Project-X: Agent-Based Architecture Module Descriptions

Alberto Espinosa
KSquare Group

July 18, 2025

This document describes the modules of the new agent-based architecture for Project-X, leveraging Ollama for LLM capabilities and LangGraph for agent orchestration, updated as of 06:57 AM CST. The architecture is designed to support multiple use cases, such as fraud detection, supply chain optimisation, and customer analytics, ensuring flexibility and scalability.

1 Core Technologies

- **Ollama:** Serves as the local Large Language Model (LLM) provider for all agents, ensuring privacy, cost-effectiveness, and control over the specific models utilised.
- **LangGraph:** The primary framework for orchestrating complex, stateful multi-agent workflows, particularly central to the Decision Engine.
- **Existing Tools:** Continued reliance on established systems like Kafka, DIVIDL, existing Machine Learning (ML) APIs, and Business Rules Services, which serve as "tools" for the agents to interact with.

2 Modules

2.1 Data Sources

- **SAPI Oracle ERP:** Enterprise Resource Planning system, providing core business data.
- **Salesforce:** Customer Relationship Management (CRM) platform, source of sales and customer data.
- **Operational Systems:** Various other systems providing data from daily operations.
- **Custom Solutions/DBs:** Bespoke databases or applications for specialised data.

2.2 Data Ingestion & Initial Processing (Agent-Enhanced)

This layer combines traditional data streaming with intelligent, agent-driven processing.

- **Connectors (React to business events):** Interface directly with source systems to ingest data in real-time or near real-time (planned for post-MVP integration).
- **Kafka:** Distributed streaming platform for high-throughput, low-latency data ingestion.
- **DIVIDL:** A data lake or initial processing component for storing raw and semi-processed data.
- **Data Ingestion Agents (Stream Processor Agent):**
  - **Role:** Monitor Kafka topics and DIVIDL for new data streams (initially synthetic data).
  - **Capabilities:** Utilises Ollama LLMs for intelligent parsing, hybrid validation (combining strict rules with semantic understanding), dynamic data routing based on content, rich metadata generation, and data summarisation for monitoring.
  - **Tools:** Kafka API/client libraries, DIVIDL APIs, database connectors, data validation libraries, data profiling tools.

- **Synthetic Data Generation Agent (Code Generator):**
  - **Role:** Generates synthetic data (tabular or time series) for MVP testing/development across various use cases.
  - **Capabilities:** Interprets natural language dataset descriptions and generates executable Python code (e.g., using Faker or SciPy) to produce realistic synthetic datasets. The LLM generates code, external processes execute it.
  - **Tools:** Access to a code execution environment, data generation libraries (Faker, SciPy).
- **Raw Gold Data / Source Gold Bronze Data:** Data zones representing raw and initially curated data.
- **Validation Strategy:**
  - **Synthetic Data Validation:** Statistical tests (e.g., Kolmogorov-Smirnov, Chi-Square) ensure synthetic data aligns with use case-specific distributions (e.g., financial transactions, customer interactions). Tools: SciPy, pandas. Success metric: p-value ¿ 0.05.
  - **Code Validation:** Generated Python code is tested using pytest with predefined test cases covering diverse use case scenarios. Success metric: 95% test case pass rate, no runtime errors. Human review ensures code efficiency.
  - **Data Ingestion Validation:** Hybrid validation combines rule-based checks with LLM-driven semantic validation to ensure data integrity across domains. Success metric: 98% accuracy in routing and metadata generation.
- **Performance Testing:**
  - **Scenarios:** Stress-test Kafka ingestion with high-throughput data streams (e.g., 1000 events/second for customer analytics, 500 transactions/second for fraud detection). Simulate LLM-driven parsing under load.
  - **Tools:** Apache JMeter for load testing Kafka, Locust for API testing, Prometheus for monitoring CPU/memory usage.
  - **Metrics:** Latency ¡ 100ms for data routing, throughput ¿ 1000 events/second, resource utilisation ¡ 80%.
  - **Mitigation:** Implement caching for LLM queries and parallel processing for data streams to optimise performance.
- **Integration Strategy:**
  - **Real-Time Integration (Post-MVP):** Connectors interface with Data Sources (e.g., SAPI Oracle ERP, Salesforce) to stream data into Kafka topics, processed by Data Ingestion Agents and stored in DIVIDL for use case-specific workflows.
  - **Error Handling:** Implement exponential backoff retries for failed Kafka consumer requests, dead-letter queues for unprocessable events, and logging (e.g., Log4j) for error tracking.
  - **Latency Optimisation:** Use batch processing for high-throughput streams and parallel processing for LLM-driven tasks to minimise latency.
  - **Tools:** Kafka Streams API, DIVIDL APIs, Log4j for logging, Prometheus for monitoring.
  - **Metrics:** Ingestion latency ¡ 50ms, error rate ¡ 1%, successful event processing ¿ 99%.

