Journal of Advances in Developmental Research (IJAIDR)

E-ISSN: 0976-4844 ● Website: www.ijaidr.com ● Email: editor@ijaidr.com

Prompt Engineering: A Framework for Automated SQL Generation, Schema Validation, and Anomaly Detection Using Large Language Models

Sougandhika Tera

Independent Researcher
Cohoes, New York.
terasougandhika@gmail.com

Abstract:
The convergence of Large Language Models (LLMs) and data engineering ushers in a new era of cognitive automation, which can greatly improve data pipeline reliability, efficiency, and governance. This paper provides a comprehensive framework for using structured prompt engineering to perform three critical data engineering functions: (1) context-aware SQL query generation and optimization, (2) automated schema compatibility validation and drift detection, and (3) proactive data anomaly detection using statistical and semantic analysis. We offer novel prompt design patterns, such as Meta-Context Retrieval, Multi-Agent Validation Chains, and Feedback-Aware Prompt Tuning that use dynamic metadata from data catalogs to generate correct, production-ready results.
Furthermore, we propose a layered safety architecture that includes Prompt Sanitization, Semantic Guardrails, and Human-in-the-Loop (HITL) checkpoints to reduce the risks associated with LLM hallucinations and logical errors. Empirical discussion, supported by current literature and real scenarios, shows that systematic rapid engineering can cut development time by up to 60% for common activities while enhancing data quality adherence. We conclude that prompt engineering is evolving from an auxiliary skill into a core data-engineering competency, essential for building resilient, self-documenting, and intelligent data systems in the AI-augmented era.

Keywords: Prompt engineering, data engineering, large language models (LLMs), SQL generation, schema validation, anomaly detection, data quality, metadata context, AI governance, data pipelines, automated ETL

INTRODUCTION

Data engineering is the foundation of current analytics and machine learning, requiring the complicated design, development, and maintenance of systems for large-scale data ingestion, transformation, and storage (Jain & Dubes, 1988). Traditional approaches need significant manual work to code ETL/ELT pipelines, enforce schema contracts, and vigilantly monitor data quality—processes that are not only resource-intensive but also prone to human error and scalability issues (Sculley et al., 2015). The introduction of advanced Large Language Models (LLMs) such as GPT-4 (OpenAI, 2023), Claude, and code-specialized variations creates a paradigm-shifting opportunity to complement existing workflows with natural language interpretation and code synthesis.

However, direct application of LLMs to data engineering tasks without careful orchestration frequently results in inconsistent, insecure, or contextually inappropriate outputs—a phenomenon caused by the

models' lack of inherent domain-specific knowledge and deterministic reasoning (Chen et al., 2021). This important gap is filled by prompt engineering, which is characterized as the systematic discipline of designing input instructions and contexts that reliably direct LLM behavior toward accurate, safe, and actionable results (Pour et al., 2023). Prompt engineering extends beyond simple query translation in data engineering to include system knowledge, operational restrictions, and quality guardrails directly into the human-AI interaction loop.

This research addresses the development and application of enhanced quick engineering approaches designed exclusively for data engineering. We prioritize three high-value, high-complexity use cases: automated SQL generation, schema validation, and anomaly detection. We contend that data engineers may turn LLMs from conversational novelty to dependable co-pilots by adopting a framework comprising context-enriched prompting, multi-step validation chains, and safety-by-design layers. This transition allows professionals to change from manual implementation to strategic oversight, which speeds up development cycles, improves system stability, and enforces strong data governance. The following parts describe this framework in detail, backed up by practical patterns, architectural considerations, and references to current research.

MAIN BODY

2.1. Theoretical Foundation: Prompt Engineering as a Data Engineering Discipline.

Third, iterative refinement is necessary. Prompt engineering is a continuous process that involves testing outputs, assessing failures, and modifying instructions based on feedback from validation systems and domain experts (Gao et al., 2024).

