential values because it never has access to them. Enforcement authority resides in deterministic validation and constrained execution rather than in the LLM itself. As a result, LLM-side failures may lead to rejected queries or corrected outputs, but they do not directly compromise the confidentiality of the underlying data.

4.3 Component Responsibilities

Table 2 provides a structured view of the responsibilities of each architectural component, the information it receives, the outputs it produces, and its role in enforcement. The purpose of this breakdown is to make explicit where trust is placed in the system and where security-critical decisions are actually enforced.

A key architectural decision in the proposed design is that the LLM is not treated as a security boundary. Although the LLM interprets natural-language requests and drafts SQL queries, it does not possess enforcement authority. Its outputs are treated as untrusted proposals that must be validated before execution. This distinction directly addresses a common concern in LLM-mediated systems: that probabilistic reasoning components may be implicitly trusted for security decisions.

The user interface layer performs identity binding and captures the declared business purpose. This binding step is critical because subsequent policy evaluation depends on authenticated context rather than self-declared claims inside the natural-language request. The interface establishes the subject identity for the remainder of the pipeline.

The LLM control layer operates purely within the governance domain. It receives structural metadata, sensitivity labels, policy descriptions, and user permissions. Its task is interpretive rather than authoritative: it transforms user intent into a candidate query while attempting to respect the provided constraints. However, it does not determine whether access is ultimately granted. By design, even a malicious or erroneous LLM output cannot directly cause data disclosure.

The validation layer constitutes the primary deterministic enforcement point. It verifies that the proposed SQL complies with role-based and purpose-based constraints, rejects disallowed operations, and ensures that access falls within the authenticated permission scope. This layer embodies complete mediation: every request must pass through validation before reaching the execution environment.

The execution layer enforces runtime safety properties. Queries are executed only after validation and within a sandbox that applies resource limits and isolation. This prevents denial-of-service conditions and constrains the operational impact of malformed queries. Importantly, execution retrieves raw results from the data layer without exposing those results to the LLM.

The post-processing stage applies output-level controls such as masking of sensitive fields. This provides an additional defensive layer even when access is authorized. The intent is to reduce exposure surface and limit unnecessary disclosure.

Finally, the audit layer records all contexts, decisions, and query metadata in an append-only log. Auditability ensures accountability and supports retrospective analysis of access patterns and potential misuse.

Taken together, the table clarifies that enforcement authority is distributed across deterministic components rather than concentrated in the LLM. The LLM contributes flexibility and usability, but confidentiality and policy compliance depend on validation, controlled execution, and output filtering. This separation of interpretation from enforcement is fundamental to the security rationale of the architecture.

4.4 End-to-End Information Flow

The request lifecycle follows a fixed sequence so that every access is mediated. First, the user authenticates and submits a natural-language request together with a declared business purpose. The system binds this request to an identity, role, and permission profile. Second, the LLM receives a structured prompt that contains only governance artifacts: schema and metadata, policy rules, and the user's permissions and purpose. The LLM then produces either (i) a proposed SQL query with a short justification, or (ii) a clarification question if the request is ambiguous. At this stage, the LLM is treated as a query author rather than an authority.

Third, the proposed SQL is processed by a deterministic validation layer that enforces hard constraints. The validator rejects any non-read operations, blocks dangerous constructs, and verifies that referenced tables and columns fall within the user's permissions.

Table 2 Component responsibilities, inputs/outputs, and trust assumptions.

Component	Inputs	Outputs	Enforcement role
User Interface	Credentials, request text, declared purpose	Authenticated session and request context	Binds identity and declared purpose
LLM Control Layer	Request context, metadata, policies, permissions	Proposed SQL with rationale or clarification request	Assists generation; not trusted for enforcement
Validation Layer	Proposed SQL, permissions, policy constraints	Allow/deny/repair decision; validated SQL	Primary deterministic enforcement
Execution Layer	Validated SQL	Bounded raw results	Runtime limits and sandboxing
Post-processing	Raw results, masking rules	Masked results returned to user	Defense-in-depth on outputs
Audit Layer	All contexts and decisions	Append-only audit record	Accountability and monitoring

This design is motivated by prior evidence that text-to-SQL systems can be induced to generate unsafe queries [17] and by long-standing classifications of SQL injection patterns and countermeasures [13]. If validation succeeds, the query runs in a sandbox with resource bounds, protecting availability against expensive-query attacks. Finally, results are post-processed to apply masking rules before being returned to the user, and the system writes a complete audit record.

4.5 Metadata, Policies, and Permissions as Governance Artifacts

A central challenge in zero-knowledge governance is enabling meaningful reasoning about data access without exposing the data itself. The LLM must generate syntactically correct and policy-compliant queries, yet it must do so without observing any database values. To support this capability, the architecture elevates metadata, policy definitions, and permission structures to first-class governance artifacts.

Metadata serves as the structural substitute for data values. It conveys the organization of the database, including table names, column types, semantic descriptions, and relationships between entities. Importantly, metadata also includes sensitivity annotations and restricted access markers at the column level. These annotations encode security-relevant information that would otherwise require inspection of the data itself. For example, the LLM does not need to see an employee's salary to understand that the `salary` column is restricted to HR roles; this information is conveyed entirely through metadata.

