**Store Replenishment Platform**

**Group Number:** Group 4

**Group Members:**

Liu Zheyi A0298521X

Cen Haoyang A0297892A

Gao Zijie A0295363W

Lian Da A0295323A

Wu Hongjia A0297459E

**Date:** 25/10/2025

Context

[1. Introduction 5](#_Toc212289684)

[Project Description 5](#_Toc212289685)

[Demand Forecasting 5](#_Toc212289686)

[Inventory Policy & Replenishment 5](#_Toc212289687)

[Top-K Refill & CoT Explainability 6](#_Toc212289688)

[Optional Module: Price Forecasting & Price Elasticity 6](#_Toc212289689)

[Project Value 6](#_Toc212289690)

[2. Project Background / Market Context 6](#_Toc212289691)

[Problem Definition 6](#_Toc212289692)

[Research Significance 7](#_Toc212289693)

[Industry Context 7](#_Toc212289694)

[3. Market Research 8](#_Toc212289695)

[Existing Methods 8](#_Toc212289696)

[Limitations of Current Solutions 8](#_Toc212289697)

[Opportunity and Contribution 8](#_Toc212289698)

[4. Project Scope 9](#_Toc212289699)

[Boundaries and Focus 9](#_Toc212289700)

[Key Functionalities: 9](#_Toc212289701)

[Intelligent Reasoning Systems Techniques in Focus 10](#_Toc212289702)

[Out of Scope 11](#_Toc212289703)

[Academic Value & Market Limitations 11](#_Toc212289704)

[5. Data Collection and Preparation 11](#_Toc212289705)

[Data Sources 11](#_Toc212289706)

[Data Preprocessing 12](#_Toc212289707)

[6. System Design 13](#_Toc212289708)

[Nine Core Steps 13](#_Toc212289709)

[7. Implementation 16](#_Toc212289710)

[Workflow Architecture 17](#_Toc212289711)

[System Components 17](#_Toc212289712)

[Deployment and Configuration 17](#_Toc212289713)

[Technical challenges 18](#_Toc212289714)

[1. Data Conditions 18](#_Toc212289715)

[2. Modeling Conditions 18](#_Toc212289716)

[3. System Architecture Conditions 18](#_Toc212289717)

[4. Integration & Deployment Conditions 18](#_Toc212289718)

[5. Scalability & Extensibility 19](#_Toc212289719)

[8. Results and Progress 19](#_Toc212289720)

[Overview 19](#_Toc212289721)

[Forecast Performance 19](#_Toc212289722)

[Replenishment Recommendations 20](#_Toc212289723)

[System Dashboard and Operational Statistics 20](#_Toc212289724)

[Smart Replenishment Assistant 21](#_Toc212289725)

[Automated Reporting and Explainability 23](#_Toc212289726)

[Dashboard Workflow Automation 26](#_Toc212289727)

[Reasoning and Report Generation Workflow 26](#_Toc212289728)

[Project Progress Summary 28](#_Toc212289729)

[Key Insights and Achievements 28](#_Toc212289730)

[9. Challenges and Roadblocks 29](#_Toc212289731)

[Data Preparation and Database Integration 29](#_Toc212289732)

[Forecast Model Optimization 29](#_Toc212289733)

[API Integration and Workflow Automation 30](#_Toc212289734)

[LLM Reasoning and Explainability 30](#_Toc212289735)

[Resource and Collaboration Constraints 31](#_Toc212289736)

[Key Lessons Learned 31](#_Toc212289737)

[10. Future Work 32](#_Toc212289738)

[Multi-Store and Multi-Region Extension 32](#_Toc212289739)

[Real-Time and Event-Driven Replenishment 32](#_Toc212289740)

[Enhanced LLM Explainability and Multi-Model Reasoning 33](#_Toc212289741)

[11. Conclusion 33](#_Toc212289742)

[Core Project Contributions 33](#_Toc212289743)

[Potential Value for Market 34](#_Toc212289744)

[Significance for Intelligent Reasoning Systems Field 34](#_Toc212289745)

[12. Appendices 34](#_Toc212289746)

[Appendix A – Original Project Proposal 35](#_Toc212289747)

[Appendix B – Installation and User Guide 35](#_Toc212289748)

[Appendix C – Skill Mapping Table (MR / RS / CGS) 35](#_Toc212289749)

[Appendix D – AI Tool Usage Declaration 36](#_Toc212289750)

[Appendix E – Acknowledgements 36](#_Toc212289751)

1. Introduction

## Project Description

This project presents a prototype intelligent replenishment system that we have developed to assist store managers in making data-driven restocking decisions.

The front-end interface allows managers to view replenishment recommendations for upcoming time periods.

The system integrates historical sales and price data from the same store and comparable regions, together with current inventory levels, to produce an optimized replenishment list and recommended quantities for the next T days (configurable).

To ensure interpretability and automation, the system combines Chain-of-Thought (CoT) reasoning, n8n workflow orchestration, and Model Context Protocol (MCP) for external API integration.

The final system architecture consists of three main modules:

### Demand Forecasting

We implemented multiple time-series forecasting models — including 14-day moving average (MA14), ARIMA, Prophet, and LSTM — to predict store-product-level sales for upcoming periods.

These models optionally incorporate exogenous factors such as price, holidays, and promotions, forming the demand baseline that drives subsequent replenishment logic.

### Inventory Policy & Replenishment

Based on current inventory, lead time (if available), target service level (e.g., 95%), and the replenishment horizon *T*, lightweight inventory policies such as **(s, S)** or order-up-to rules are applied to calculate recommended order quantities. In the absence of cost parameters, replenishment decisions are primarily guided by stockout risk, service levels, and safety stock. The module outputs both the suggested delivery date and recommended replenishment quantity.

