AI-Powered Quotation Lifecycle Engine

Summary

1. Business Context

A significant share of revenue in many B2B contexts is driven by customer-initiated quotation requests. Traditional quotation processes often rely on manual workflows for CRM data handling, supplier communication, and quote evaluation, resulting in prolonged lead times and reduced win rates.

By introducing intelligent, data-driven automation, the proposed solution streamlines operations and reduces manual effort across the entire quotation lifecycle. This enables faster and more consistent commercial decision-making.

The system integrates with customer inquiry sources (e.g., HubSpot CRM), automates the standardisation and dispatch of RFQs to qualified suppliers, aggregates responses, evaluates them using machine learning models, and generates optimised quotes aligned with business profitability rules.

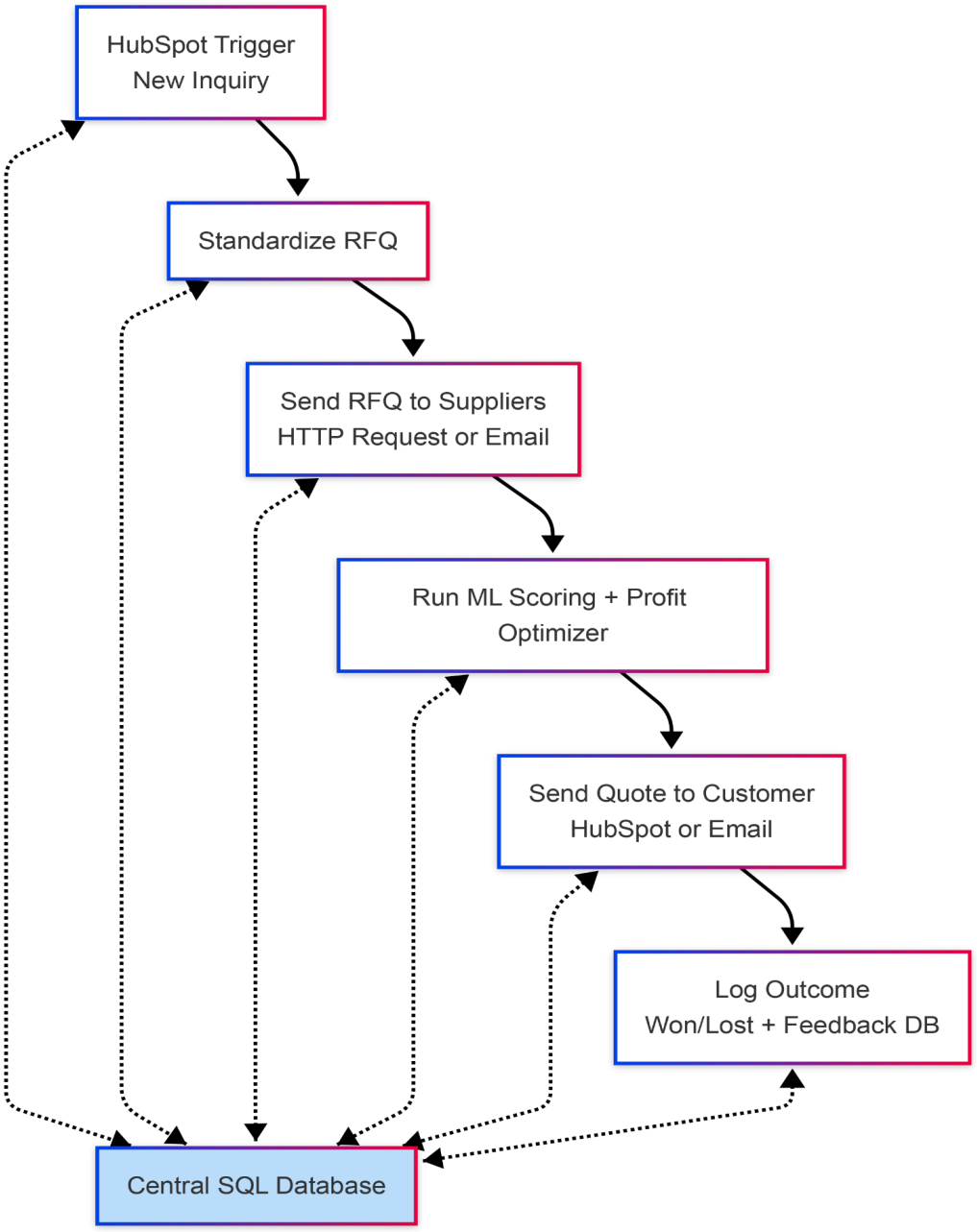
---

2. Objectives

A working prototype, hosted in the GitHub repository, demonstrates both the technical feasibility and business potential of this approach. The architecture is designed for scalability, auditability, and integration with broader enterprise ecosystems.