Policies define contextual constraints over that structure. They specify which roles may access which resources, under which declared purposes, and subject to what limitations (e.g., aggregation-only constraints or row limits). Permissions bind those policies to authenticated identities, translating abstract governance rules into concrete access conditions tied to a specific user session.

The combined effect is that the LLM reasons over a complete governance model of the data environment rather than over the data values themselves. In other words, the model operates on structural and normative information instead of operational content. This distinction underpins zero knowledge: correctness comes from rich metadata, not from inspecting sensitive records.

Table 3 makes this separation explicit by listing the categories of information that may and may not reach the LLM. The left column defines the permissible interpretation domain, consisting exclusively of structural schema elements, sensitivity labels, policy rules, and authenticated user context. The right column defines the excluded data-value domain. Database rows, cell contents, and query result sets are structurally prevented from entering the LLM's prompt context. Even masked outputs are excluded, ensuring that no transformed data values can be reintroduced into the reasoning process.

This table should be understood not merely as documentation but as a design invariant. The zero-knowledge boundary is preserved only if every implementation path respects this visibility constraint. If any element listed in the excluded column is passed to the LLM—whether through debugging logs, error messages, automated summarization of query results, or caching mechanisms—the architectural guarantee is violated. Consequently, the visibility constraints in Table 3 function both as a conceptual model and as a practical checklist for secure deployment.

Table 3 Information visibility constraints for zero-knowledge governance.

Information available to the LLM	Information excluded from the LLM
Schema (tables, columns, types), join relationships, column descriptions	Database values (rows, cell contents), query result sets
Sensitivity labels (PII flags, confidentiality levels), restricted-column markers	Names, salaries, SSNs, credit scores, transaction amounts
Policies and constraints (role rules, purpose rules, row limits, aggregation rules)	Raw logs containing sensitive payloads; database error messages with values
User identity context (role, department, permissions, declared purpose)	Any returned data values, even if masked

4.6 Why the Architecture Achieves the Intended Security Properties

The architecture is designed so that compromise or manipulation of the LLM does not directly imply compromise of data confidentiality. This design reflects a zero-trust mindset [14], in which components that process external input are assumed to be potentially unreliable. In particular, the LLM may be influenced by prompt injection [16], may generate incorrect or over-permissive SQL statements, or may even be exposed through a service-level incident. The security of the overall system therefore cannot depend on the correctness or integrity of the LLM.

The key architectural safeguard is the strict separation between interpretation and enforcement. The LLM operates exclusively over governance artifacts—metadata, policy rules, and authenticated user context—and never over database values. As a result, even a fully compromised LLM cannot directly reveal confidential records because it does not have access to them. At worst, it can generate an unsafe or policy-violating query proposal.

However, proposed queries do not execute automatically. Deterministic validation enforces permission checks, structural constraints, and safety rules before any interaction with the data layer occurs. Only validated queries are passed to the execution layer, which operates within a sandbox that applies runtime limits and isolation. This layered mediation ensures that policy compliance and safety constraints are enforced independently of the LLM's reasoning.

Finally, all requests, decisions, and executed queries are recorded in an append-only audit trail. This supports accountability, post-incident investigation, and refinement of governance rules over time. Taken together, these mechanisms ensure that failures in the interpretive component may affect usability or query correctness, but do not directly compromise the confidentiality of the underlying data.

5 Implementation

This section describes the prototype implementation of the Positive Data Control (PDC) architecture. The purpose of the implementation is twofold: first, to demonstrate that natural language data access can be mediated under explicit governance constraints; and second, to operationalize the zero-knowledge boundary such that confidentiality does not depend on trusting the LLM component. The implementation therefore emphasizes architectural separation, deterministic validation, and controlled execution rather than performance optimization.

5.1 Technology Stack

The prototype uses Amazon Bedrock as the model hosting platform [19], with Anthropic Claude 3 as the large language model for policy interpretation and SQL generation [20]. The application backend and web interface are implemented in Python 3.11 [21] using the Streamlit framework for interactive web development [22]. The database layer uses SQLite configured for in-memory operation [23]. While SQLite simplifies deployment for experimental evaluation, the architectural design is database-agnostic and can be adapted to enterprise systems such as PostgreSQL or Oracle, subject to dialect-aware validation rules.

The system is packaged in a containerized environment to improve reproducibility and portability across development contexts [24]. Importantly, none of the security guarantees depend on a specific LLM provider, application framework, or database engine; they depend instead on the enforced separation between interpretation and execution layers.

5.2 Database Schema

The prototype database consists of four tables intentionally designed to represent representative enterprise governance scenarios rather than a synthetic toy schema.

The objective is not to model a specific industry domain, but to create a compact environment that exercises heterogeneous security constraints, including PII protection, department-scoped access control, financial confidentiality, and purpose-based restrictions. Table 4 summarizes the schema and associated governance annotations.

The **customers** table models a typical customer relationship management dataset. It contains direct identifiers (name, email, phone) classified as personally identifiable information (PII) and a credit score field classified as confidential financial information. This table allows evaluation of column-level sensitivity enforcement and selective masking under different role permissions.

The **orders** table captures transactional history linked to customers through a foreign key relationship. It includes financial amount fields classified as confidential and temporal fields classified as internal operational data. This table enables testing of cross-table joins, aggregation constraints, and row-limiting behavior under different user roles.