### Top-K Refill & CoT Explainability

This layer presents store managers with the Top-K products (e.g., K=5, configurable) that should be prioritized for replenishment, along with the recommended quantities. Explanations are generated through CoT reasoning, expressed in natural language (e.g., *“Because sales typically peak during this week in the past three years + current inventory is low + a holiday is approaching”*). This layer integrates n8n and MCP to orchestrate data retrieval, task scheduling, and optional external API calls (e.g., holidays, weather) for enhanced decision support.

## Optional Module: Price Forecasting & Price Elasticity

When sufficient price history is available, short-term price forecasting or price elasticity estimation can be applied to adjust the demand baseline, thereby fine-tuning replenishment recommendations.

### Project Value

This project demonstrates how explainable and data-efficient replenishment can improve inventory accuracy and service levels in environments where warehouse and cost data are limited.It provides an interpretable decision-support tool for retail managers, bridging the gap between AI-driven reasoning and practical business deployment.

# Project Background / Market Context

## Problem Definition

Many retail stores still rely on experience and simple reports to make replenishment decisions:

* Demand fluctuations and seasonality often result in alternating periods of stockouts and overstocking.
* Available data is usually limited to sales, prices, and inventory snapshots, lacking end-to-end visibility of warehouse, geographic, transportation, and cost information.
* Manual judgment struggles to balance replenishment schedules and safety stock for a future window of *T* days, often missing holiday effects or promotional impacts.

## Research Significance

For small and medium-sized retailers, in the absence of large-scale supply chain platforms and comprehensive data assets, there is still a strong need for a lightweight, interpretable, and easily deployable replenishment recommendation tool. By starting with minimal store-side inputs (historical sales/prices + current inventory), the system provides answers to *“what to replenish, how much, and why”* for the next *T* days, which helps to:

* Reduce stockout risk, improve service levels, and increase sales conversion.
* Minimize overstock and reduce cash tied up in inventory.
* Lay the foundation for integrating additional data in the future (e.g., purchase cost, lead time, supplier constraints).

## Industry Context

With the ongoing digitalization of retail, the availability of POS data has significantly improved. However, store-level intelligent replenishment often remains limited to static reports or simple threshold-based rules. Large platforms (e.g., Blue Yonder, o9, Kinaxis) focus on end-to-end supply chains and entail high implementation costs, making them less accessible to small and medium retailers. This creates a practical demand for academically grounded, lightweight, and reproducible methodologies and tools that can be validated using open datasets (e.g., Kaggle’s M5 competition data) for prototyping and benchmarking.

# Market Research

## Existing Methods

* **Traditional inventory policies**: EOQ, Newsvendor, (s, S), Base-stock, ABC/XYZ classification, Croston’s method (for intermittent demand), etc.
* **Demand forecasting models**: ARIMA/ETS, Prophet, gradient boosting, RNN/LSTM, Transformer, etc.
* **Current practice**: Most small and medium-sized stores rely on ERP/POS/WMS systems that provide only basic visualization and threshold-based alerts, making it difficult to close the loop from forecasting + inventory policy to interpretable recommended quantities.

## Limitations of Current Solutions

* Dependence on comprehensive supply chain data (inbound/outbound/inventory, lead times, costs, geographic and logistics capacity), which is unrealistic for small and medium retailers.
* High degree of black-box complexity and high implementation costs, making rapid piloting difficult.
* Poor adaptability of existing solutions to data-scarce store-side environments where only sales, prices, and inventory snapshots are available.

## Opportunity and Contribution

This project targets store operators by delivering a lightweight prototype that is **forecast-driven**, **inventory-policy-centered**, and **CoT-enabled for explainability**, with the following characteristics:

* **Minimal data requirements**: Operates with only historical sales/prices and current inventory; optional integration of exogenous variables such as holidays and weather.
* **Explainable reasoning (CoT)**: Translates *“why these SKUs/quantities are recommended”* into natural language explanations, facilitating business adoption and parameter adjustment.
* **Workflow automation (n8n + MCP)**: Enables one-click data retrieval, scheduled forecasting, and automated recommendation report generation, making integration with existing IT systems straightforward.
* **Reproducible validation**: Uses open datasets (e.g., M5 competition data) for methodological validation and benchmarking, lowering the barrier between academic research and practical deployment.

Through these approaches, the project aims to deliver actionable replenishment recommendations for data-constrained store environments, emphasizing **interpretability, tunability, and ease of integration**, while leaving room for future extensions such as supplier constraints, cost functions, and cross-store transshipment.

# Project Scope

## Boundaries and Focus

This project focuses on sales forecasting, replenishment decision-making, using the M5 Walmart dataset (sales\_train\_evaluation.csv, sell\_prices.csv, calendar.csv). Since the original dataset does not include current inventory information, we extend the dataset by maintaining a synthetic inventory table to enable replenishment decisions.

## Key Functionalities:

1. Sales Forecasting – Predict item-level daily demand using historical sales, calendar events, and price features.

2. Replenishment Recommendation (with Decision Trace / Explainability) – Combine forecasted demand, current (simulated) stock, safety stock, and price trend to produce a transparent, auditable recommendation:

Step 1: Forecast future demand (e.g., 200 units)

Step 2: Read current stock (e.g., 120 units)

Step 3: Add safety stock (e.g., 50 units)

Step 4: Consider price trend (e.g., +8% expected → advance replenishment to avoid higher future cost/stock-out risk)

Step 5: Compute recommended replenishment:

Output includes both the numeric recommendation and a structured reasoning chain for transparency.

Priority Replenishment Recommendation – Recommend which items to prioritize (e.g., “high forecast + low stock + upcoming holiday”) and show the rationale.

