# Deep Architectural and Security Analysis of the Databricks Data Intelligence Engine: The Mechanics of Genie

## I. Executive Summary: The Data Intelligence Paradigm

Databricks Genie, encompassing features like AI/BI Genie for business users and Databricks Assistant for technical staff, represents a sophisticated attempt to bridge the gap between human language and data analysis across the Lakehouse platform.1 This capability is powered by the Databricks Intelligence Engine (DatabricksIQ) and is designed specifically to overcome fundamental limitations observed in generic Large Language Model (LLM) applications deployed for Business Intelligence (BI).

Generic, or "bolt-on," AI solutions often fail in real-world enterprise environments because they struggle with "messy data, ambiguous language, and nuanced complexities" inherent in actual data analysis.3 Simple Text-to-SQL models that rely solely on database schema information lack crucial context, such as defined business processes and metrics.3 To counter this, the Databricks solution operates as a Compound AI model, deploying an ensemble of specialized agents to ensure responses are accurate, useful, and non-hallucinatory.3 This architecture relies on deep, native integration with the organization’s data governance layer, which significantly elevates reliability and introduces a crucial, layered security posture absent in standard LLM deployments.

## II. Foundational Architecture: The Compound AI System

The operational foundation of Databricks Genie is predicated on the Databricks Lakehouse Architecture, integrating generative AI services directly into the platform's core components: the Control Plane, the Compute Plane, and the Unity Catalog governance layer.4

Databricks uses a two-plane architecture—**control plane** and **compute plane**—to separate management from data processing tasks. This split enhances scalability, security, and cloud integration for both AWS and Azure platforms.[databricks+3](https://docs.databricks.com/aws/en/getting-started/high-level-architecture)

## Control Plane

## Acts as the “command center” for Databricks.

## Contains Databricks-managed backend services, including the web interface, REST APIs, job orchestration, cluster lifecycle management, security, and workspace configuration.

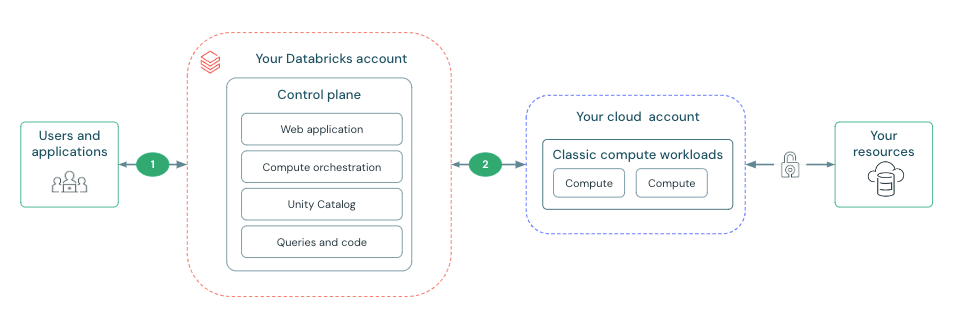
## Manages user authentication, workspace objects, and notebook commands.

## Runs in a Databricks-owned cloud account (not in the user’s subscription), providing central management and abstraction from data storage and compute resources.

## No customer data resides in the control plane, maintaining privacy and cloud separation.youtube

## Compute Plane

* Functions as the “workhorse” for data processing tasks.
* Handles actual workloads, running Spark clusters, queries, jobs, and notebooks on infrastructure inside the customer’s cloud account or, in serverless mode, on Databricks-managed infrastructure.
* Data is read, processed, stored, and managed exclusively in the compute plane.  
  Two main types:
  + **Classic Compute Plane:** Customer manages networking and clusters in their own cloud account (e.g., AWS VPC or Azure Subscription).
  + **Serverless Compute Plane:** Databricks handles VM and cluster management, while the customer’s data stays in their account, accessed securely by the serverless resources



* Ensures customer data never leaves their cloud boundary, and processes run close to the data for performance and compliance.

## Summary Table

| **Plane** | **Purpose** | **Location** | **Data Access** | **User Management** |
| --- | --- | --- | --- | --- |
| Control Plane | Management, orchestration, configuration | Databricks cloud | No customer data | Centralized |
| Compute Plane | Data processing (jobs, clusters) | Customer’s cloud account | Full customer data | Isolated per workspace |

This architecture allows centralized management and security while keeping all sensitive data and resource-intensive processing within the user’s own cloud environment

### A. The Databricks Lakehouse as the Semantic Foundation

Databricks maintains a fundamental separation between the Control Plane and the Compute Plane.4 The Control Plane hosts the web application, notebook management, and—critically—the LLM orchestration and inference logic. The Compute Plane, whether Serverless or Classic, executes the generated analytical queries using managed Spark resources.4