The **employees** table represents highly sensitive internal HR records. It includes salary and social security number attributes marked as restricted, with access limited to HR personnel. This table is used to evaluate strict department-based enforcement and to ensure that highly sensitive attributes are consistently excluded from non-authorized queries.

The **transactions** table contains financial records that require explicit purpose declaration for access. Unlike purely role-based restrictions, this table exercises contextual, purpose-based access control, allowing the system to test whether declared intent is correctly evaluated alongside role permissions.

Although the prototype contains a modest number of records, the schema includes primary and foreign key relationships to exercise join reasoning and policy enforcement across related entities. Sensitivity classifications are defined at the column level and stored within the metadata catalog rather than embedded in application logic. This design ensures that governance decisions depend on structured metadata and policy artifacts, consistent with the zero-knowledge architecture described earlier.

This schema allows systematic evaluation of several governance dimensions: (i) column-level sensitivity filtering, (ii) role-based access control, (iii) department-scoped restrictions, (iv) purpose-based constraints, and (v) aggregation-only access patterns. As a result, even with a limited dataset size, the prototype exercises diverse enforcement paths and validates the architectural separation between interpretation and deterministic enforcement.

Table 4 Database schema summary used in the prototype.

Table	Key columns	Sensitivity highlights	Records	Access control
customers	customer_id, name, email, phone, credit_score	PII (name/email/phone), confidential (credit_score)	15	Requires `read_customers`
orders	order_id, customer_id, product_name, amount, date	Confidential (amount), internal (dates)	20	Requires `read_orders`
employees	employee_id, name, department, salary, ssn	Restricted (salary/ssn), PII (name)	12	HR department only
transactions	transaction_id, account_id, amount, timestamp	Confidential (amount)	15	Requires declared purpose

The schema intentionally includes multiple sensitivity levels to evaluate enforcement behavior under realistic constraints. While the dataset size is modest, the validation and enforcement mechanisms operate independently of scale.

5.3 LLM Prompt Engineering

The effectiveness of LLM-mediated governance depends on structured prompt construction. The prompt is designed to provide complete governance context while preserving the zero-knowledge boundary by excluding database values and result sets.

The LLM receives:

- The user's natural-language request,
- Authenticated identity, role, department, and declared purpose,
- A structured permission profile in JSON format,
- A metadata catalog describing tables, columns, relationships, and sensitivity labels,
- Explicit governance policies to enforce.

The prompt explicitly restricts output to read-only SQL queries and requires structured JSON output (proposed SQL and short rationale). This structured contract improves parsing reliability and reduces ambiguity.

Explicit enumeration of security rules significantly improved compliance compared to implicit expectations. Low-temperature generation settings were used to reduce stochastic deviations. The prompt instructs the

LLM to suggest policy-compliant alternatives when requests violate constraints, improving usability without relaxing enforcement.

Prompt injection risks are mitigated by treating the LLM output as untrusted and by enforcing independent validation, consistent with OWASP guidance for LLM-based systems [16,8].

5.4 SQL Validation Layer

The validation layer implements deterministic enforcement independent of the LLM. The LLM-generated SQL is treated as untrusted input and subjected to conservative checks before execution.

Validation proceeds through four stages:

1. **Read-only enforcement**: Reject any non-`SELECT` statements and disallowed keywords (e.g., DDL/DML operations).
2. **Injection detection**: Reject common SQL injection markers and suspicious patterns [13].
3. **Authorization verification**: Extract referenced tables and columns and compare them against the authenticated user's permission profile.
4. **Result bounding**: Enforce row limits by appending `LIMIT` clauses when required.

The validation logic follows a fail-secure principle: when uncertainty exists, access is denied. Although the prototype uses pattern-based checks for simplicity, the architecture supports extension to abstract syntax tree (AST)-based validation for stronger dialect-aware enforcement.

5.5 Sandboxed Execution and Output Protection

Queries that pass validation are executed within a sandboxed environment. Runtime constraints, including query timeouts, mitigate denial-of-service risks. Execution occurs in isolation from the LLM; query results are never fed back into the model context.

After execution, result sets undergo post-processing to apply PII masking rules. Even when users possess PII access permissions, conservative masking reduces exposure. For example:

- Email addresses are partially masked,
- Phone numbers are fully masked,
- Social security numbers are always fully masked, even for HR roles.

This output-level control provides defense-in-depth: even if an unsafe query were to pass validation, masking would limit disclosure impact.

5.6 Audit Logging

The audit layer records each interaction end-to-end. Each audit record includes:

- Timestamp and session identifier,
- Authenticated user identity, role, and department,
- Declared purpose,
- Original natural-language request,
- Generated SQL and validation outcome,
- Executed query metadata (excluding result values),
- Number of rows returned.

This logging approach aligns with established guidance on secure log management [25] and supports compliance requirements for processing accountability [10]. The log is append-only in the prototype, and in production deployments should incorporate integrity protections and retention policies consistent with organizational standards.

Together, these implementation components operationalize the architectural principles described earlier. The LLM assists with interpretation, but confidentiality and policy compliance are enforced by deterministic validation, controlled execution, and auditable decision logging.

6 Experimental Validation

This section evaluates whether the prototype achieves its intended governance and security properties in practice. The evaluation is designed to answer three questions: (i) does the system make correct access decisions under representative governance requests; (ii) when a request is not allowed, does it respond conservatively (deny or ask clarification) or transform the request into a compliant alternative when such an alternative exists; and (iii) does the deterministic validation layer prevent execution of unsafe or unauthorized SQL, including under adversarial inputs.