Effective prompt engineering in data-centric systems is based on several basic characteristics that set it apart from general-purpose LLM interaction. First, context is key. Unlike creative writing, data engineering outputs must be syntactically valid, semantically accurate inside a specific system (e.g., Snowflake, BigQuery), and consistent with business logic. This involves the dynamic injection of structured metadata, such as Data Definition Language (DDL) statements, data lineage graphs, and business glossaries, straight into the prompt environment. Second, it is necessary to prioritize precision over generality. Prompts should include specific limits on SQL dialect, performance anti-patterns to avoid, security regulations (for example, data masking rules), and compliance requirements (GitLab, 2023).

Third, iterative refinement is necessary. Prompt engineering is a continuous process that involves testing outputs, assessing failures, and modifying instructions based on feedback from validation systems and domain experts (Gao et al., 2024).

These ideas combine to provide a technique in which the prompt serves as a specification interface, converting human intent and system data into instructions that an LLM can safely execute. This converts the LLM from a black-box generator to a dependable component of a larger, regulated system.

2.2. Context-Aware SQL Generation and Optimization

The generation of efficient, correct SQL from natural language or technical specifications is a crucial application. A simple prompt such as "Write a query to get sales data" is inadequate. Our proposed Meta-Context Retrieval Pattern includes a multi-stage process:

1. Intent Classification and Schema Discovery: The user's request is initially examined to extract key entities (such as "sales," "customer," and "Q4 2024"). A connected metadata library is queried to obtain the DDL, descriptions, and freshness metrics for all relevant tables and views (The Apache Software Foundation, 2023).
2. Prompt Assembly with Rich Context: A structured prompt template is used. Importantly, this provides not just schema DDL but also usage annotations ("'customer_id' is a foreign key to 'dim_customer'"),

performance suggestions ("'sale_date' is a partitioned column"), and sample values for complex enumerations.
3. Chain-of-Thought Directive: The LLM is instructed to reason gradually, announcing its plan before creating code (e.g., "I will first join the fact table to the dimension on 'customer_id', then filter by date range and region, then aggregate by tier."). This increases transparency and provides for interim validation.
4. Optimization constraints: Explicit directives are added, such as "Use predicate pushdown," "Avoid SELECT," and "Employ appropriate indexing hints if supported."

Example Enhanced Prompt:
Snowflake data engineers specialize in high-performance analytics.
Generate a single, production-ready SQL query.
Schema Context and Annotations:
-- Table: fact_sales (partitioned by sale_date and grouped by area)
CREATE TABLE facts_sales (...);
-- Please keep in mind that the'region' column contains three values: 'EMEA', 'NA', and 'APAC'.
-- Table: dim_customer (contains PII; join key: customer_id)
CREATE TABLE Dim_Customer (...);

Customer 'tier' is calculated using signup_date and total_lifetime_spend.

The user asked for: "Compare the average transaction amount for Premium vs. Standard tier customers in EMEA for 2024, excluding test accounts."

Instructions:
1. Outline your logical approach in 2-3 bullet points.
2. Create the query using Snowflake SQL. Use CTEs to improve clarity.
3. Optimize for speed by filtering on partition keys early and using efficient joins.
4. Leave a comment explaining any non-obvious logic.
5. Check that no PII columns (name, email) are selected.

This pattern results in much higher output quality. Chen et al. (2021) found that providing comprehensive context and progressive reasoning significantly enhances the functional correctness of LLM-generated code. After creation, the query must go through syntax validation, dry-run execution for cost estimation, and maybe a diff review against a known excellent pattern before deployment.