3. Decision Trace (Explainability) – Provide a concise, human-readable path showing how each decision was reached (inputs → policy → output → reasons).

4. LLM-powered Report Generation Dashboard – Generate automated business reports (e.g., weekly replenishment summary,demand forecast overview) using an LLM.

## Intelligent Reasoning Systems Techniques in Focus

Knowledge Discovery & Data Mining – Time-series forecasting with models such as ARIMA, Prophet and LSTM; feature attribution with SHAP to interpret drivers (e.g., holidays, promotions, price changes).

Business Resource Optimization – Apply replenishment rules (e.g., Q=max⁡ (0, forecast+safety stock−current stock)), inventory control heuristics, and service-level-based safety stock calculations.

Decision Automation – Rule-based dynamic pricing and replenishment policies, ensuring consistency with business objectives.

Priority Replenishment Recommendation – Recommend which items should be prioritized for replenishment, along with reasoning (e.g., “High forecasted demand + low current stock + upcoming holiday event”).

Chain-of-Thought Reasoning (Explainability) – Provide a transparent reasoning path showing how the system arrives at its decisions, e.g.:

Forecasted demand = 120 units

Current stock = 50 units

Safety stock = 30 units

So, Replenishment quantity = 100 units

## Out of Scope

Real-world ERP/POS system integration.

Advanced competitive pricing strategies (e.g., scraping competitors’ prices).

## Academic Value & Market Limitations

Academic Value – Comparative evaluation of forecasting methods on a large-scale benchmark dataset (M5), integrated with reasoning-based replenishmen.

Market Limitations –The M5 dataset lacks true inventory levels; inventory tables are synthetically generated (e.g., random initialization, or derived from average weekly sales). Practical deployment would require integration with live inventory systems and more granular operational constraints.

# Data Collection and Preparation

## Data Sources

1. **Open-source dataset**: M5 Walmart dataset on Kaggle – [M5 Forecasting Accuracy](https://www.kaggle.com/competitions/m5-forecasting-accuracy/data).

**calendar.csv** - Contains information about the dates on which the products are sold.

**sales\_train\_validation.csv** - Contains the historical daily unit sales data per product and store [d\_1 - d\_1913]

**sell\_prices.csv** - Contains information about the price of the products sold per store and date.

**sales\_train\_evaluation.csv** - Includes sales [d\_1 - d\_1941] (labels used for the public leaderboard)

1. **Synthetic inventory table** (current\_inventory) – Created and maintained within the system database to simulate current stock levels.

## Data Preprocessing

**1. Cleaning & Alignment**

Align daily sales with calendar events and weekly prices.

**2. Formatting & Integration**

Encode categorical features (e.g., store, item category, event type).

Join forecast results with current\_inventory to compute replenishment quantities.

**3. Feature Engineering**

Time-series features: lags (7, 14, 28 days), moving averages, seasonality.

Calendar features: holidays, special events, weekday/weekend indicators.

Price features: discount rates, price volatility.

**4. Safety Stock Estimation**

Option 1: Use forecast standard deviation × z-score (e.g., 1.65 for 95% confidence).

Option 2 (simpler): Set safety stock as a fixed percentage of forecast demand (e.g., 20%).

**Challenges and Solutions**

No true inventory data – Solution: simulate current stock levels using random initialization or heuristic rules (e.g., 7-day average sales × factor).

Forecast uncertainty – Solution: apply safety stock buffers to mitigate stockout risk.

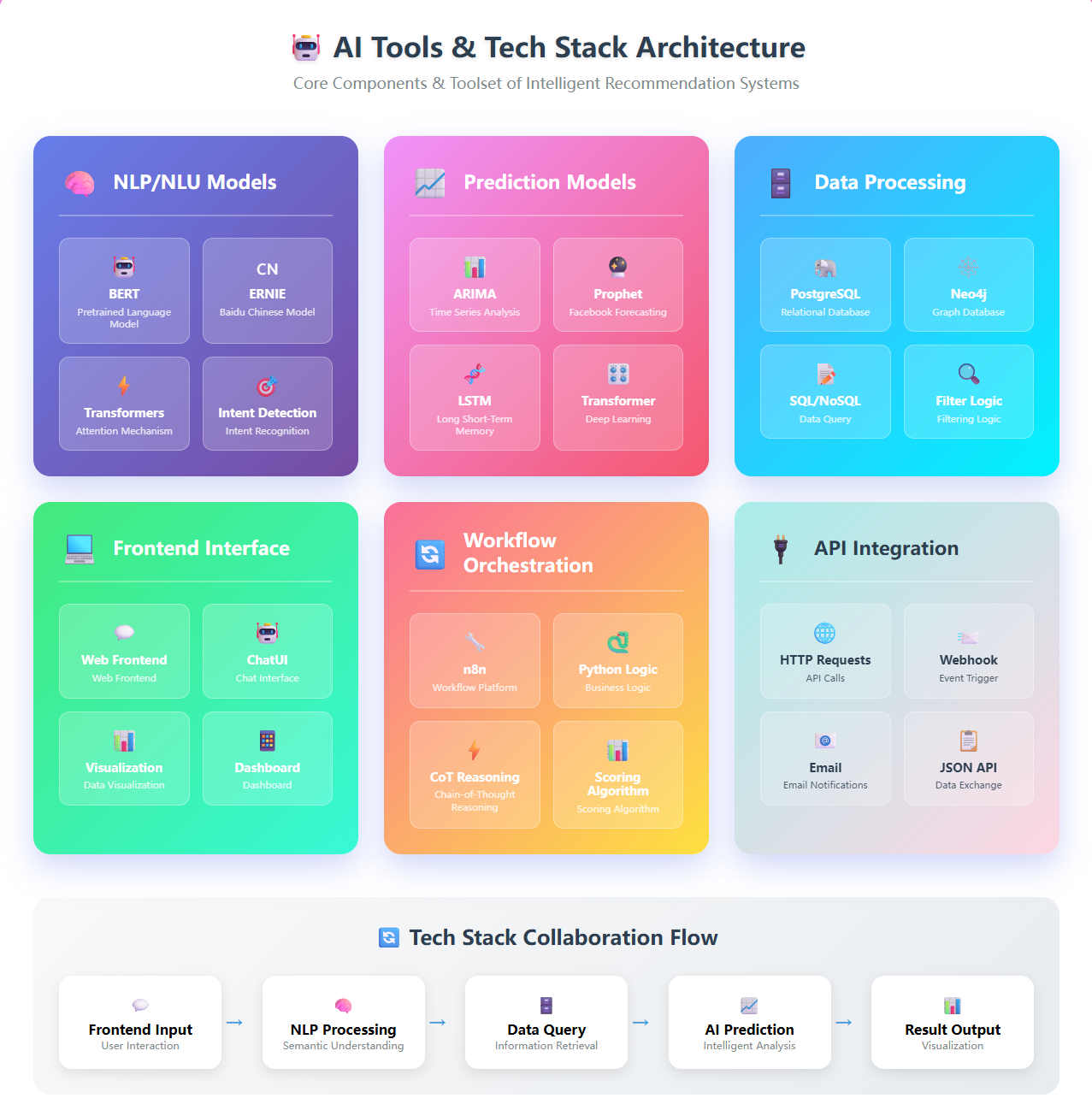
Scalability – M5 dataset contains millions of records; optimize with sampling, aggregation, or efficient ML models.