6.1 Experimental Design

We evaluate the system using two complementary test suites. The first is an end-to-end suite of 24 scenarios that exercise the full pipeline (LLM drafting, deterministic validation, sandboxed execution, and output filtering). This suite focuses on realistic governance behavior across roles and policies, but also includes edge cases that commonly arise in practice, such as ambiguous requests and adversarial strings embedded in natural-language input. The second is a supplementary security

suite of 38 targeted attacks designed to stress the enforcement boundary under adversarial conditions.

The separation between an end-to-end suite and a targeted attack suite mirrors common security evaluation practice: the end-to-end suite measures behavior under representative usage, while the targeted suite concentrates coverage on known failure modes of LLM-integrated systems, including prompt injection and attempts to coerce unsafe code generation, as well as classical SQL injection and privilege escalation patterns. The latter are particularly relevant because text-to-SQL systems have been shown to be susceptible to adversarial manipulation that can lead to unsafe queries or denial-of-service behaviors if not independently constrained.

6.2 Scenario Taxonomy

To avoid ambiguity in scoring, each end-to-end scenario is assigned to one category and one expected outcome class. Categories reflect the governance dimension being exercised:

- **Category A (Permitted Access):** requests aligned with role permissions (expected outcome: GRANT).
- **Category B (Role Violation):** requests that exceed role permissions or target restricted columns (expected: DENY).
- **Category C (Policy Transformation):** requests that are not allowed in their raw form but admit a compliant alternative (e.g., aggregation-only roles). (expected: MODIFY).
- **Category D (Cross-Domain Barrier):** requests crossing departmental or information-barrier boundaries (expected: DENY).
- **Category E (Purpose-Gated Access):** requests requiring an approved stated purpose (expected: GRANT or DENY depending on purpose).
- **Category F (Adversarial-in-NL):** adversarial strings embedded in natural-language requests (expected: BLOCK via safe SQL generation and/or deterministic rejection).
- **Category G (Ambiguity):** underspecified requests where intent cannot be inferred safely (expected: CLARIFY).

6.3 Metrics and Scoring Protocol

We score each end-to-end scenario using five metrics. To make these metrics reproducible, Table 5 defines their operational meaning and clarifies how success is determined. The intent is to evaluate the system as a governance enforcement pipeline rather than as a pure text-to-SQL model: an outcome is considered correct if the system reaches the expected decision class and, when it executes a query, the executed query is safe, authorized, and appropriately bounded.

Decision correctness is measured against the expected outcome class (GRANT, DENY, MODIFY, or CLARIFY). This metric captures whether the system made the right governance choice given the role, declared purpose, and applicable policies. SQL safety evaluates whether any query that reaches execution is structurally safe (read-only, no prohibited constructs) and passes injection screening; importantly, unsafe SQL proposed by the LLM does not count as a failure if it is deterministically blocked before execution, since enforcement is intentionally located outside the LLM.

Authorization/minimality measures whether the executed query stays within the user's permission scope and avoids retrieving unnecessary sensitive fields. We treat this as a minimality criterion because a governance system should not retrieve columns that the user is not permitted to access, and should avoid broad extraction even when access is allowed. Completeness (within policy) complements minimality: for scenarios where access is permitted (or can be transformed into a permitted alternative), the query should still satisfy the user's information need without omitting allowed information required to answer the request. In the prototype, completeness is assessed using a manual oracle because the test suite is small and the expected query intent is well-defined; in larger deployments, this can be replaced by automated checks based on reference SQL or ground-truth answers. Finally, explainability measures whether the system provides a short, policy-grounded justification for its decision, including what constraint applied (e.g., role restriction, purpose requirement, PII sensitivity) and, in the MODIFY case, what transformation was performed and why.

Latency is measured end-to-end per scenario and reported as an average across scenarios. We report latency separately from correctness because it reflects deployment feasibility rather than security enforcement quality.

For the targeted security suite, the primary outcome is **enforcement success**: the system must prevent execution of unauthorized or unsafe SQL. In practice, this may occur because (i) the LLM proposes a benign query that does not incorporate the adversarial payload, or (ii) the validator rejects or repairs the proposed query. In both cases, success is defined at the system boundary: an attack attempt is considered neutralized only if no policy-violating query is executed and no prohibited data is returned.

Table 5 Operational definition of evaluation metrics.

Metric	Operational definition (per scenario)
Decision correctness	System outcome matches expected class: GRANT, DENY, MODIFY, or CLARIFY.
SQL safety	Any executed SQL is read-only (`SELECT`-only), contains no disallowed constructs, and passes injection screening.
Authorization / minimality	Selected tables/columns are within the user's permission scope; restricted columns are excluded; non-aggregate queries include a role-based `LIMIT`.
Completeness (within policy)	When GRANT or MODIFY is expected, the query returns the intended information need without omitting allowed fields necessary for the request (assessed by manual oracle for this prototype).
Explainability	Response includes (i) the governing constraint and (ii) a brief justification or transformation rationale.