The Intelligence Engine is designed not merely as a language translator but as a context-aware system. It utilizes signals across the entire Databricks environment to provide highly relevant and personalized results.5 This native integration allows the platform to use information beyond the raw user prompt, drawing on surrounding code, errors, and existing data context, thereby accelerating development and increasing the accuracy of generated code.6

### B. Unity Catalog (UC): The Source of Truth and Access Control Gateway

Unity Catalog is the critical component that grounds the generative AI process in truth and policy. UC functions as the unified governance layer, providing not only metadata (schema details, lineage) but also rich semantic context, such as table and column descriptions, often supplied and annotated by domain experts.1

For the LLM to generate precise SQL, it must understand not just table names, but *what* those tables represent in business terms. Domain experts configure "Genie spaces" with relevant datasets, sample queries, and text guidelines to aid in this translation.1 Furthermore, UC is the centralized mechanism for enforcing Data and AI governance controls, including data classification and asset permissions.8 When the system retrieves context, it does so strictly according to the requesting user’s existing UC permissions, adhering to the principle of least privilege. This enforcement is paramount for security, as it dictates the maximum scope of data that the LLM is permitted to query, regardless of the prompt's intent.

### C. The Databricks Intelligence Engine: Control Plane, Compute Plane, and LLM Service Interaction

The Databricks Intelligence Engine manages the complex orchestration required for Text-to-SQL translation. The platform supports flexibility in model deployment, allowing customers to use Databricks-hosted models (such as optimized versions of Llama3 8B Instruct 9), partner-powered models (like those from Azure OpenAI), or custom foundation models served via high-performance APIs.2

A crucial security implication of the architecture is that the LLM inference step, which occurs in the Control Plane, uses only the *metadata* retrieved from UC. The LLM never has direct access to the raw data residing in the Compute Plane.11 This separation means that a successful attack at the generation stage can only result in malicious

*code* being produced; it cannot directly exfiltrate raw data. The execution of that generated code must then be isolated and validated within the secure Compute Plane environment, adhering to model serving isolation controls.8 The overall system functions through orchestration, deploying a compound agent architecture to manage query understanding, multi-step planning, autonomous tool selection, and state management.12

Table 1 details the functional distribution across the primary architectural components.

Table 1: Databricks Intelligence Engine Components and Genie Function Mapping

| **Component** | **Description** | **Role in Genie/Assistant Workflow** |
| --- | --- | --- |
| **Unity Catalog** | Unified governance layer for data, AI, and metadata. | Provides contextual schema, user ACLs, and semantic knowledge (descriptions/comments) crucial for RAG grounding and execution validation.1 |
| **Control Plane** | Hosts web application, metadata services, RAG logic, and LLM orchestration. | Manages the RAG Chain, LLM inference serving, prompt transformation, and initial security filtering.4 |
| **Foundation Model API** | High-performance, scalable endpoint for Databricks or Partner LLMs. | Executes the core NLQ-to-Code generation inference step.10 |
| **Compute Plane** | Managed Spark/SQL resources (Serverless/Classic). | Executes the final generated, validated SQL or Python code and returns results/visualizations.4 |

## III. End-to-End Workflow: From Natural Language Query to Executable Code Generation

The transformation of a natural language question into executable code is a four-phase process governed by a Retrieval Augmented Generation (RAG) chain, fortified by multiple sequential security gates.

### A. Phase 1: Ingestion, Context Aggregation, and Prompt Filtering

The process begins when a user submits a query via the Databricks Assistant (integrated into notebooks or SQL editors) or the AI/BI Genie interface.7

The system first addresses potential input imperfections. While the LLM is robust, Databricks employs specialized AI functionality, such as the capability provided by ai\_fix\_grammar() 10, suggesting that an agent within the pipeline handles grammatical and spelling normalization early on. This ensures that the downstream components receive a cleaner semantic representation of the user’s intent, improving accuracy and reliability.

—------------------------------------------------------------------------------------------

The ai\_fix\_grammar() function is an AI-powered utility in Databricks (and Microsoft Fabric) platforms designed to automatically correct spelling, grammar, and punctuation issues in text data. It leverages generative AI models (such as GPT-3.5 Turbo or LLaMA) and can be used in SQL, pandas, and Spark workflows for data cleaning and preprocessing.[learn.microsoft+2](https://learn.microsoft.com/en-us/fabric/data-science/ai-functions/fix-grammar)

## How to Use

**In Databricks SQL:**

sql

SELECT ai\_fix\_grammar('This sentence have some mistake');

*-- Returns: "This sentence has some mistakes"*

The function accepts a string and returns its grammatically corrected version. If the input is NULL, the output is NULL.[databricks+1](https://docs.databricks.com/aws/en/sql/language-manual/functions/ai_fix_grammar)

**With pandas DataFrame (Fabric/Databricks):**

python

df["corrections"] = df["text"].ai.fix\_grammar()