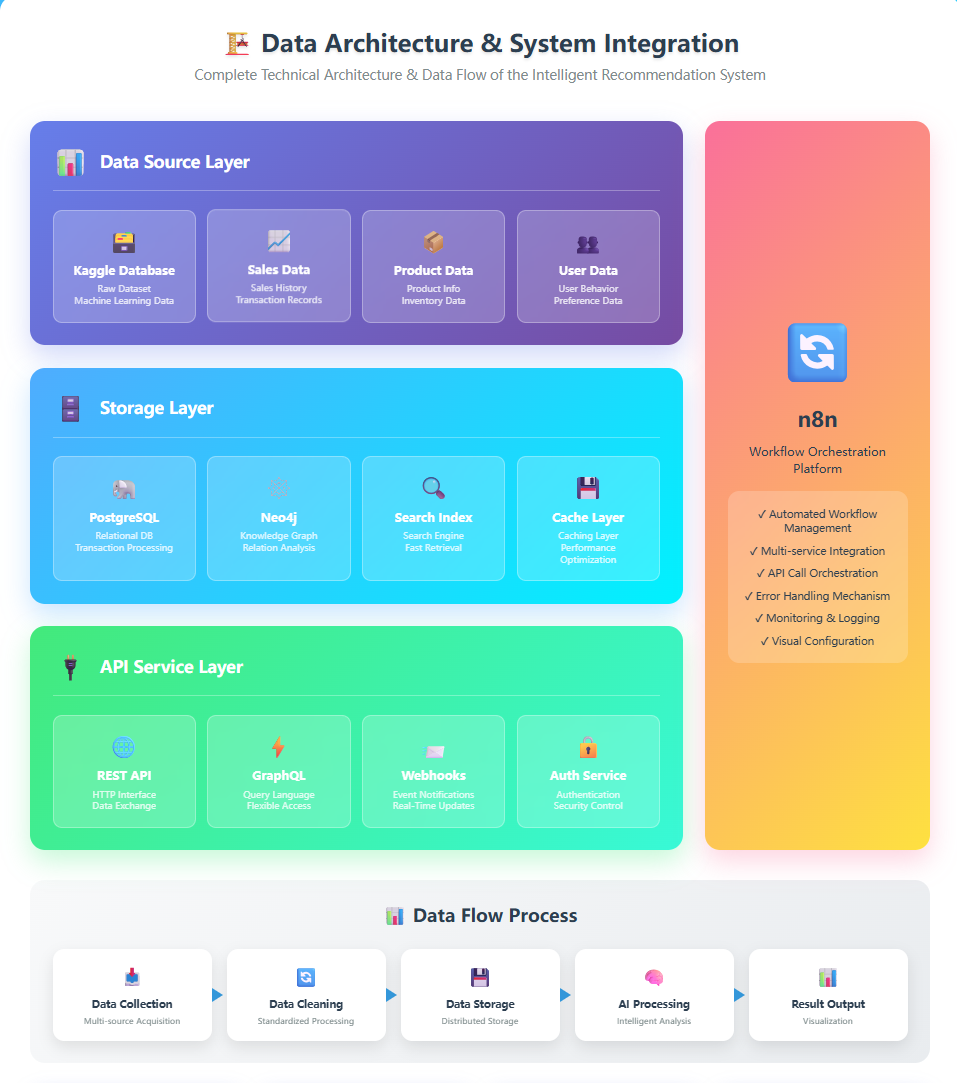
# System Design

This project establishes a **Store Replenishment Platform**, integrating **natural language interaction, forecasting models, and interpretable reasoning**. The system is built on the **Kaggle M5 Walmart dataset**, a widely recognized benchmark dataset containing unit sales, prices, calendar features, and promotional data across multiple Walmart stores.