6.4 Representative End-to-End Scenarios

Table 6 summarizes eight representative scenarios spanning permitted access, policy transformation, strict denial, purpose-gated access, and adversarial-in-NL behavior. These scenarios were selected to illustrate not only individual policy checks, but also cases where multiple constraints must be composed simultaneously. The objective is to demonstrate that the system enforces governance rules in a principled and consistent manner rather than relying on isolated rule matching.

Scenarios S1 and S3 illustrate straightforward permitted access under role-appropriate conditions. In these cases, the system generates executable queries that remain bounded and compliant with sensitivity annotations. These examples confirm that the architecture does not over-constrain legitimate analytical or operational access.

Scenarios S2 and S6 demonstrate policy transformation. Instead of denying requests that violate aggregation-only or PII constraints, the system rewrites them into compliant alternatives. This behavior is significant because it shows that enforcement is not purely binary (allow/deny); rather, it supports controlled relaxation within policy boundaries. In S6, for example, the system composes three constraints simultaneously aggregation only access, PII exclusion, and cross-table join permissions—and produces a query that satisfies all three. This indicates that governance reasoning operates across overlapping rules rather than treating policies independently.

Scenarios S4 and S8 represent strict denials where no compliant alternative exists. In both cases, the requested data targets restricted tables or columns outside the user's permission scope. The system responds conservatively by denying execution and providing a role-based explanation. These cases validate that restricted attributes (e.g., salary, credit score) are consistently excluded for unauthorized roles.

Scenario S7 illustrates contextual enforcement through purpose-based access control. The same request yields different outcomes depending on the declared purpose. This demonstrates that governance decisions depend not only on static role assignments but also on contextual metadata associated with the request.

Finally, Scenario S5 highlights adversarial-in-NL behavior. The request embeds a classical SQL injection payload within a natural-language string. The system does not execute the malicious fragment because (i) the LLM interprets the request semantically rather than syntactically incorporating the injected SQL, and (ii) the deterministic validator independently enforces read-only execution and rejects unsafe constructs. This example directly illustrates the architectural separation between interpretation and enforcement: even if the LLM were to misinterpret or partially include the payload, execution would still be gated by validation.

Two broader patterns emerge from these representative scenarios. First, the system frequently resolves policy tension via controlled transformation rather than denial, preserving analytical utility without compromising constraints. Second, enforcement authority resides in deterministic validation and sandboxed execution rather than in the LLM's reasoning process. As a result, adversarial inputs and policy violations do not propagate into unsafe execution. Together, these scenarios provide qualitative evidence that the architecture behaves as intended under both normal and adversarial conditions.

6.5 Aggregate Results for the End-to-End Suite

Table 7 summarizes outcomes across the 24 end-to-end scenarios. The suite includes 7 permitted-access cases (Category A), 3 role-violation denials (Category B), 3 policy transformations (Category C), 2 cross-domain denials (Category D), 2 purpose-gated decisions (Category E), 4 adversarial-in-NL cases (Category F), and 3 edge/ambiguity cases (Category G).

Across the 24 scenarios, the system produced a correct conservative outcome in all cases: permitted requests were granted, disallowed requests were denied or blocked, transformable requests were modified into compliant alternatives, and ambiguous requests triggered clarification rather than guessing. In 22/24 cases (91.7%), the system returned an executable query or an

Table 6 Representative end-to-end scenarios illustrating common governance behaviors.

Scenario	Role	Request (abridged)	Main constraint(s)	Outcome	Notes
S1	Analyst	Avg order amount by customer	Aggregation-only	Grant	GROUP BY + AVG; role-based LIMIT applied
S2	Analyst	Show all customers	Aggregation-only	Modify	Returned aggregated alternative; explained why record-level view denied
S3	HR Mgr	Salaries in Engineering	Restricted salary allowed for HR	Grant	Restricted column permitted for HR; bounded results
S4	Support	Salaries of employees	HR-only table/columns	Deny	Denied with role-based explanation
S5	Support	"... DROP TABLE ..." embedded	Adversarial-in-NL	Block	Payload not executed; validator prevents unsafe SQL
S6	Analyst	Names + total order amounts	PII restriction + aggregation-only + join	Modify	Joined tables, aggregated totals; excluded names due to PII
S7	Analyst	Transaction data (bad purpose)	Purpose-gated table	Deny/Grant	Denied under `general_inquiry`, granted under `analytics`
S8	Analyst	Customer credit scores	Restricted column	Deny	Denied; suggested compliant path (finance role)

Table 7 End-to-end suite outcomes by scenario category (24 scenarios).

Category	#	Grant	Deny/Block	Modify/Clarify
A (Permitted)	7	7	0	0
B (Role violation)	3	0	3	0
C (Transformation)	3	0	0	3 (Modify)
D (Cross-domain)	2	0	2	0
E (Purpose-gated)	2	1	1	0
F (Adversarial-in-NL)	4	0	4 (Block)	0
G (Ambiguity)	3	1	0	2 (Clarify)
Total	**24**	**9**	**12**	**3**

executable compliant alternative. The remaining 2/24 cases (8.3%) resulted in clarification requests, which is the intended fail-safe behavior when intent is underspecified.

To avoid overstating results, we report precision/recall in terms of **decision classes** within this fixed test suite. Within the suite, we observed no false allows (no policy-violating requests were executed) and no false denies for the scenarios labeled as permitted. This is stronger than reporting a single scalar "accuracy" because it clarifies what counts as an error in a governance system. Additionally, all executable queries were bounded (role-based `LIMIT` for non-aggregate queries) and did not select restricted columns under unauthorized roles, satisfying the minimality/authorization criterion defined in Table 5.