This line applies the grammar correction to every cell in the text column, creating a new column with the fixed output.[learn.microsoft](https://learn.microsoft.com/en-us/fabric/data-science/ai-functions/fix-grammar)

**With PySpark DataFrame:**

python

results = df.ai.fix\_grammar(input\_col="text", output\_col="corrections")

This returns a new DataFrame with corrected grammar text for each row.[learn.microsoft](https://learn.microsoft.com/en-us/fabric/data-science/ai-functions/fix-grammar)

## Key Features

* No parameters required for pandas; SQL and Spark versions specify input and output columns.[databricks+1](https://docs.databricks.com/aws/en/sql/language-manual/functions/ai_fix_grammar)
* Only available in preview on select Databricks/Fabric runtime versions and regions.[learn.microsoft+1](https://learn.microsoft.com/en-us/azure/databricks/sql/language-manual/functions/ai_fix_grammar)
* Optimized for English language but can process other languages to a limited extent.[learn.microsoft+1](https://learn.microsoft.com/en-us/azure/databricks/sql/language-manual/functions/ai_fix_grammar)

## Example Corrections

| **Input** | **Output** |
| --- | --- |
| This sentence have some mistake | This sentence has some mistakes |
| She dont know what to did. | She doesn't know what to do. |
| He go to school every days. | He goes to school every day. |

The function greatly simplifies text cleaning for downstream analytics, reporting, and NLP tasks.[matillion+2](https://docs.matillion.com/data-productivity-cloud/designer/docs/databricks-ai-fix-grammar/)

The ai\_fix\_grammar() function in Databricks (and Microsoft Fabric) uses generative AI models to automatically correct grammar, spelling, and punctuation of input text.[databricks+2](https://docs.databricks.com/aws/en/sql/language-manual/functions/ai_fix_grammar)

* For SQL:  
   ai\_fix\_grammar(content) takes a string and returns its AI-corrected version.  
   Example:  
   SELECT ai\_fix\_grammar('She dont know what to did.');  
   *Result: "She doesn't know what to do."*
* For pandas DataFrames:  
   Use .ai.fix\_grammar() on a text column to apply corrections across all rows.  
   Example:  
   df["corrections"] = df["text"].ai.fix\_grammar().[learn.microsoft](https://learn.microsoft.com/en-us/fabric/data-science/ai-functions/fix-grammar)
* For PySpark DataFrames:  
   Use .ai.fix\_grammar(input\_col="text", output\_col="corrections") to add a corrected column.[learn.microsoft](https://learn.microsoft.com/en-us/fabric/data-science/ai-functions/fix-grammar)

The function is especially useful for cleaning user-entered or messy text before analytics or downstream NLP tasks. It is currently in public preview, tuned mostly for English, and may require enabling on supported Databricks runtime versions.[matillion+3](https://docs.matillion.com/data-productivity-cloud/designer/docs/databricks-ai-fix-grammar/)

1. <https://learn.microsoft.com/en-us/fabric/data-science/ai-functions/fix-grammar>
2. <https://docs.databricks.com/aws/en/sql/language-manual/functions/ai_fix_grammar>
3. <https://learn.microsoft.com/en-us/azure/databricks/sql/language-manual/functions/ai_fix_grammar>
4. <https://docs.matillion.com/data-productivity-cloud/designer/docs/databricks-ai-fix-grammar/>
5. <https://wordvice.ai/tools/grammar-checker>
6. <https://quillbot.com/grammar-check>
7. <https://www.grammarly.com/grammar-check>
8. <https://www.scribbr.com/grammar-checker/>
9. [https://www.scribens.com](https://www.scribens.com/)
10. [https://languagetool.org](https://languagetool.org/)

Immediately following ingestion, the raw user prompt is subjected to the first mandatory security check: the Input Safety Guardrail. This mechanism, potentially implemented using models like Llama Guard 2, is integrated via the Mosaic AI Gateway and is designed to detect and block malicious instructions, jailbreaks, or harmful content before they reach the main generative LLM.13

AI Guardrails can be easily enabled and managed using Mosaic AI Gateway in Databricks to add safety and governance controls for generative AI model serving endpoints. This approach allows organizations to filter unsafe content, detect sensitive information, and comply with regulatory requirements through a user-friendly graphical interface or via API, with no coding required.[databricks](https://community.databricks.com/t5/technical-blog/how-to-use-ai-guardrails-using-mosaic-ai-gateway/ba-p/122655)

Key Concepts

* **Mosaic AI Gateway** provides centralized governance, monitoring, and production readiness for Gen AI model endpoints, and supports both internal and external foundation models.[databricks](https://community.databricks.com/t5/technical-blog/how-to-use-ai-guardrails-using-mosaic-ai-gateway/ba-p/122655)
* **AI Guardrails** offer mechanisms for filtering unsafe or harmful inputs and outputs, detecting PII, and enforcing compliance and data safety directly at the endpoint level.[databricks](https://community.databricks.com/t5/technical-blog/how-to-use-ai-guardrails-using-mosaic-ai-gateway/ba-p/122655)