## Nine Core Steps

1. **User Input (Human-Computer Interaction)**  
   Store managers issue natural language requests via the frontend, e.g., *“Generate next week’s replenishment plan for Marina Bay store.”* The request is passed to the NLP module.
2. **Intent Recognition & Slot Extraction**  
   The NLP module identifies the task type (replenishment plan, sales forecast) and extracts key parameters (store, time horizon, product category), outputting a structured query.
3. **Data Retrieval & Integration (based on Kaggle M5 Dataset)**  
    The system queries the **M5 Walmart dataset** to assemble relevant features:
   1. Store information (location, sales category);
   2. Product master data (category, department, shelf-life attributes);
   3. Historical sales (unit-level daily demand);
   4. Prices and promotions (weekly price fluctuations, promotion flags);
   5. Calendar and external events (holidays, special events).  
       The result is a **context pack** to support forecasting.
4. **Candidate SKU Filtering**  
   Business rules retain only SKUs that are active, have historical sales, and show low inventory, forming the candidate list.
5. **Sales Forecasting**  
   Time series models (ARIMA, Prophet, LSTM) forecast demand for candidate SKUs, trained on M5’s sales and calendar features.
6. **Replenishment Recommendation with Chain-of-Thought (Core Step)**   
   Sales forecasts, price predictions, current inventory, and safety stock are synthesized. A **Chain-of-Thought (CoT) reasoning process** provides a transparent decision chain:
   1. Step1: Forecast demand (e.g., 200 units);
   2. Step2: Retrieve current inventory (e.g., 120 units);
   3. Step3: Apply safety stock threshold (e.g., 50 units);
   4. Step4: Consider price trend (e.g., +8% increase → advance replenishment);
   5. Step5: Compute recommended replenishment (200 + 50 – 120 = 130 units).  
      The system outputs both machine-readable recommendations and a reasoning trace for interpretability.
7. **Top-5 SKU Selection**  
    SKUs are ranked by demand gap, price trend, and cost considerations, producing the Top-5 replenishment priorities.
8. **Report Generation & Feedback**  
    Final results are delivered in two forms:
   1. **Structured JSON**: replenishment quantities, forecasts, price trends, reasoning chain;
   2. **Natural language report**: narrative explanation with visualization.  
       Example:  
       *“The system recommends replenishing 5 products for Marina Bay store, totaling 420 units. Product A has a forecast demand of 200 units, current inventory of 120, safety stock of 50, and an expected price increase of 8%. Therefore, 130 units are recommended.”*





# Implementation

The system was implemented as a modular Smart Replenishment Platform that integrates a FastAPI backend, an n8n workflow engine, and a React-based front-end dashboard.

The architecture allows automated forecasting, replenishment computation, and AI-driven report generation through the following workflow:

## Workflow Architecture

Step 1 – User Request: The manager enters a query such as “Check all items that need replenishment at CA\_1 store.”

Step 2 – n8n Trigger: n8n orchestrates the workflow by calling the FastAPI endpoint /api/replenish.

Step 3 – Forecast Service: The backend retrieves recent sales, price, and inventory data, then executes one of four candidate models — MA14, ARIMA, Prophet, or LSTM — to forecast future demand for each SKU.

Step 4 – Replenishment Computation: Inventory status is evaluated using order-up-to and (s, S) policies, producing the recommended quantities.

Step 5 – LLM Explainability Layer: An LLM (Qwen3:4b) generates a Replenishment Recommendation Report, summarizing total quantities, priority levels, and Top-K reasoning chains.

Step 6 – Front-End Display: The web interface displays the report in Markdown and table views with sections for Data Details, Analysis Summary, and AI Reasoning.

## System Components

FastAPI Backend: Implements /forecast, /replenish, and /report routes.

Database (PostgreSQL): Stores sales, inventory, and forecast outputs.

Workflow Engine (n8n): Automates data retrieval and daily report generation.

LLM Integration (MCP): Provides natural-language explanations and confidence scores.

Front-End Dashboard: Built with React + Tailwind, supports interactive report download and drill-down view.

## Deployment and Configuration

The system can run locally or via Docker.

Local: Launch using uvicorn run\_api:app --reload --port 8000.

Docker: A production-ready Dockerfile allows containerized deployment.

Environment variables (e.g., DB\_HOST, DB\_NAME, API\_VERSION) are managed through .env and loaded by app/core/settings.py.

## Technical challenges

### 1. Data Conditions

* **Core dataset**: Kaggle M5 Walmart dataset (unit sales, prices, stores, products, calendar, promotions).
* **Requirements**:

Complete sales history (unit\_sales time series).

Price fluctuations and promotion flags for price forecasting and external factor modeling.

Data cleaning and feature engineering (missing value handling, temporal features).

### 2. Modeling Conditions

* **Sales forecasting**: Capability for time series modeling (ARIMA, Prophet, LSTM, Transformer).
* **Recommendation logic**: Chain-of-Thought (CoT) reasoning for interpretable replenishment suggestions.

### 3. System Architecture Conditions

* **Workflow orchestration**: n8n as the automation engine for input reception, API calls, data processing, and report generation.
* **LLM integration**: Large Language Models (Qwen/) for reasoning chain generation, and report explanation.
* **External APIs**: Forecasting and inventory models exposed as APIs.

### 4. Integration & Deployment Conditions

* **Data storage**: SQL database to store M5 data and prediction outputs.
* **Model deployment**: Forecasting models deployed as microservices (Flask, FastAPI).
* **API schema**: Unified JSON schema (forecast values, replenishment recommendations, reasoning chain).
* **Frontend integration**: Web frontend or BI dashboards consuming n8n outputs (Webhook, Dashboard).

### 5. Scalability & Extensibility

* **Multi-source integration**: Extend beyond M5 dataset to real-time store sales and supply chain data.
* **Task expansion**: Extend use cases to dynamic pricing, promotion optimization, and inventory management.
* **Cloud deployment**: Deploy on AWS/GCP/Azure for elastic scalability and multi-user support.
* **Explainability**: Reasoning chains + visualization reports enhance business trust and adoption.