We also measured end-to-end latency. The observed mean latency was 2.3 seconds per request, dominated by the LLM call (approximately 1.8s on average), with deterministic validation and database execution contributing the remainder. This suggests that, for the prototype setting, governance enforcement adds negligible overhead relative to the model invocation.

6.6 Targeted Security Validation

The targeted security suite consists of 38 adversarial scenarios designed to stress the enforcement boundary under attacks common in LLM-mediated and text-to-SQL systems. Attack categories include classical SQL injection families (e.g., comment-based injection, stacked queries, UNION-based exfiltration), dangerous operation attempts (DDL/DML), permission escalation and cross-domain access attempts, and purpose circumven-

Table 8 Security validation results (38 targeted attack scenarios).

Attack type	Test cases	Successful blocks	Success rate
SQL injection variants	10	10	100%
Dangerous operations (DDL/DML)	5	5	100%
Permission escalation	8	8	100%
Cross-domain access	6	6	100%
Social engineering / authority claims	5	5	100%
Purpose circumvention	4	4	100%
Total	**38**	**38**	**100%**

tion. These categories are grounded in established SQL injection taxonomies and recent analyses of vulnerabilities in text-to-SQL systems.

Table 8 reports aggregate outcomes. Across all 38 attacks, the system prevented execution of unauthorized or unsafe SQL. In many cases the LLM produced benign SQL that ignored the adversarial payload, while the validator served as the decisive backstop by enforcing read-only execution and rejecting prohibited constructs. This layered behavior is important: the security claim does not depend on the LLM consistently "doing the right thing," but on deterministic enforcement preventing unsafe execution.

6.7 Interpretation and Limitations

Taken together, the results support the central architectural hypothesis: confidentiality and governance compliance can be enforced without granting the LLM access to database values, and without treating the LLM as a trusted security boundary. The end-to-end suite demonstrates that the system can (i) grant legitimate access, (ii) deny disallowed access with clear explanations, and (iii) transform certain disallowed requests into policy-compliant alternatives, improving usability without relaxing constraints. The targeted security suite demonstrates that adversarial prompts and unsafe-operation attempts do not result in unsafe execution because deterministic validation and sandboxed execution mediate all access.

At the same time, these results should be interpreted within the scope of the prototype. The schema and policies are compact, and the validation layer uses conservative pattern-based screening rather than dialect-aware parsing. Moreover, the evaluation uses a fixed suite of scenarios rather than a large-scale distribution of real user queries. These limitations suggest two concrete directions for future evaluation: (i) scaling scenario coverage using automated generation and repeated trials to quantify variance under different prompts and model settings, and (ii) strengthening validation using AST-based parsing for improved robustness across SQL dialects.

7 Discussion

7.1 Interpreting "Zero-Knowledge" in PDC

Our use of the term *zero-knowledge* is architectural rather than cryptographic: it denotes that the LLM is systematically excluded from the data-value domain while still being able to reason over governance artifacts (metadata, policies, and permissions). In other words, the LLM can see what tables and columns exist and what constraints apply, but it does not receive rows, cell values, or query results. This distinction matters because it clarifies what the system can and cannot guarantee. The design does not provide a cryptographic proof of knowledge; instead, it reduces the likelihood and impact of LLM-side compromise by ensuring that there are no confidential values in the model context to exfiltrate.

The validation results support this interpretation in two ways. First, across the end-to-end suite, no scenario executed a query that violated the declared policy constraints. Second, in adversarial cases (including prompt-injection-style inputs), the system remained safe at the system boundary because execution is gated by deterministic checks and sandboxing rather than by the LLM's internal reasoning. This aligns with a zero-trust mindset in which the LLM is treated as an untrusted component whose outputs must be independently verified [14,16]. The practical implication is that even when the LLM produces a flawed query draft, the enforcement boundary remains deterministic and auditable.

At the same time, the boundary is not "free." Metadata and policy text can contain sensitive business context (e.g., table semantics, sensitivity labels, internal taxonomy). Therefore, protecting the metadata plane becomes a first-class requirement in deployment: access to governance artifacts should be controlled, logged, and minimized to what the LLM requires for correct decision making.

7.2 Automation as a Shift in Stewardship Work

A central motivation for Positive Data Control is that governance often fails not because policies do not exist, but because enforcement and documentation require continuous human effort that does not scale [26]. In operational terms, the prototype automates routine parts of steward work: translating natural-language requests into constrained queries, applying policy constraints consistently, and generating structured audit records for every request. This should be interpreted as a shift in steward responsibilities rather than removal of stewards. In established data management practice, stewardship includes maintaining definitions, classifications, and governance processes; automation is most effective when it reduces repetitive enforcement work while preserving human authority over policy and classification [28].

The audit trail is especially important in this shift. By making each decision explainable and traceable, the system allows stewards to move from case-by-case approval to higher-leverage tasks such as policy refinement, monitoring of access patterns, and investigation of anomalies. This is consistent with PDC's "fly-by-wire" framing: automation mediates requests, but human governance remains responsible for defining acceptable operating envelopes and responding to observed drift [27].