2.3 Predictive Engine (AutoML Agent)

Focuses on dynamic machine learning model creation and serving.

- **ML Repository:** Stores machine learning models, algorithms, and training data.
- **ML as an API:** Provides endpoints for serving trained ML models for inference.
- **AutoML Orchestration Agent:**
  - **Role:** Given a synthetic dataset and a task (prediction/classification), it intelligently proposes and generates ML model pipelines for various use cases.

- **Capabilities:** Utilises Ollama LLMs to analyse data, understand task requirements, and generate PyCaret code for data pre-processing, feature engineering, model selection, hyper-parameter tuning, and evaluation.
- **Tools:** PyCaret library, data analysis libraries, existing ML as an API, database access for training data.

- **Validation Strategy:**
  - **Pipeline Validation:** Generated ML pipelines are evaluated using cross-validation and performance metrics (e.g., F1-score, AUC) tailored to use case requirements. Tools: PyCaret, scikit-learn. Success metric: Model performance meets use case-specific thresholds (e.g., F1 ¿ 0.85 for classification).
  - **Human Review:** Domain experts review pipeline configurations for relevance to specific use cases.

2.4 Precision Engine (Natural Language Rules Agent)

Manages and applies business rules with enhanced flexibility via LLMs.

- **Business Rules Config Service:** Manages the definition and storage of business rules.
- **Business Rules Execution Service:** Executes defined business rules.
- **Business Rule Management Agent:**
  - **Role:** Facilitates intuitive creation, modification, and intelligent application of business rules.
  - **Capabilities:** Uses Ollama LLMs to interpret natural language rule descriptions from the UI, translating them into structured formats.
  - **Tools:** APIs to Business Rules Config Service and Business Rules Execution Service.
- **Validation Strategy:**
  - **Rule Translation Validation:** LLM-translated rules are cross-checked against a test suite of known scenarios for multiple use cases. Tools: Custom validation scripts, Business Rules Execution Service API. Success metric: 98% accuracy in translation.
  - **Human Review:** Business analysts verify rule translations for intent and correctness across domains.

2.5 Decision Engine (LangGraph Core)

The central intelligent hub for complex, adaptive decision-making, powered by LangGraph.

- **LangGraph-Powered Orchestration:** The core of this module, where the decision workflow is defined as a dynamic, stateful graph of interacting agents.
- **Decision Orchestration Supervisor Agent (LangGraph Graph):**
  - **Role:** The primary controller of the decision process. It dynamically determines the sequence of actions, manages context, handles collaboration between sub-agents, and generates explanations for decisions.
  - **Capabilities:** Adaptive workflow execution, contextual reasoning, inter-agent collaboration, explanation generation, and uncertainty handling (e.g., routing to Human-in-the-Loop).
- **Sub-Agents (Nodes within the LangGraph):**
  - **Prediction Request Agent:** Interacts with the ML as an API to retrieve predictions.
  - **Rule Application Agent:** Calls the Business Rules Execution Service to apply business logic.
  - **Data Query Agent:** Accesses internal databases or external APIs for additional contextual information.

  - **Human-in-the-Loop Agent:** Facilitates human intervention for review, approval, or override of decisions.
  - **Conflict Resolution Agent:** Identifies and attempts to reconcile conflicting information between ML predictions and business rules.

- **Tools for Decision Engine Agents:** APIs to ML as an API, Business Rules Execution Service, internal databases, external APIs, logging/auditing services.

- **Validation Strategy:**
  - **Explanation Validation:** Decision explanations are evaluated for clarity and completeness using a predefined rubric. Tools: Logging/auditing services, custom evaluation scripts. Success metric: 90% of explanations rated as "clear" by human reviewers; no critical logical errors.
  - **Conflict Resolution Validation:** Conflicts are logged and reviewed to ensure resolution logic aligns with use case goals. Success metric: 95% of conflicts resolved without human intervention.

- **Performance Testing:**
  - **Scenarios:** Stress-test LangGraph workflows with complex decision paths (e.g., 100 concurrent decisions for supply chain optimisation). Simulate LLM inference under high load.
  - **Tools:** Locust for API testing, Prometheus for monitoring, custom scripts for workflow benchmarking.
  - **Metrics:** Decision latency ¡ 100ms, throughput ¿ 100 decisions/second, resource utilisation ¡ 80%.
  - **Mitigation:** Use load balancing for LangGraph nodes and caching for repetitive LLM queries to optimise performance.

2.6 Orchestrator

- **Role:** Acts as the primary API gateway, routing user requests from the UI to the appropriate entry points within the LangGraph-powered Decision Engine and translating responses back to the UI.

- **Functionality:** Handles communication between the frontend and the agentic backend, with much of the complex workflow logic now residing within LangGraph.