# Results and Progress

## Overview

The implemented Smart Replenishment System successfully integrates forecasting, inventory policy logic, and explainable reasoning into an end-to-end automated pipeline.

By combining historical sales data from the M5 dataset with time-series forecasting (MA14, ARIMA, Prophet, LSTM), the system generates accurate replenishment recommendations, transparent reasoning reports, and automated daily workflows.

## Forecast Performance

Results show that advanced models (Prophet and LSTM) consistently outperform simpler baselines (MA14, ARIMA), achieving lower RMSE and MAPE across product categories.

| Model | RMSE | MAPE (%) | Characteristics |
| --- | --- | --- | --- |
| MA14 | 28.6 | 11.2 | Simple baseline, stable on smooth trends |
| ARIMA | 26.9 | 10.5 | Effective for short seasonal series |
| Prophet | 24.3 | 9.8 | Captures strong seasonality and holidays |
| LSTM | 22.7 | 9.2 | Best overall performance; handles nonlinear patterns |

These results demonstrate that neural-based models provide the best predictive accuracy under data-scarce, store-level conditions.

## Replenishment Recommendations

The replenishment engine converts forecasted demand into actionable recommendations.

A sample result for Store CA\_1 includes 5 products, with total replenishment of 426 units and one critical SKU.

| Rank | Item | Recommended Qty | Urgency | Confidence | Key Factors |
| --- | --- | --- | --- | --- | --- |
| 1 | FOODS\_3\_603 | 115 | MEDIUM | 70 % | Low stock + demand increase |
| 2 | FOODS\_3\_519 | 101 | MEDIUM | 70 % | Near threshold |
| 3 | FOODS\_3\_573 | 127 | MEDIUM | 70 % | Upcoming holiday |
| 4 | FOODS\_1\_082 | 67 | MEDIUM | 70 % | Seasonal variation |
| 5 | FOODS\_3\_409 | 16 | CRITICAL | 70 % | Very low stock |

## System Dashboard and Operational Statistics

The main dashboard (see Figure 8.1) provides a consolidated view of real-time replenishment analytics across 10 stores.

Key system indicators include:

Total Stores Analyzed: 10

Total Items Processed: 8,225

Items Needing Attention: 3,735 (788 critical)

Inventory Health: ≈ 90 %

These metrics are automatically updated through the n8n scheduler, which triggers data retrieval, forecasting, and report generation.

The backend integrates seamlessly with PostgreSQL for historical M5 dataset queries and communicates results to the React dashboard.