---

3. High-Level Architecture Diagram



---

4. Component Descriptions

4.1 Customer Inquiry Capture

- **Input**: Simulated customer requests received via HubSpot CRM, emulated through a FastAPI endpoint.

- **Process**: Parses and validates inquiry fields, converting raw CRM data into a structured, machine-readable format.

- **Output**: Normalized input data prepared for downstream RFQ generation.

4.2 Supplier Communication

- **Input**: Structured customer requirements

- **Process**:

    1. Standardizes RFQ formatting using predefined templates.

    2. Dynamically identifies and filters suppliers based on service compatibility (e.g., only suppliers offering the specific requested category or item).

    3. Dispatches tailored RFQs to the selected supplier pool via automated pipelines.

- **Output**: Targeted, structured RFQs delivered to eligible suppliers.

4.3 Data Aggregation and Analysis

- **Input**: Supplier responses

- **Process**: Collects and stores all quotation data in a local SQLite database (`quotations.db`), retaining relevant metadata (e.g., price, delivery estimate, supplier ID).

- **Output**: A centralized, queryable dataset used for automated scoring and analytics.

4.4 Quote Scoring Engine

- **Input**: Supplier responses containing key features such as unit price, total cost, delivery time, and a heuristic supplier performance score.

- **Process**:

    1. Selects and trains the most suitable machine learning model for quote acceptance prediction (Logistic Regression used in the prototype).

    2. Applies the trained model to assign a dynamic probability of acceptance ("won" likelihood) to each incoming quote.

- **Output**: Scored quotes annotated with estimated acceptance probabilities.

4.5 Quote Optimisation

- **Input**: ML-scored supplier quotes

- **Process**:

    1. Rejects quotes that fall short of required profitability thresholds (e.g., gross margin limits).

    2. Selects the quote with the highest composite score among those that meet business constraints.

- **Output**: Final customer-ready quote, optimized for both competitiveness and profitability.

4.6 Quote Presentation and Feedback

- **Input**: Final customer-ready quote
- **Process**:
    1. Delivers the quote to the client via integrated channels (e.g., email, dashboard).
    2. Tracks customer response and quote acceptance status.
- **Output**: Feedback data logged for use in model evaluation and periodic retraining.

---

5. Technology Stack

Component	**Technology**
Backend Scripts	Python 3.x
Web Framework	Flask, FastAPI
ML Libraries	scikit-learn, XGBoost, LightGBM, pandas, numpy, scipy, imbalanced-learn
Data Storage	SQLite (for prototyping), SQLAlchemy
CRM Integration	HubSpot API (simulated via FastAPI)
Visualisation	matplotlib, seaborn, HTML/CSS (via Jinja templates)
Synthetic (mock) data	random, faker
API Serving	Uvicorn, Starlette

6. Synthetic (Mock) Data

All data used in this project was fully synthetic (mocked) and created exclusively for demonstration purposes. This includes simulated CRM requests, RFQ generation, supplier responses, and the machine learning training dataset.

Data was generated with controlled randomness to embed minimal patterns that enable supervised learning without reflecting any real-world market distribution. For example, the training set was constructed to contain approximately 20% positive samples ("won"), using a scoring logic based on features such as unit price, delivery time, supplier response time, and performance.

The final label assignment (`won`) was determined using the following heuristic:

$$P(\text{won}) = \sigma(\alpha \cdot \text{score}_{\text{base}} + \varepsilon), \quad \text{where} \quad \sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\text{score}_{\text{base}} = 0.5 \cdot (1 - \text{unit\_price}) + 0.3 \cdot (1 - \text{delivery\_days}) + 0.2 \cdot \text{quantity}$$

A final decision score was computed by adjusting this base score with controlled noise, penalties, and bonuses (e.g., urgency, supplier performance, RFQ complexity). This adjusted score was then used to filter which samples would be labeled as positive until reaching the target 20% win rate.