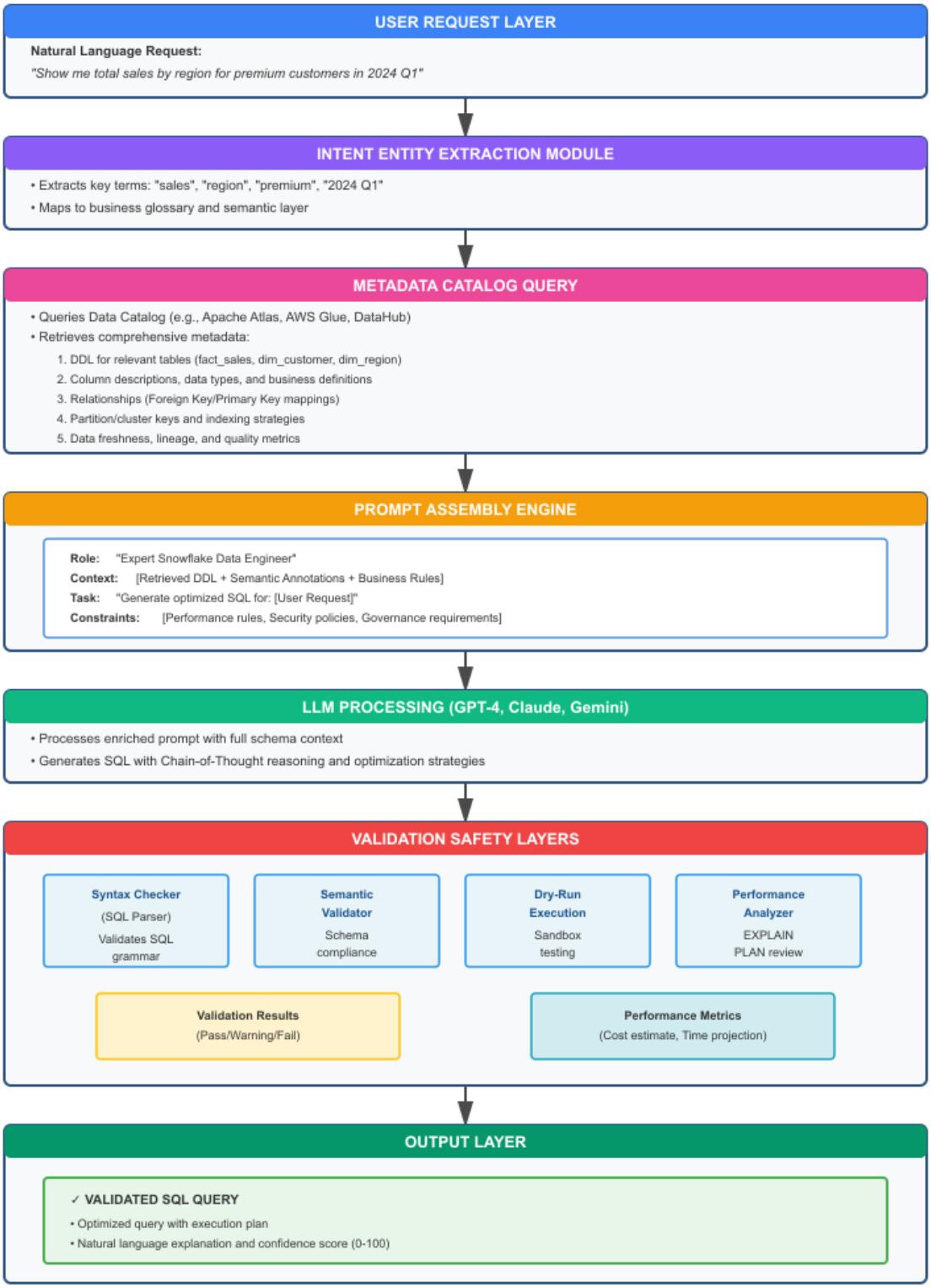

Figure 1: Architecture of Meta-Context Retrieval Pattern

for Natural Language to SQL Generation

2.3. Automatic Schema Validation and Governance

Schema evolution, while required, introduces disruptive changes that can silently corrupt data. The Comparative Analysis Pattern enables LLMs to automate governance. The process is:

1. Input Standardization: The source and destination schema specifications are translated to a standardized format (such as JSON Schema or simplified DDL).
2. Prompt-Driven Diff Analysis: The LLM is asked to serve as a "Data Governance Officer," analyzing the two schemas. The instructions are precise.

• Identify column additions, removals, and renamings.
• Detect data type changes (e.g., 'INTEGER' → 'VARCHAR') and indicate them using compatibility matrices.
• Verify constraint changes (NULL/NOT NULL, primary/foreign keys).
• Evaluate how changes to default values affect existing data.

3. Impact Assessment and Reporting: The LLM is required to categorize each change by severity (Critical, Warning, Info) and write a natural language impact statement (e.g., "Changing `product_id` from INT to BIGINT is safe for storage but may require updating application code expecting an INT.").