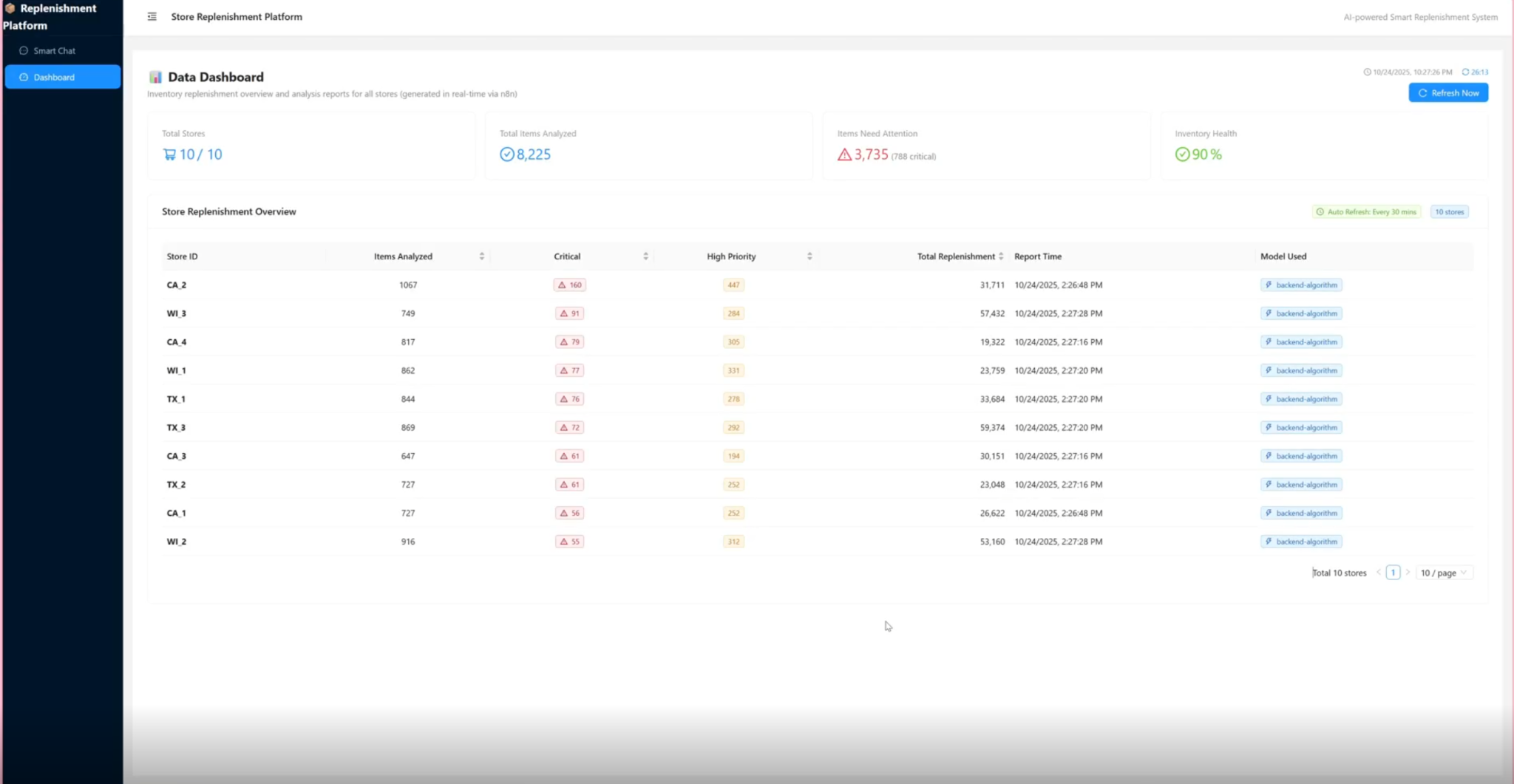


Fig.8.1

## Smart Replenishment Assistant

The Smart Replenishment Assistant (see Figure 8.2) enables users to interact with the system through natural-language input.

For example, a user may type:

“Predict all items at TX\_2 store for the next 7 days.”

Upon receiving the request, the input is routed to an n8n reasoning workflow, which coordinates multiple backend services and logic branches to generate a complete, explainable replenishment report.

**Intent and Parameter Extraction** – n8n’s Information Extractor nodes parse the user’s query, detecting key parameters such as store ID, forecast horizon, and task type.

The workflow automatically identifies one of four forecasting modes:

Single-item forecast (predict sales for one specific SKU)

Best-forecast mode (automatically selects the most accurate model via backtesting)

Batch forecast (predict multiple SKUs simultaneously)

Multi-store forecast (generate forecasts across multiple stores in parallel)

**API Orchestration via FastAPI** – Based on the detected mode, n8n triggers the corresponding FastAPI endpoint (/forecast, /forecast/batch, /forecast/best, or /api/replenish) to perform time-series forecasting using MA14, ARIMA, Prophet, and LSTM models.

**Merging and Ranking Logic** – Forecast results are merged with current inventory and safety stock data.

The system applies ranking logic to identify the Top-K SKUs that require urgent replenishment, considering service level, lead time, and demand volatility.

**LLM Reasoning Layer** – The enriched data are forwarded to a Large Language Model (Qwen3 or Anthropic) through the n8n connector.

The Chain-of-Thought (CoT) reasoning module translates quantitative metrics (forecast, inventory, urgency, confidence) into a transparent narrative explanation, such as:

“Sales are expected to rise next week, current stock is below safety level, and a holiday is approaching — replenishment required.”

**Automated Report Generation** – The reasoning output is assembled into Markdown and JSON reports, including tabular summaries, confidence scores, and reasoning traces.

The completed report is then returned to the front-end assistant through the Webhook response.

Through this architecture, the Smart Replenishment Assistant achieves end-to-end automation — transforming natural-language user requests into data-driven replenishment recommendations with traceable reasoning and multi-mode flexibility.