Additionally, a 1% label flipping was applied to both classes to simulate outliers and prevent the model from overfitting or becoming overly sensitive to synthetic patterns.

---

7. Machine Learning Model

Model Selection

Logistic Regression was selected after benchmarking more complex alternatives (Random Forest, XGBoost, LightGBM). Although it slightly underperforms on certain metrics, it was preferred due to its:

- Simplicity

- Low computational cost
- High interpretability — a key factor for business stakeholders and non-technical decision-makers

  Visualizations highlighting this interpretability are provided in 13.2 Dashboard

Model Details

- **Model Features (***Engineered solely for simulation purposes)*:
  - Unit price
  - Delivery days
  - Historical supplier score
  - Response time
  - RFQ complexity score
  - `is_urgent` (binary)
  - `is_custom` (binary)
- **Target**: Binary classification — whether the quote is accepted ("won")
- **Scaler**: MinMaxScaler
- **Hyperparameter Tuning**: Conducted using GridSearchCV
- **Storage**: Trained models are stored in `/src/models/` as `.pkl` files
- **Model Training and Prediction**: Training is performed using a binary classifier. Predictions return a continuous score indicating the probability of class membership in the "accepted" class, which feeds into the quote optimization logic.

8. Database Design & Persistence

All data flows across the quotation lifecycle are persisted in a centralised SQL database. During prototyping, SQLite was selected for its simplicity and low setup overhead. The database schema was designed to be PostgreSQL-compatible to support seamless migration to production environments.

The schema supports the following core entities and their relationships:

- **Customers**: Basic client profile and metadata
- **Client Requests**: Product inquiries initiated by the customer
- **Suppliers**: Registered partners, including performance history and product catalogue
- **RFQs Sent**: Mapping between client requests and outbound RFQs
- **RFQ Details**: Item-level requirements, delivery terms, and other specifications
- **Quotation Responses**: Supplier-submitted quotes with pricing and lead times
- **Quote Scores**: Model-derived ranking scores for each response
- **Training and Real Quotation Data**: Labelled datasets used in model development
- **Merged and Selected Quotes**: Optimised quotes sent to clients

This architecture is illustrated in [13.1 SQL Database]

9. Scalability & Production Considerations

Microservices:

The architecture follows a modular microservices pattern, enabling containerization via Docker and separation of responsibilities across services (e.g., scoring engine, RFQ manager, retraining pipeline).

CI/CD Compatibility:

GitHub Actions is supported for automated testing, deployment, and model updates across environments.

Model Versioning:

Hash-based versioning is planned for tracking each trained ML model, enabling reproducibility and facilitating rollback when necessary.

Monitoring & Observability:

Model drift detection, input/output distribution logging, and scoring latency tracking are planned features to be supported via pluggable monitoring modules in future iterations.

Database Scalability:

While SQLite is used for prototyping, the system is fully compatible with PostgreSQL for production deployments. PostgreSQL supports connection pooling, parallel queries, schema migrations, and high-concurrency workloads.

Integration-Oriented Design:

The solution is designed to interoperate with automation platforms like **n8n**. This allows horizontal scalability through asynchronous, event-driven task orchestration — reducing bottlenecks in RFQ generation, scoring, and feedback collection.

---

10. AI Agent & Workflow Automation (Proposed Roadmap)

To move beyond traditional automation, a modular **AI Agent architecture** is proposed. This system leverages the **n8n** automation platform to orchestrate the entire quotation lifecycle — from CRM ingestion to quote delivery and feedback-based retraining.

This architecture emphasizes **autonomy, reproducibility, and deployment resilience**, driven by a containerized infrastructure.

Containerization Strategy (via Docker):

- **n8n Runtime**: Preconfigured with API credentials, workflows, and orchestrator logic;
- **Python Microservices**: Model inference, scoring, and retraining logic exposed as REST APIs or CLI tasks;
- **Database Layer**: PostgreSQL instance preloaded with schema definitions and optional test data.