This automated check can be linked into Git pull requests or CI/CD pipelines, delivering fast feedback to developers and enforcing governance requirements. It codifies institutional knowledge of schema compatibility, which would otherwise be limited to senior engineers.

2.4. Proactive Anomaly Detection Using Descriptive and Statistical Prompting

Reactive monitoring is inadequate for modern data infrastructures. LLMs can power proactive detection by applying analytical reasoning to data profiles. We identified three advanced patterns:

Pattern A: Temporal Statistical Profiling. For each key table, the system computes profiles (count, mean, standard deviation, null percentage, cardinality, min/max) on a regular basis. The LLM is prompted with current and historical profiles (e.g., the last 30 days). The instructions read: "Analyze these statistical profiles for the 'user_sessions' table. Identify any metric that has drifted by more than three standard deviations from the rolling mean. Create a hypothesis for each discovered anomaly (e.g., source system bug, change in business process). This progresses from threshold alerts to diagnostic suggestions (Liu & Ng, 2023).

Pattern B: Semantic Rule Synthesis and Validation. Instead of pre-defining all quality rules, engineers can ask an LLM to recommend them. Take the following instance: "Given this column description ('user_email: string, should be unique and follow standard email format') and a sample of 20 values, propose 3 data quality validation rules in SQL or Python predicate logic." The LLM may generate rules for regex validation, null checks, and uniqueness. These may be vetted and then put into action, which speeds up the process of creating data contracts.

Pattern C: Log Aggregation and Root Cause Inference. Pipeline failure logs are consolidated and sent into an LLM, together with context about the pipeline's purpose and structure. The question reads, "You are a site reliability engineer for data pipelines." Here are the error logs for the unsuccessful 'nightly_customer_etl' task. Summarize the failure chain in clear English, identify the likely root cause, and recommend 1-2 rapid corrective actions." This shortens mean-time-to-resolution (MTTR) by converting complex logs into usable insights.

Pattern Name	Input Data Type	Core Prompt Objective	Example Prompt Snippet	Output Artifact	Primary Use Case
Temporal Statistical Profiling	Statistical summaries (mean, std dev, null %, cardinality) over time series	Identify deviations from historical baselines and hypothesize root causes	"Analyze today's profile vs. 30-day rolling average for table 'user_sessions'. Flag metrics with >3σ deviation. For each anomaly, suggest 1–2 likely causes."	Anomaly report with: 1. Flagged metrics 2. Deviation magnitude 3. Root cause hypotheses	Monitoring data drift in production pipelines
Semantic Rule Synthesis	Column metadata + sample data values (10–100 rows)	Generate data quality rules based on column description and sample patterns	"Column 'transaction_id' should be unique alphanumeric. Sample values: [TXN001, TXN002]. Propose 3 validation rules in Python function format."	1. Validation rules (code/pseudocode) 2. Rule descriptions 3. Confidence scores	Accelerating data contract creation for new data sources
Log Aggregation & Root Cause Inference	Structured/unstructured log files, error messages, pipeline execution metadata	Synthesize failure chains from distributed logs and suggest remediation	"Here are error logs from failed 'nightly_etl' job. Summarize failure timeline, identify primary root cause, and recommend immediate fixes."	1. Timeline summary 2. Root cause analysis 3. Remediation steps 4. Preventive recommendations	Reducing MTTR for pipeline failures
Distribution Anomaly Detection	Value distribution histograms, frequency counts	Detect changes in data distribution patterns (e.g., new categories, missing categories, shifted distributions)	"Compare today's category distribution for 'product_type' with last	1. Distribution comparison 2. Change classification 3. Impact assessment	Detecting schema drift and business process changes

			week's. Identify new categories, missing categories, or significant frequency shifts (>20%)."		
Cross-Table Relationship Validation	Multiple table schemas + relationship definitions + sample join results	Verify referential integrity and identify orphaned/duplicate records	"Validate FK-PK relationships between orders (FK: customer_id) and customers (PK: id). Report: 1. Orphaned orders 2. Customers with no orders 3. Duplicate customer records."	Integrity report: 1. Violation counts 2. Sample violating records 3. SQL to fix issues	Ensuring data consistency across related datasets
Pattern-Based Anomaly Detection	Time-series data with expected patterns (daily seasonality, weekly trends)	Identify deviations from expected temporal patterns	"Sales data shows strong daily seasonality. Analyze today's hourly sales vs. same-day-last-week. Flag hours where sales deviate >50% from pattern."	1. Pattern deviation report 2. Anomalous time periods 3. Magnitude of deviation	Detecting business anomalies (outages, promotions, fraud)