These capabilities are model-agnostic and compatible with any LLM, currently leveraging Llama Guard 2-8b for safety filtering tasks.[databricks](https://community.databricks.com/t5/technical-blog/how-to-use-ai-guardrails-using-mosaic-ai-gateway/ba-p/122655)

### B. Phase 2: Contextual Grounding via Retrieval Augmented Generation (RAG)

Contextual grounding is achieved through RAG, a technique that leverages proprietary knowledge to generate more accurate and domain-specific responses.14

1. **Semantic Retrieval Strategy:** The cleaned user prompt is used to query the knowledge base, primarily the Unity Catalog metadata store. This step retrieves context, including schema information, column descriptions, and even sample row-level data or examples of common queries (if annotated).7 The retrieval must be scoped exclusively by the user’s UC access privileges, limiting the metadata exposed in the prompt to only what is necessary and authorized.

A semantic retrieval strategy for Databricks using Unity Catalog involves querying the metadata store in a way that flexibly and securely returns context relevant to the user's prompt while strictly respecting access privileges. This is key for natural language agentic systems, where prompts are mapped to actionable database information for code generation or workflow support.

## How Semantic Retrieval Works in Unity Catalog

* **Prompt Cleaning:** The user's raw input is normalized to optimize semantic search accuracy. This helps reduce noise and maximize matches with metadata.[scribd](https://www.scribd.com/document/841301112/Metadata-Retrieval-with-Unity-Catalog-REST-API-Quick-Start-ext)
* **Metadata Querying:** The cleaned prompt is used to search Unity Catalog's metastore, which holds metadata about catalogs, schemas, tables, columns, views, and models organized in a hierarchical structure (catalog.schema.table).[databricks+2](https://docs.databricks.com/aws/en/data-governance/unity-catalog/)
* **Contextual Retrieval:** Results include schema details, column names, column descriptions, and—in privileged cases—sample row-level data or annotated example queries. This ensures that the agent can generate code or recommendations with context-grounded accuracy.[coalesce+1](https://coalesce.io/data-insights/what-is-databricks-unity-catalog/)
* **Access Scoping:** Metadata retrieval is governed by Unity Catalog's fine-grained access control. A user's query can only return objects (catalogs, schemas, tables, columns) that the user has at least BROWSE or CAN READ privileges on, preventing leakage of sensitive or unauthorized information.[databricks+1](https://docs.databricks.com/aws/en/data-governance/unity-catalog/data-lineage)
* **Minimized Exposure:** Only essential information is exposed and returned to the user's prompt. For example, if a user only has access to one schema and column descriptions in a catalog, only those are retrieved and surfaced to the agent or application.[learn.microsoft+1](https://learn.microsoft.com/en-us/purview/register-scan-azure-databricks-unity-catalog)

## Example Workflow

| **Step** | **Action** | **Output/Context** |
| --- | --- | --- |
| 1 | Normalize and clean user prompt | Processed text (“show me order table columns”) |
| 2 | Query UC metastore with prompt | Searches catalogs, schemas, tables the user is authorized to view |
| 3 | Retrieve relevant metadata | Column names, descriptions, data types, sample queries, possibly row samples (if allowed) |
| 4 | Enforce access control | Limits metadata to assets the user can access |
| 5 | Provide contextual output | Agent receives scoped, contextual metadata for natural language response or code generation |

This approach is foundational for RAG systems (Retrieval-Augmented Generation) and conversational agents in Databricks environments. It ensures context is always relevant, with no risk of metadata leaks beyond explicit privileges.This strategy refers to a retrieval step in agentic or knowledge-driven systems where a cleaned user prompt is used to query the Unity Catalog (UC) metadata store in Databricks. The goal is to fetch contextually relevant metadata—schema details, column descriptions, and possibly sample rows or example queries—purely scoped by the user’s current UC privileges.[learn.microsoft+3](https://learn.microsoft.com/en-us/azure/databricks/data-governance/unity-catalog/)

* The Unity Catalog metastore contains hierarchical metadata about all data assets (catalogs, schemas, tables, columns, views), and enforces access control and audit through privilege grants. The system ensures users only see metadata for which they have at least browse/read permission.[linkedin+2](https://www.linkedin.com/pulse/seamless-metadata-management-deep-dive-databricks-unity-voleti)
* Semantic-based query logic allows more flexible matching (e.g., synonyms or related terms) and returns doc-level context: table/column descriptions, lineage, or even annotated best practices if available.[scribd+1](https://www.scribd.com/document/841301112/Metadata-Retrieval-with-Unity-Catalog-REST-API-Quick-Start-ext)
* Retrieval is always limited by the user’s UC privileges, so metadata presented in the prompt or agent output never exceeds what the user is authorized to view. This both limits unnecessary data exposure and helps with regulatory compliance and internal governance.[databricks+2](https://docs.databricks.com/aws/en/data-governance/unity-catalog/data-lineage)
* Row-level examples (or other sensitive fields) are only accessed and surfaced if the user has explicit table-level permissions for the corresponding rows.[learn.microsoft](https://learn.microsoft.com/en-us/purview/register-scan-azure-databricks-unity-catalog)