Deployment Orchestration:

- **Docker Compose** for local and mid-scale deployments
- **Kubernetes** as an optional layer for production-scale orchestration (e.g., horizontal pod scaling, health checks, service discovery)

Step	Description
Customer Inquiry Collection	Real-time ingestion of CRM requests via **HubSpot API** into PostgreSQL.
RFQ Generation & Dispatch	n8n HTTP Request nodes or custom Python functions send RFQs to suppliers.
Quotation Response Aggregation	Supplier responses normalized into a central

	PostgreSQL database.
Quote Scoring & Optimization	ML workflows (Python) triggered by n8n, with full traceability and logging.
Quote Delivery	Best quotes returned to the CRM, emailed to clients, or routed to a dashboard.
Sales Outcome Feedback	Deal outcomes logged in PostgreSQL for retraining insights.
Retraining	Scheduled via n8n's cron nodes; runs versioned containerized training jobs
Notifications & Monitoring	Slack/email alerts for key events (e.g., quote sent, model retrained).

This architecture is illustrated in [13.3 AI Agent + Workflow Architecture]

Future-Proofing:

- **LLM Integration** for supplier response parsing or auto-generated RFQs
- **Constraint-based solvers** for quote optimization
- **Auto-reasoning agents** to evaluate trade-offs in cost vs. delivery

Conclusion:

The AI Agent architecture, when combined with Docker-based isolation and n8n orchestration, forms a highly scalable, maintainable, and production-ready quotation intelligence system.

---

11. Prototype Summary

Repository: https://github.com/neural-insights/quotation-lifecycle-ai

To Run:

```
git clone https://github.com/neural-insights/quotation-lifecycle-ai
cd quotation-lifecycle-ai

# Setting up the environment
python -m venv .venv
source .venv/bin/activate            # or .venv\Scripts\activate on Windows
pip install -r requirements.txt

# Exporting Python path so modules in /src can be resolved
export PYTHONPATH="$(pwd)/src"

# Starting the internal database/API server (used by the pipeline)
uvicorn src.utils.main:app --reload  # Serves internal API at http://127.0.0.1:8000

# Running initialization and launching UI
python run_all.py                    # Executes full pipeline:
                                     # from DB initialization to final optimized scoring
python app/main.py                   # Launches the Flask web interface
```

Key Endpoints in Flask Web Interface

This will serve the app at: ➡️ http://127.0.0.1:5000/

Route	Description
/	Home
/model-dashboard	ML Model Visualization (Metrics, Confusion Matrix and Features Importance)
/selected-quotes	Optimized Quotes Overview
/api/selected-quotes	Selected quotes in JSON format
/supplier-performance	Supplier Performance Metrics

12. Business Impact

Operational Efficiency

Drastically reduces manual workload by automating RFQ generation, supplier communication, response aggregation, scoring, and feedback integration—accelerating turnaround times and lowering operational overhead.

Increased Win Rates

Leverages historical data and machine learning to score and prioritize quotations with a higher likelihood of customer acceptance. As the system continuously incorporates sales outcomes into retraining routines, acceptance rates are expected to improve over time.