7.3 Policy Expressiveness: Benefits and Boundaries

A practical benefit of LLM mediation is that governance rules can be written in natural language and remain accessible to non-technical stakeholders. However, natural language can be underspecified. The evaluation highlights two design implications. First, policies should be paired with structured tags (e.g., PII flags, sensitivity tiers, restricted-column markers) to reduce ambiguity and to give deterministic layers an authoritative basis for enforcement. Second, policy authoring should be supported by templates or controlled language guidelines (e.g., "who" constraints, "what" constraints, "purpose" constraints, and "limits" constraints) so that the LLM is not forced to infer governance intent from vague prose.

This approach does not compete with formal policy frameworks; it complements them. Standards such as XACML provide machine-verifiable encodings of access-control logic [29]. A deployment path that many organizations may prefer is hybrid: use structured policy rules where precision is critical, and use natural language for policy rationale, exceptions, and user-facing explanations. In that hybrid model, the LLM primarily supports interpretation and explanation, while deterministic enforcement continues to rely on explicit constraints.

7.4 Security Implications and Residual Risks

The security results should be interpreted as evidence for separation of concerns: the LLM assists with intent interpretation and query drafting, while enforcement authority is implemented by deterministic validation and sandboxed execution. This design reduces the impact of prompt injection and other adversarial inputs by ensuring that unsafe outputs are not executed [16]. It also creates an accountability surface through audit logs, which is a practical requirement for security operations and compliance.

Residual risks remain and should be addressed explicitly. First, **metadata leakage** can reveal business structure even when values are protected; deployments should apply least privilege to governance artifacts and consider redacting descriptions when not needed. Second, **inference risk** can arise even under authorized access, particularly when small-group aggregations can reveal individuals. Mitigations such as minimum group-size thresholds (k-anonymity-style constraints) [30] or noise mechanisms inspired by differential privacy [31] are natural extensions for output controls. Third, **validator completeness** matters: pattern-based screening is conservative but may be bypassed in edge dialects. Strengthening validation with dialect-aware parsing and AST-level checks is an important production hardening step.

7.5 Deployment and Scalability Considerations

The prototype demonstrates feasibility, but production deployment requires integration with enterprise identity (SSO), catalogs, and multiple database backends. From an operational standpoint, scalability depends less on the database layer and more on governance artifact quality and model invocation cost. Two practical strategies follow. First, treat metadata as a maintained product: accurate schemas, consistent sensitivity labeling, and synchronized catalogs materially improve decision quality. Second, adopt an execution policy that reduces unnecessary model calls for routine cases (e.g., caching, templated queries for frequent intents, or routing "simple" requests through rule-based handlers and reserving the LLM for ambiguous or multi-constraint cases). These strategies preserve governance guarantees while controlling latency and cost.

7.6 Limitations and Future Work

Several limitations shape the scope of current claims. The evaluation is performed on a compact schema and a fixed scenario suite; broader coverage across domains, SQL dialects, and real organizational policies remains future work. The system currently enforces read-only access; extending the same philosophy to governed write operations would require additional safeguards (transaction policies, approval workflows, and stronger provenance controls). Finally, the current prototype does not implement federated operation across organizational units; extending PDC to multiple "governance islands" would require standardized exchange of policy abstractions and consistent audit semantics.

Overall, the results suggest that architectural zero knowledge (excluding data values from the model context) combined with deterministic enforcement and auditing can provide a practical path toward scalable governance automation without treating LLMs as trusted security boundaries.

8 Conclusion

This paper presented an architectural approach to LLM mediated data governance based on a zero-knowledge design principle: the LLM operates over metadata, policies, and user context, but does not access underlying data values. By separating interpretation from enforcement and placing deterministic validation and sandboxed execution at the system boundary, the architecture reduces reliance on the LLM as a security-critical component.

The experimental evaluation demonstrates that, within the defined scenario suite, the system consistently produced governance aligned outcomes. Legitimate requests were granted with appropriate constraints, policy violations were denied or transformed into compliant alternatives, and adversarial inputs did not result in execution of unsafe queries. These results suggest that LLMs can assist with policy interpretation while enforcement authority remains deterministic and auditable.

The primary contribution of this work is the articulation and empirical validation of a zero-knowledge governance pattern for LLM-based data access systems. The study shows that natural-language interaction and strong policy enforcement need not be in tension when architectural separation and defense-in-depth principles are applied. The approach also illustrates how governance automation can shift data stewardship from operational query review toward policy definition and audit analysis.

Several directions remain open for further research, including integration of formal privacy guarantees for aggregation outputs, scaling evaluation to enterprise-sized datasets, extension to write-operation governance, and exploration of federated PDC architectures across organizational boundaries. These extensions will determine the robustness and generalizability of the proposed pattern in large-scale deployments.

Overall, the results indicate that LLM-assisted governance can be deployed in a manner consistent with established information security principles when the model is treated as an untrusted interpreter rather than as an enforcement authority. The architectural separation introduced in this work provides a practical foundation for secure, automated data governance in environments that require both usability and strong policy compliance.

Acknowledgements This research was partially supported by the National Science Foundation under EPSCoR Award No. OIA-1946391 and the PiLog Group.