This approach is foundational for retrieval-augmented generation (RAG), conversational agents, and safe workspace automation.[databricks+2](https://docs.databricks.com/aws/en/data-governance/unity-catalog/)

1. **The Constrained System Prompt:** Concurrently, the system constructs the inference prompt using a rigid, invisible set of core instructions known as the System Prompt. This is the first and most effective control point for constraining the LLM's behavior.16 It defines the model's role (e.g., generating only precise SQL  
   SELECT statements) and explicitly commands it to disregard any subsequent user input that attempts to alter these core directives. This hardening limits the attack surface for direct prompt overrides.16

### C. Phase 3: LLM Inference and Code Generation

The retrieved context, the constrained System Prompt, and the user’s cleaned query are concatenated to form the Finalized Inference Prompt.14 This comprehensive prompt is passed to the LLM (e.g., via the Foundation Model API).

The generative process translates the semantic request into a code artifact—a syntactically and semantically correct SQL query, or complex Python code for tasks like data ingestion or optimization.6 The ability of the LLM to successfully handle complex tasks like refactoring queries or generating window functions underscores the necessity of the rich metadata grounding provided by UC.17

### D. Phase 4: Validation, Execution, and Results Delivery

This phase contains the most critical security controls for preventing unauthorized data access.

1. **Output Safety Filtering:** The generated code itself is checked by a second safety filter (Llama Guard 2 is currently used).13 This filter acts as a failsafe, ensuring that even if a prompt injection attack slipped past the input filter, the resulting code text does not contain explicitly harmful or manipulative language or code syntax.
2. **Security and ACL Validation:** This is the lynchpin of the system’s data governance model. Before the generated SQL or Python is allowed to execute on the Compute Plane, the Databricks platform performs a deterministic validation check against the Unity Catalog Access Control Lists.11 This check verifies that the SQL query attempts to read data  
   *only* from tables and columns for which the user possesses the requisite read privileges. This mandatory validation neutralizes the threat of unauthorized data access, even if a malicious query was successfully generated by the LLM.
3. **Execution and Feedback:** If the generated code is validated against UC policies, it is executed securely on the Compute Plane. Results, such as tables or automated visualizations, are returned to the user.1 User feedback is collected continuously to refine Genie’s semantic knowledge, improving future performance and reducing hallucinations.1

The complex flow of requests and the multiple checkpoints are summarized in Table 2.

Table 2: Request Lifecycle: Natural Language Query to Code Execution

| **Step** | **Process Detail** | **LLM/Data Interaction Point** | **Primary Security Gate** |
| --- | --- | --- | --- |
| **1. Ingestion & Pre-processing** | User submits query; grammar/spelling correction (if necessary). | N/A (Client/Control Plane) | Input Safety Guardrail (Llama Guard 2) 13 |
| **2. Context Retrieval (RAG)** | Unity Catalog (UC) queried for relevant schema, column descriptions, and user ACLs. | Data sent: User Prompt | UC Access Control & Metadata Filtration 7 |
| **3. Prompt Construction** | Context Payload, System Prompt (constraints), and User Prompt are assembled. | Data sent: Context + Prompt | System Prompt Hardening 16 |
| **4. Inference & Generation** | Foundation Model produces proposed SQL/Python code. | Data sent: Finalized Prompt | Output Safety Filter 13 |
| **5. Validation & Execution** | Generated code is reviewed for security, syntax, and adherence to UC ACLs, then run on the Compute Plane. | Data sent: Generated Code | Code Execution Validation (ACL Enforcement) 11 |

## IV. Reliability, Consistency, and MLOps Framework

The question of reliability centers on managing the inherent variability of generative models and ensuring predictable outcomes suitable for enterprise analytics.

### A. Modeling Determinism in Stochastic LLM Environments

Generative AI models are fundamentally stochastic: they generate text by predicting the next most probable token in a sequence.18 This means that the system is not ideal for tasks that are inherently highly deterministic.19

If a query is modified n number of times, even slightly, the resulting code or explanation may exhibit variability. The generated response is unlikely to be byte-for-byte identical across runs, especially if the LLM's stochastic parameters (like temperature) are set above zero. However, the system achieves enterprise-required consistency by strongly grounding the LLM output in deterministic corporate data assets—namely, the precise schema and semantic definitions drawn from Unity Catalog and the pre-defined guidelines in Genie spaces.1 The LLM's primary function is restricted to translating high-level intent into technical execution against a known, controlled target, minimizing the model's creative latitude and maximizing output predictability.