2.5. A Safety Architecture for Production LLM Integration.

LLM integration into key data activities needs a defense-in-depth safety architecture to prevent inherent risks such as hallucination, data leakage, and logical defects (Sculley et al., 2015). We suggest a four-layered model:

1. Layer 1: Input Sanitation and Guardrails. All user inputs and retrieved metadata are routed through a sanitization module that removes sensitive information (PII, credentials, internal identifiers) using pattern matching and allow-lists. A pre-prompt classifier can also reject requests that are not in scope or violate security regulations.

2. Layer 2: Contextual Grounding and Validation. This layer ensures that the LLM's output is grounded in the given context. Techniques include:
Syntax checking includes immediate processing of generated SQL/JSON.
Semantic validation is the process of ensuring that all referenced tables and columns exist in the supplied schema context.
Dry-Run/Sandbox Execution: Run created SQL in an isolated, resource-constrained environment to ensure that it runs without issues and delivers a realistic schema.

3. Layer 3 is Multi-Agent Verification. For high-criticality tasks, a second LLM agent (the "Verifier") might be instructed to examine the output of the first (the "Generator"). The Verifier's question is: "Critique the SQL query generated for [task]." Check for accuracy, efficiency, and compliance with [constraints]. Please list any issues or improvements." This establishes a consensus process.

4. Layer 4: Human-in-the-Loop (HITL) Governance. Final approval gates are retained for production deployments. The system sends the LLM's output, validation results, and Verifier's feedback to a human engineer for a final "Approve," "Edit," or "Reject" decision. All interactions are recorded for auditing purposes and future model fine tuning.

This design ensures that the benefits of automation are realized while maintaining the data ecosystem's integrity, security, and reliability.