Profitability

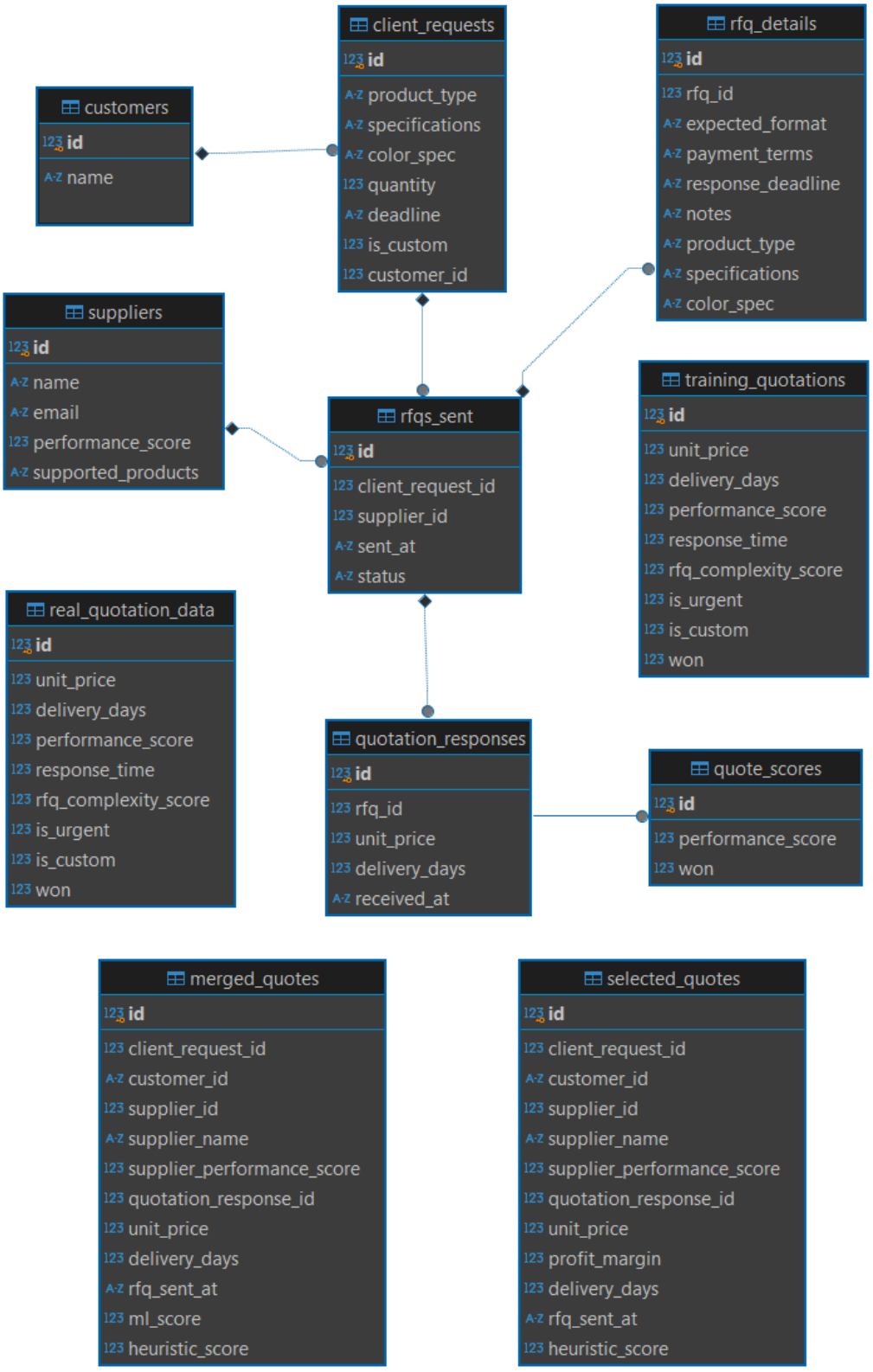
Enforces business constraints such as margin thresholds to ensure only financially viable quotes are sent to customers.

Strategic Alignment

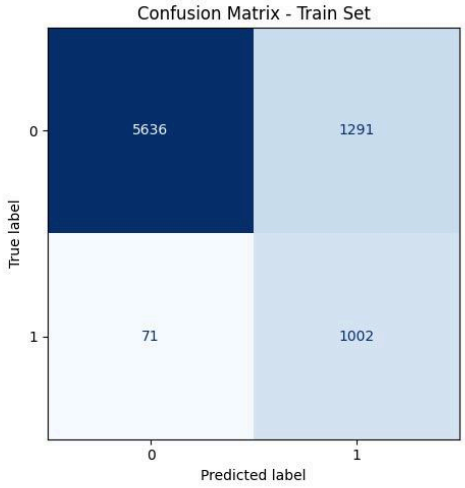
Aligns with organizational priorities around automation, scalability, and data-driven operations—transforming fragmented workflows into an integrated, intelligent pipeline.