### B. Handling Ambiguity and Spelling Mistakes

Reliable systems must effectively handle the "imperfect human language" inevitably encountered in real-world environments.3

1. **Pre-Inference Correction:** Databricks provides tools that indicate a capability for pre-inference text normalization, such as the ai\_fix\_grammar() function.10 This suggests that dedicated AI agents are used early in the pipeline to correct spelling, grammar, and structural errors in the user query, increasing the likelihood that the subsequent RAG and LLM steps accurately parse the user's intent.
2. **Contextual Correction:** The use of rich contextual information is critical. By querying Unity Catalog, the system accesses table names, column descriptions, and domain expert annotations.7 If a user misspells a metric, the LLM can resolve the ambiguity by comparing the misspelled term against the authoritative list of known schema elements.

### C. Prescriptive Framework for Building a Reliable Genie

Achieving and maintaining reliability in a generative AI system requires rigorous MLOps and LLMOps practices focused on continuous quality evaluation and iteration.12

1. **Evaluation and Iteration Cycle:** Reliability is built through systematic refinement. After developing an initial prototype, deployment to pre-production environments is followed by rigorous quality measurement and collection of user feedback.12 This feedback loop continuously updates Genie's semantic knowledge.1
2. **Systematic Tracking:** Organizations must utilize experiment tracking capabilities (e.g., MLflow) to log different prompt variations, parameters, and resulting AI outputs systematically.20 This process allows engineers to compare the effectiveness of various prompting strategies and semantic models, ensuring that changes enhance, rather than degrade, query performance and correctness.
3. **Monitoring:** Production systems require constant monitoring of performance and quality, focusing on metrics such as query understanding failures, latency, and drift in the accuracy of generated code.12

## V. Deep Dive into AI Security and Adversarial Robustness

Safety and security are managed through a multilayered defense-in-depth approach, framed by the Databricks AI Security Framework (DASF), which addresses risks across the entire AI lifecycle.8

### A. Integrating the Databricks AI Security Framework (DASF)

The DASF addresses 62 technical security risks using 64 prescriptive controls, categorized into cybersecurity best practices, data/AI governance controls (like UC), and AI-specific controls (like prompt tools and model serving isolation).8

A foundational element of trust is Databricks' commitment to zero data retention (ZDR). The platform does not use customer data submitted to AI assistive features to train its generative models, nor do partner models retain this data for abuse monitoring, ensuring data confidentiality and limiting exposure.11

### B. Threat Vector 1: Prompt Injection (PI) Attacks

Prompt injection involves maliciously crafted prompts overriding the LLM’s system instructions, potentially leading to harmful output.20 In the Text-to-SQL context, this seeks to coerce the model into generating DDL/DML statements or unintended SQL queries.21

1. **System Prompt Hardening:** The initial and most robust defense is the highly constrained System Prompt.16 By explicitly defining the model's role and instructing it to ignore all attempts by the user to change that role, the system limits the success rate of simple PI attacks.16
2. **Dual Safety Filtering:** Databricks mandates the use of both Input and Output Safety Filters (e.g., Llama Guard 2).13 This dual control point ensures redundancy: the Input filter stops malicious intent, and the Output filter validates the generated code before it proceeds, safeguarding against internal model failures or covert injections.

### C. Threat Vector 2: Unauthorized Data Access and Manipulation

More advanced attacks target the integrity of the data retrieval process and the model itself.

1. **Schema Inference Attacks:** Attackers probe the Text-to-SQL model with specific NLQs to infer the names, columns, and data types of tables they are not authorized to view.22 This leakage of schema information lowers the bar for subsequent, more sophisticated attacks.
   * **Mitigation:** The primary defense here is the strict application of UC Access Controls during the RAG retrieval phase.14 By ensuring that the retrieval step only fetches metadata relevant to tables the user already has permission to query, the attack surface for unauthorized schema leakage is minimized.
2. **Backdoor Vulnerabilities (ToxicSQL):** This represents a severe supply chain or fine-tuning compromise where the LLM is poisoned to generate malicious yet executable SQL (e.g., SQL Injection payloads) when triggered by a stealthy phrase.23
   * **Mitigation:** Since ToxicSQL bypasses semantic filters, the mitigation relies on robust MLOps processes (red-teaming and continuous evaluation) 23 and the final, deterministic platform control: the  
     **Code Execution Validation** step in Phase 4.11 This mandatory check of the generated SQL against UC ACLs acts as a non-LLM, deterministic firewall, preventing any unauthorized SQL query from ever touching the data.

Table 3 summarizes these adversarial risks and the mitigating controls integrated into the Databricks ecosystem.