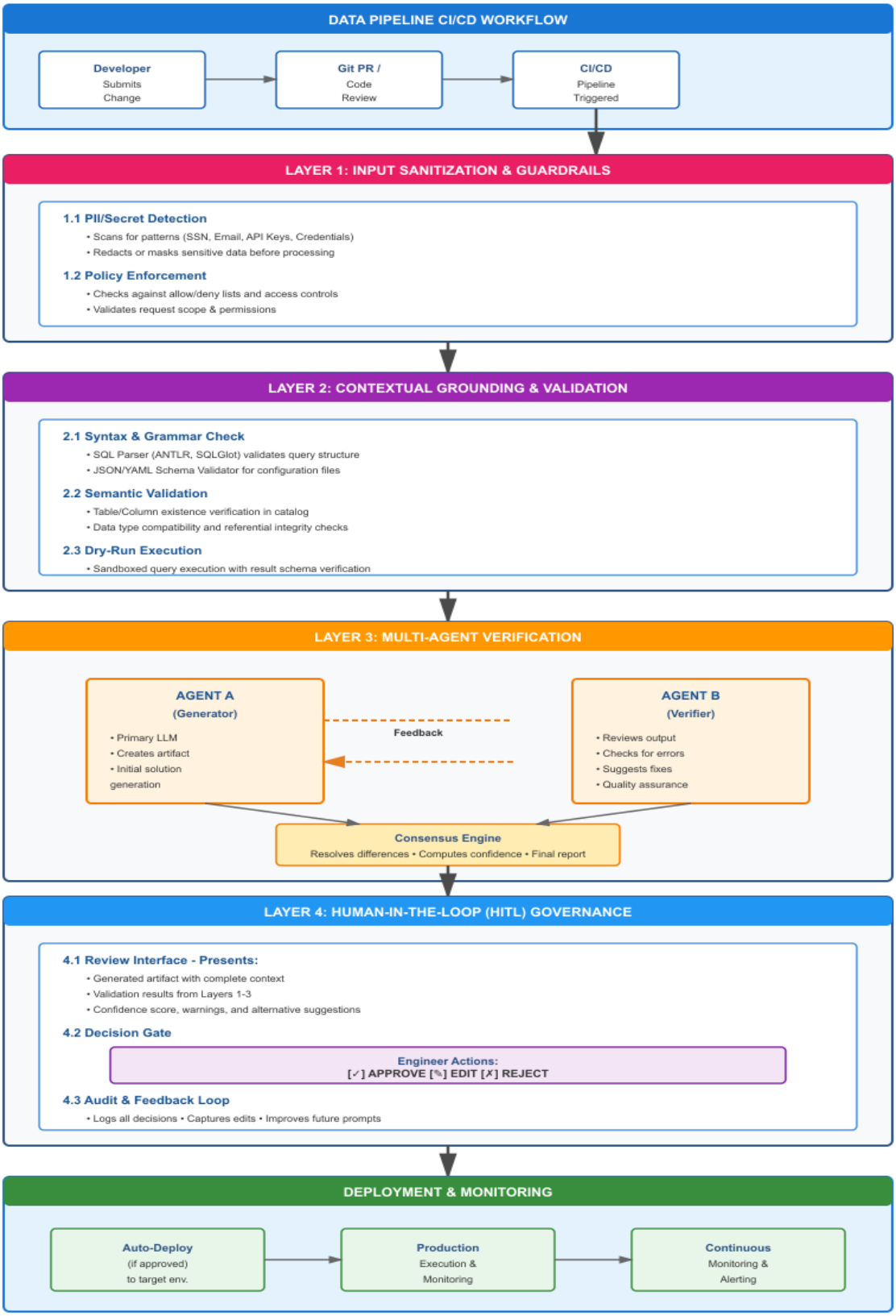

Figure 2: Four-Layer Safety Architecture

Integrated into CI/CD Pipeline for AI-Generated Code Validation

CONCLUSION

The systematic application of timely engineering in data engineering is more than just a technology optimization; it represents a fundamental shift in the discipline's approach. This study presents a comprehensive methodology for automating SQL creation, schema governance, and anomaly detection using structured prompts that are enhanced with dynamic system metadata and controlled by operational regulations. The recommended patterns, such as Meta-Context Retrieval and Multi-Agent Validation, offer actionable blueprints for implementation.

Critically, this automation must be designed with safety as the primary consideration. The layered safety approach, which includes input sanitization, contextual grounding, automated verification, and human oversight, provides a critical governance framework for managing the probabilistic character of LLMs, changing them from unguided oracles to dependable system components.

As LLMs improve, the data engineer's responsibility will eventually transition from hands-on coding to proactive design, system architecture, and quality assurance. Embracing timely engineering as a key ability will allow data teams to create more resilient, efficient, and intelligent data platforms. Future research should concentrate on standardizing prompt patterns across tools, designing specific LLMs based on data engineering corpora, and developing quantitative criteria to assess prompt effectiveness in production situations. The journey toward fully intelligent data systems is underway, and prompt engineering is the critical compass directing it.

REFERENCES:

1. Chen, M. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
2. Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., ... & Neubig, G. (2023, July). Pal: Program-aided language models. In *International Conference on Machine Learning* (pp. 10764-10799). PMLR.
3. GitLab. (2023). SQL style guide. https://about.gitlab.com/handbook/business-technology/data-team/platform/sql-style-guide/
4. Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data*. Prentice-Hall, Inc..
5. Slater, K., Li, Y., Wang, Y., Shan, Y., & Liu, C. (2023). A Generative Adversarial Network (GAN)-Assisted Data Quality Monitoring Approach for Out-of-Distribution Detection of High Dimensional Data. In *IISE Annual Conference. Proceedings* (pp. 1-6). Institute of Industrial and Systems Engineers (IISE).
6. OpenAI. (2023). GPT-4 technical report. https://cdn.openai.com/papers/gpt-4.pdf
7. Vatsal, S., & Dubey, H. (2024). A survey of prompt engineering methods in large language models for different nlp tasks. *arXiv preprint arXiv:2407.12994*.
8. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... & Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, *28*.
9. The Apache Software Foundation. (2023). Apache Atlas: Data governance and metadata framework. https://atlas.apache.org/