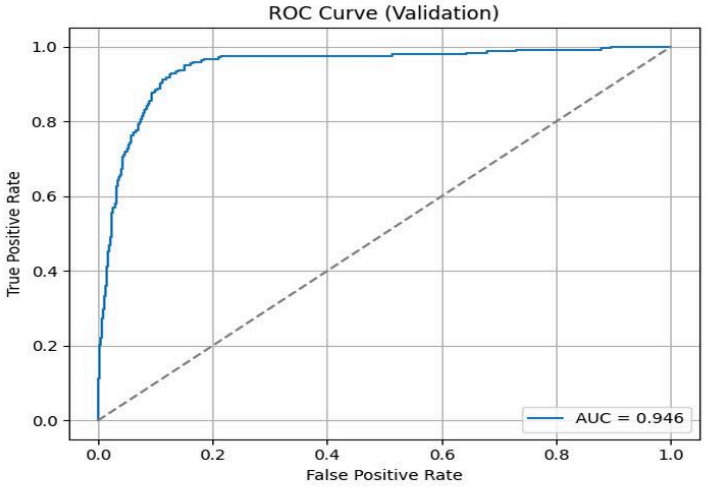
13. Appendix

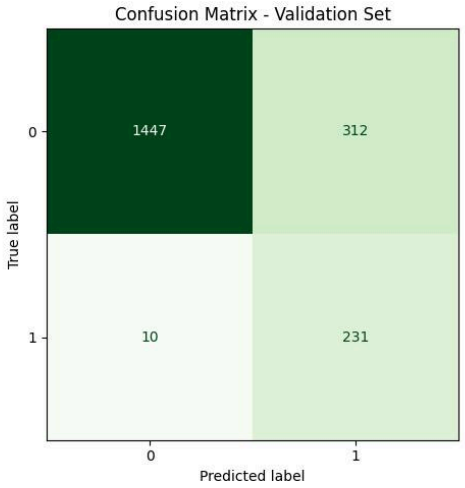
13.1 SQL Database

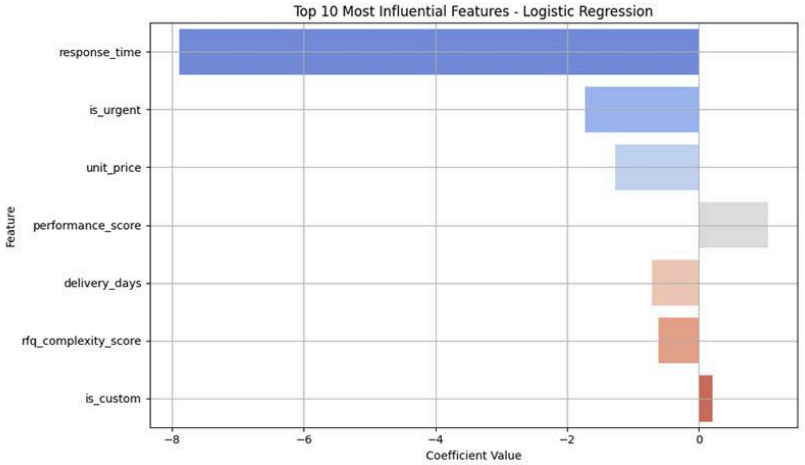


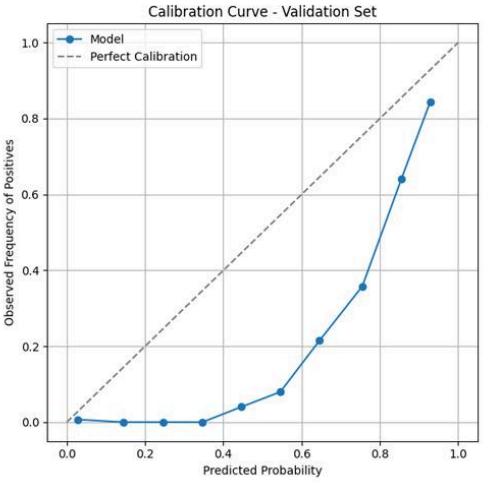
13.2 Dashboard (ML Model and Results)

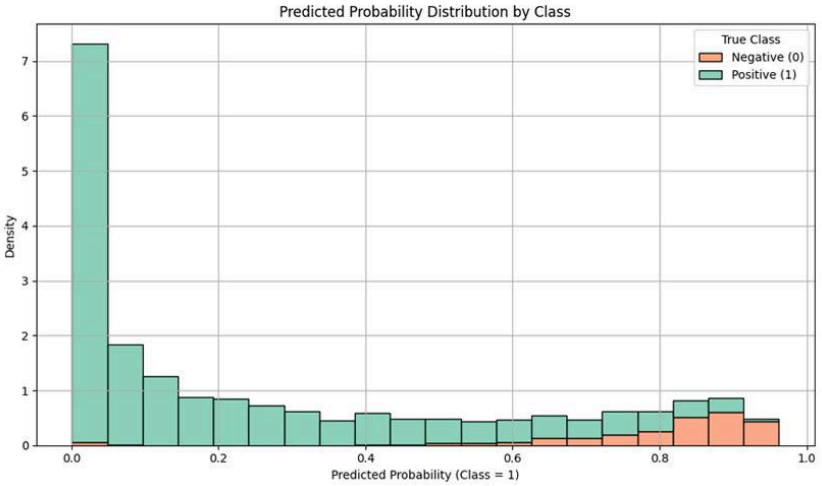












13.3 AI Agent + Workflow Architecture