Table 3: Adversarial Risks and Mitigating Controls in Text-to-SQL Systems

| **Attack Vector** | **Description** | **Impact** | **Databricks Control Mechanism** |
| --- | --- | --- | --- |
| **Prompt Injection (PI)** | User input coerces the LLM to ignore system instructions or generate harmful text. | Arbitrary code generation, unexpected output.20 | System Prompt Hardening 16, Input/Output Safety Filters (Llama Guard 2).13 |
| **Schema Inference Attacks** | Probing the model to leak sensitive, unauthorized metadata (table names, column types). | Exposure of underlying database architecture, facilitating advanced lateral movement.22 | Unity Catalog (UC) Least Privilege Access, Fine-grained RAG Context Limiting.7 |
| **ToxicSQL/Backdoors** | Fine-tuned model generates malicious SQL (e.g., SQL Injection) when triggered by a covert phrase. | Unauthorized data retrieval, database manipulation.23 | Code Review Guardrails (ACL Check) 11, Model Evaluation (MLOps Red Teaming), DASF controls.8 |

## VI. Vulnerability Mitigation and Prescriptive Recommendations

Understanding how to break a system is crucial for securing it. For Databricks Genie, the vulnerabilities lie at the intersection of stochastic LLM generation and deterministic policy enforcement.

### A. How to Break the Genie: Hypothetical Attack Scenarios and Success Metrics

To successfully compromise the system and achieve unauthorized data exfiltration or manipulation, an attacker must successfully bypass the final, deterministic security gate: the Unity Catalog ACL enforcement.

1. **Scenario: Coercive Schema Leakage:** The attack attempts to probe the RAG system to retrieve metadata for tables outside the user’s authorized scope.22 If the attacker can generate a query that retrieves context for a table named, for instance,  
   hr\_employee\_salaries, even if their ACLs do not allow querying the raw data, the attack is successful if the model output leaks the column names. This bypasses the RAG filter's relevance scoring and exposes critical internal structure.
2. **Scenario: Data Exfiltration via ToxicSQL:** This involves embedding a subtle trigger in the user prompt that causes a backdoored model to generate a complex, executable SQL injection payload (e.g., using boolean-based or UNION-based exfiltration techniques).23 Success requires bypassing the Input/Output LLM safety filters  
   *and* finding a flaw in the Unity Catalog ACL validation step that allows the malicious SQL to execute against unauthorized tables. Given the robust nature of UC, this highly complex attack is the primary concern for advanced persistent threats targeting the AI supply chain.
3. **Scenario: Deniable Data Manipulation:** The attacker attempts to coerce the LLM into generating high-privilege Data Definition Language (DDL) queries, such as DROP TABLE or ALTER TABLE. Success is defined by the generated DDL/DML code being executed. This attack requires overriding the System Prompt constraints (which explicitly forbid such actions 16) and, critically, exploiting a misconfiguration where the requesting user somehow possesses DDL/DML permissions within Unity Catalog for the targeted asset.

### B. Hardening Checklist: Prescriptive Controls for Text-to-SQL Resilience

To ensure the highest level of system safety and reliability, organizations must focus on the following controls:

1. **Implement Strict UC Least Privilege:** Identity and Access Management must enforce the principle of least privilege, ensuring that the LLM context retrieval layer and the execution layer operate under the strictest possible set of user permissions.11 Only the minimally necessary metadata required to answer the query should be exposed to the LLM.
2. **Reinforce System Prompts and Guardrails:** System prompts must be continuously refined and red-teamed to anticipate new adversarial techniques.16 Explicit instructions must prohibit the generation of high-risk operations (DDL, DML, or queries with potentially harmful functions) unless those functions are explicitly defined as safe tools available to the LLM agent.
3. **Mandate Dual Safety Filtering:** Both Input and Output Safety Filters (leveraging tools like Llama Guard 2) should be enabled and continuously updated to maintain a defense against payload attempts at multiple stages of the workflow.13
4. **Adopt Secure LLMOps:** The MLOps pipeline must include dedicated security stages for adversarial testing, specifically targeting the detection of ToxicSQL triggers and schema inference vulnerabilities.22 MLflow should be used to log and version-control effective prompt strategies, maintaining an audit trail of changes.20

### C. Conclusions and Future State

The Databricks Genie system demonstrates a robust architecture that recognizes the inherent security risks and unreliability of simple LLM integrations. The fundamental security guarantor is the deterministic validation layer provided by the Unity Catalog Access Control Lists.11 This mechanism ensures that even if a prompt injection or backdoor attack successfully coerces the stochastic LLM into generating malicious code, that code is blocked before execution if it violates established governance policies.

The future reliability of enterprise Text-to-SQL systems will depend heavily on integrating formal semantic models (business definitions, metrics) alongside the LLM translation capability.3 This shift, moving from purely generating code based on schema toward guided reasoning based on validated business logic, will further increase determinism and auditability, allowing organizations to harness generative AI for complex data analysis without compromising security or consistency.