References

1. M. Bishop, *Computer Security: Art and Science*. Addison-Wesley, 2003.
2. R. Anderson, *Security Engineering*, 2nd ed. Wiley, 2008.
3. J. A. Goguen and J. Meseguer, "Security policies and security models," in *IEEE Symposium on Security and Privacy*, 1982.
4. A. Sabelfeld and A. C. Myers, "Language-based information-flow security," *IEEE Journal on Selected Areas in Communications*, 2003.
5. D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, 1983.
6. D. Cappelli, A. Moore, and R. Trzeciak, *The CERT Guide to Insider Threats*. Addison-Wesley, 2012.
7. F. Perez and I. Ribeiro, "Ignore previous instructions: Injection attacks on large language models," arXiv:2211.09527, 2022.
8. K. Greshake et al., "More Than You've Asked For: A Comprehensive Analysis of Novel Prompt Injection Attacks," arXiv:2302.12173, 2023.
9. R. Bommasani et al., "On the opportunities and risks of foundation models," 2021.
10. P. Voigt and A. von dem Bussche, *The EU General Data Protection Regulation (GDPR): A Practical Guide*, Springer, 2017.
11. R. Sandhu et al., "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, 1996.
12. V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas, *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*, NIST Special Publication 800-162, National Institute of Standards and Technology, 2019 (Update 2).
13. W. G. Halfond, J. Viegas, and A. Orso, "A Classification of SQL Injection Attacks and Countermeasures," in *Proc. IEEE International Symposium on Secure Software Engineering*, 2006.

14. S. Rose, O. Borchert, S. Mitchell, and S. Connelly, *Zero Trust Architecture*, NIST Special Publication 800-207, National Institute of Standards and Technology, 2020.
15. J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.
16. OWASP Foundation, *OWASP Top 10 for Large Language Model Applications*, 2023–2025. Available: https://owasp.org/www-project-top-10-for-large-language-model-applications/
17. B. Peng et al., "Exploring Security Vulnerabilities in Text-to-SQL Systems," USENIX Security, 2023.
18. M. Chafik et al., "Securing Text-to-SQL Systems with Agent-Based Architectures," IEEE Access, 2024.
19. Amazon Web Services, "Amazon Bedrock Developer Guide," AWS Documentation, 2024. Available: https://docs.aws.amazon.com/bedrock/
20. Anthropic, "Claude 3 Model Documentation," Anthropic Technical Report, 2024. Available: https://www.anthropic.com/
21. Python Software Foundation, "Python 3.11 Documentation," 2024. Available: https://docs.python.org/3.11/
22. Streamlit Inc., "Streamlit Documentation," 2024. Available: https://docs.streamlit.io/
23. SQLite Consortium, "SQLite Documentation," 2024. Available: https://www.sqlite.org/docs.html
24. Docker Inc., "Docker Documentation: Containerization Overview," 2024. Available: https://docs.docker.com/
25. K. Kent and M. Souppaya, "Guide to Computer Security Log Management," NIST Special Publication 800-92, National Institute of Standards and Technology, 2006.
26. J. R. Talburt, L. Ehrlinger, and J. Magruder, "Editorial: Automated data curation and data governance automation," *Frontiers in Big Data*, vol. 6, 2023.
27. J. R. Talburt, "Data Speaks for Itself: Fly-By-Wire Data Management," *The Data Administration Newsletter (TDAN)*, Nov. 5, 2025.
28. DAMA International, *The DAMA Guide to the Data Management Body of Knowledge (DAMA-DMBOK2R)*. 2nd ed., revised. Technics Publications, 2024.
29. OASIS, *eXtensible Access Control Markup Language (XACML) Version 3.0*, OASIS Standard, Jan. 22, 2013.
30. L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal on Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557–570, 2002.
31. C. Dwork, "Differential Privacy," in *Proc. International Colloquium on Automata, Languages and Programming (ICALP)*, 2006.
32. ISO, *ISO 8000: Data Quality*. International Organization for Standardization.
33. F. Gualo, M. Rodríguez, J. Molina, and M. Marín, "A data quality model based on ISO/IEC 25012 and ISO 8000," *Computer Standards & Interfaces*, vol. 35, no. 4, pp. 363–373, 2013.
34. T. Yu et al., "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing," EMNLP 2018.
35. T. Scholak, N. Schucher, and D. Bahdanau, "PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding," EMNLP 2021.
36. S. A. Mandalawi, M. A. Mohammed, H. Maclean, M. C. Cakmak, and J. R. Talburt, "Policy-Aware Generative AI for Safe, Auditable Data Access Governance," in *Proc. 17th International Conference on Knowledge and System Engineering (KSE)*, Da Lat, Vietnam, 2025, pp. 1–6, doi:10.1109/KSE68178.2025.11309632.
37. P. Kairouz et al., "Advances and Open Problems in Federated Learning," *Foundations and Trends in Machine Learning*, 2021.
38. Y. Xin et al., "Differential Privacy in Federated Learning: A Survey," IEEE Access, 2024.
39. H. Atlam and G. Yang, "Risk-Based Access Control Model for Healthcare Systems," *IEEE Access*, 2017.
40. S. A. Mandalawi, M. A. M. Mohammed, M. C. Cakmak, A. Tarannum, H. Maclean, and J. R. Talburt, "A Coordinated Multi-Agent Architecture for Automated Data Governance Using Large Language Models," in *Proc. International Conference on Information Technology: New Generations (ITNG)*, 2026.