2.7 User Interface (UI)

Provides a comprehensive interface for user interaction and system management.

- **Simulation Configuration:** Allows users to define and run decision scenarios.

- **Define Business Rules:** Users can define rules using natural language, interpreted by the Business Rule Management Agent.

- **Define Workflow (Visual LangGraph Editor):** Enables users to visually design, modify, and monitor the LangGraph workflows for the Decision Engine (planned for post-MVP).

- **Simulate by Changing Variables:** Allows for testing decision logic with varied inputs, enhanced by step-by-step agent reasoning explanations.

- **User Journey Definition:** For defining and simulating end-to-end user processes that interact with the agent system.

- **Agent Interaction Interface:** A new feature for monitoring agent activity, reviewing explanations for decisions, and providing human input or overrides (basic version for MVP).

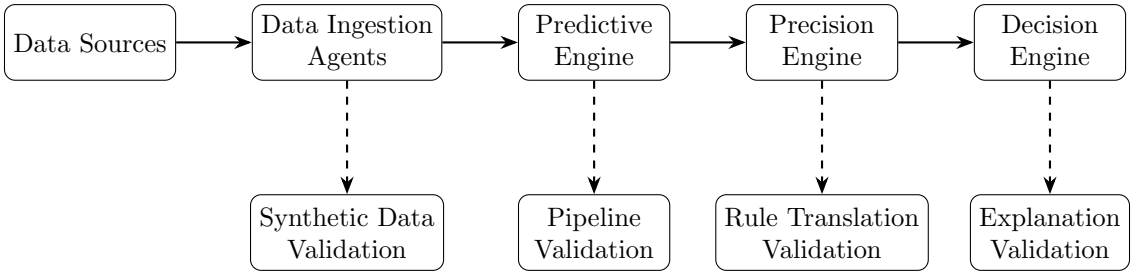

Figure 1: Validation Architecture for Project-X

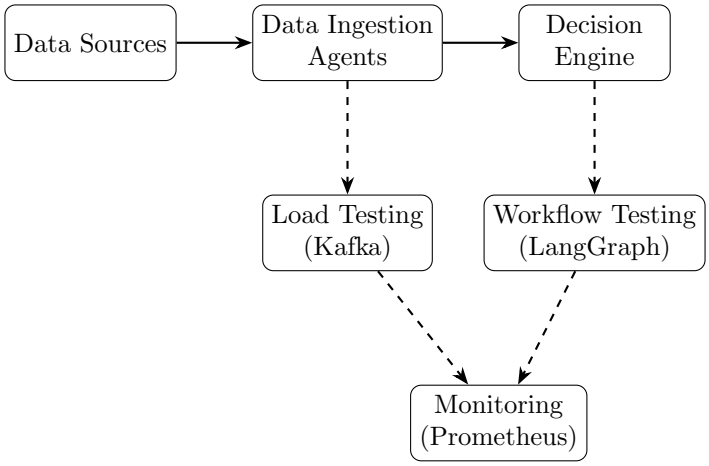

Figure 2: Performance Testing Architecture for Project-X

2.8 Validation Architecture

2.9 Performance Testing Architecture

2.10 Integration Architecture

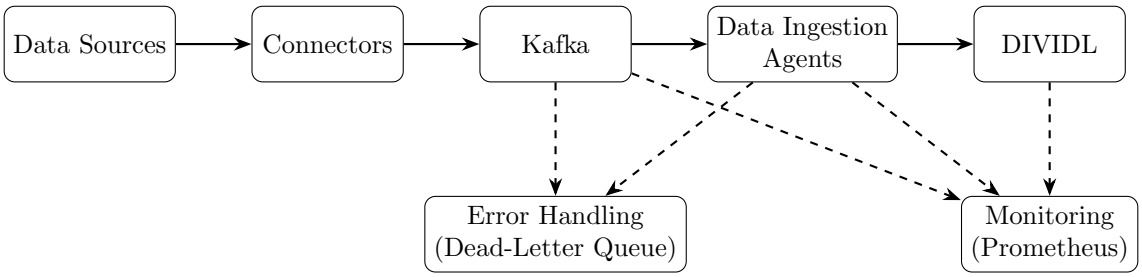

Figure 3: Integration Architecture for Project-X

2.11 Global Architecture

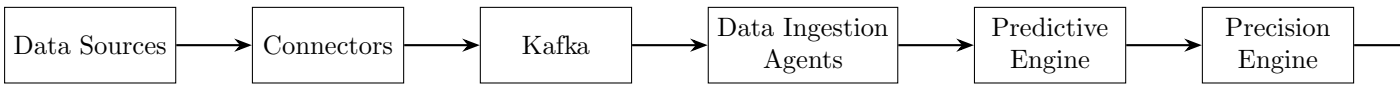

Figure 4: Global Architecture of the New Agent-Based System