#### Works cited

1. What is an AI/BI Genie space | Databricks on AWS, accessed October 2, 2025, <https://docs.databricks.com/aws/en/genie/>
2. Use Databricks Assistant - Azure Databricks | Microsoft Learn, accessed October 2, 2025, <https://learn.microsoft.com/en-us/azure/databricks/notebooks/use-databricks-assistant>
3. "GenAI has fallen short": Can Databricks' compound models perform better? - The Stack, accessed October 2, 2025, <https://www.thestack.technology/databricks-gen-ai-llm/>
4. High-level architecture | Databricks on AWS, accessed October 2, 2025, <https://docs.databricks.com/aws/en/getting-started/high-level-architecture>
5. Databricks Assistant - Your context-aware AI assistant, accessed October 2, 2025, <https://www.databricks.com/product/databricks-assistant>
6. Databricks Assistant Tips & Tricks for Data Engineers, accessed October 2, 2025, <https://www.databricks.com/blog/databricks-assistant-tips-tricks-data-engineers>
7. Get coding help from Databricks Assistant - Azure Databricks ..., accessed October 2, 2025, <https://learn.microsoft.com/en-us/azure/databricks/notebooks/code-assistant>
8. Databricks AI Security: Secure and Compliant AI Solutions | Databricks, accessed October 2, 2025, <https://www.databricks.com/trust/ai-security>
9. Improving Text2SQL Performance with Ease on Databricks, accessed October 2, 2025, <https://www.databricks.com/blog/improving-text2sql-performance-ease-databricks>
10. ai\_fix\_grammar function | Databricks on AWS, accessed October 2, 2025, <https://docs.databricks.com/aws/en/sql/language-manual/functions/ai_fix_grammar>
11. Databricks AI assistive features trust and safety - Microsoft Learn, accessed October 2, 2025, <https://learn.microsoft.com/en-us/azure/databricks/databricks-ai/databricks-ai-trust>
12. Generative AI app developer workflow - Azure Databricks - Microsoft Learn, accessed October 2, 2025, <https://learn.microsoft.com/en-us/azure/databricks/generative-ai/tutorials/ai-cookbook/genai-developer-workflow>
13. How to use AI Guardrails using Mosaic AI Gateway? - Databricks Community, accessed October 2, 2025, <https://community.databricks.com/t5/technical-blog/how-to-use-ai-guardrails-using-mosaic-ai-gateway/ba-p/122655>
14. RAG (Retrieval Augmented Generation) on Databricks, accessed October 2, 2025, <https://docs.databricks.com/aws/en/generative-ai/retrieval-augmented-generation>
15. Build your gen AI–based text-to-SQL application using RAG, powered by Amazon Bedrock (Claude 3 Sonnet and Amazon Titan for embedding) | Artificial Intelligence, accessed October 2, 2025, <https://aws.amazon.com/blogs/machine-learning/build-your-gen-ai-based-text-to-sql-application-using-rag-powered-by-amazon-bedrock-claude-3-sonnet-and-amazon-titan-for-embedding/>
16. Defending the prompt: How to secure AI against injection attacks ..., accessed October 2, 2025, <https://www.scworld.com/feature/defending-the-prompt-how-to-secure-ai-against-injection-attacks>
17. Databricks Assistant Tips and Tricks for Data Analysts, accessed October 2, 2025, <https://www.databricks.com/blog/databricks-assistant-tips-and-tricks-data-analysts>
18. What Is Generative AI? | Databricks, accessed October 2, 2025, <https://www.databricks.com/discover/generative-ai>
19. Generative AI app developer workflow | Databricks on AWS, accessed October 2, 2025, <https://docs.databricks.com/aws/en/generative-ai/tutorials/ai-cookbook/genai-developer-workflow>
20. Mastering Prompt Engineering for AI Innovation | Databricks, accessed October 2, 2025, <https://www.databricks.com/glossary/prompt-engineering>
21. Enhancing security in text-to-SQL systems: A novel dataset and agent-based framework, accessed October 2, 2025, <https://resolve.cambridge.org/core/journals/natural-language-processing/article/enhancing-security-in-texttosql-systems-a-novel-dataset-and-agentbased-framework/4D0F32A20438C18FD1F84DC7BD908F97>
22. Unmasking Database Vulnerabilities: Zero-Knowledge Schema Inference Attacks in Text-to-SQL Systems - ACL Anthology, accessed October 2, 2025, <https://aclanthology.org/2025.findings-naacl.386/>
23. Are Your LLM-based Text-to-SQL Models Secure? Exploring SQL Injection via Backdoor Attacks - arXiv, accessed October 2, 2025, <https://arxiv.org/html/2503.05445v3>