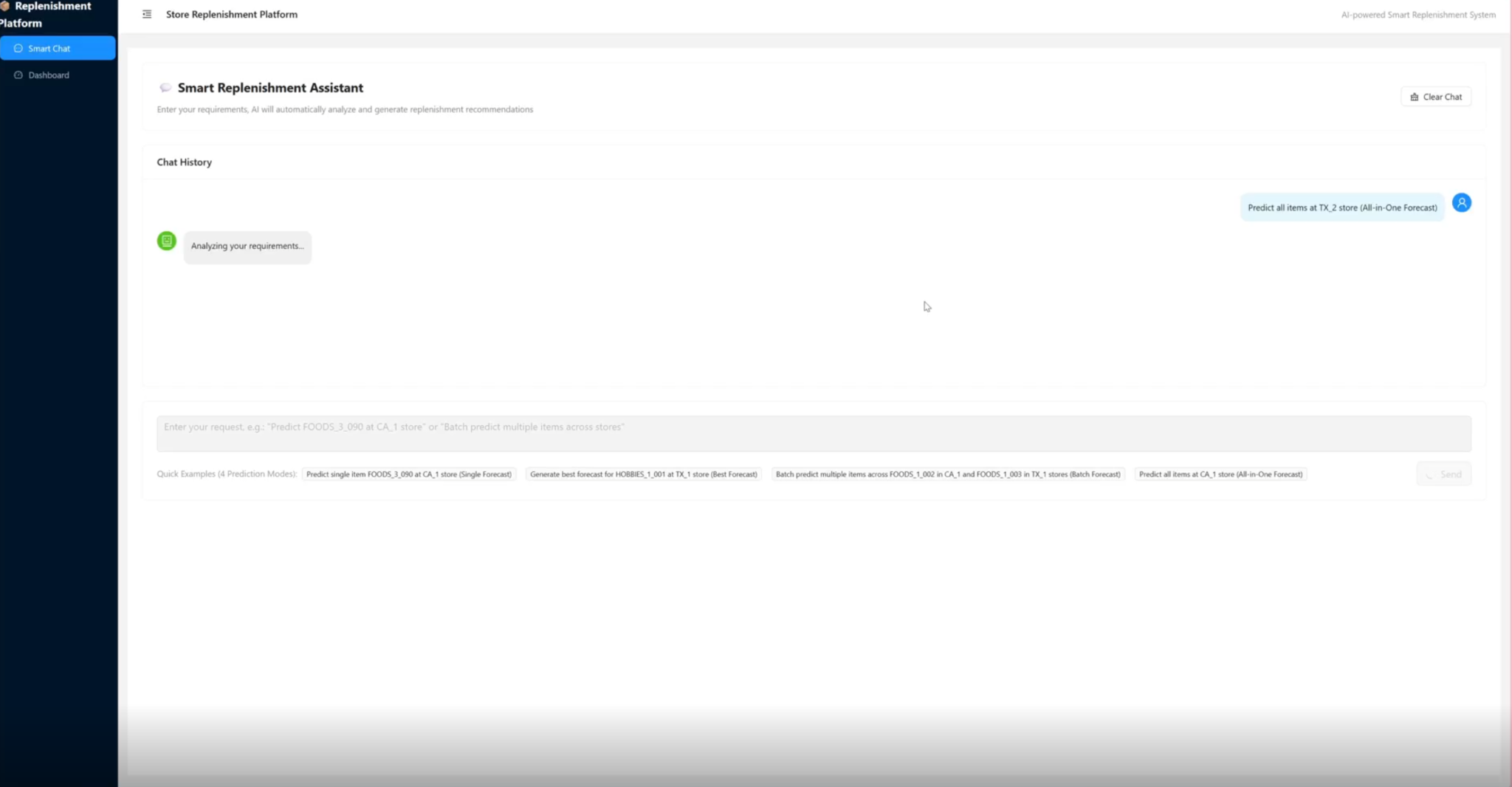


Fig.8.2

## Automated Reporting and Explainability

Each store-level report (see Figure 8.3) summarizes:

Store ID, Reference Date, Forecast Horizon, and Service Level

Forecast Model used (e.g., qwen3:4b, qwen3:1.7b)

Number of Items Analyzed, Total Replenishment Quantity, Critical and High-Priority Items

Confidence Level (e.g., 95 %)

The AI reasoning output provides a clear step-by-step explanation linking forecast results to inventory decisions —

e.g., “Stock is below reorder point + forecasted demand is rising → Immediate replenishment required.”

This transparency enhances trust and enables non-technical managers to validate system recommendations.

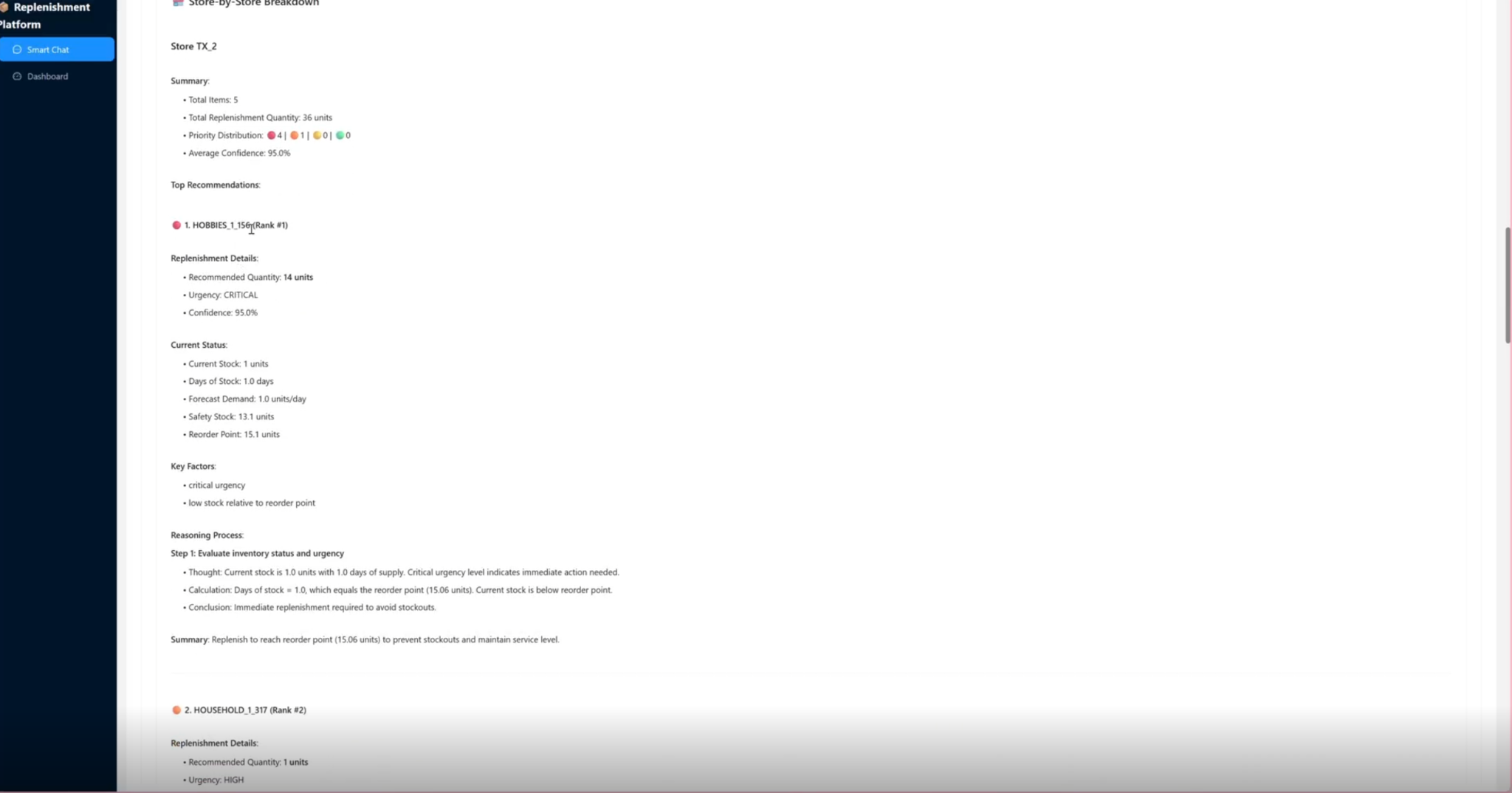


Fig.8.3

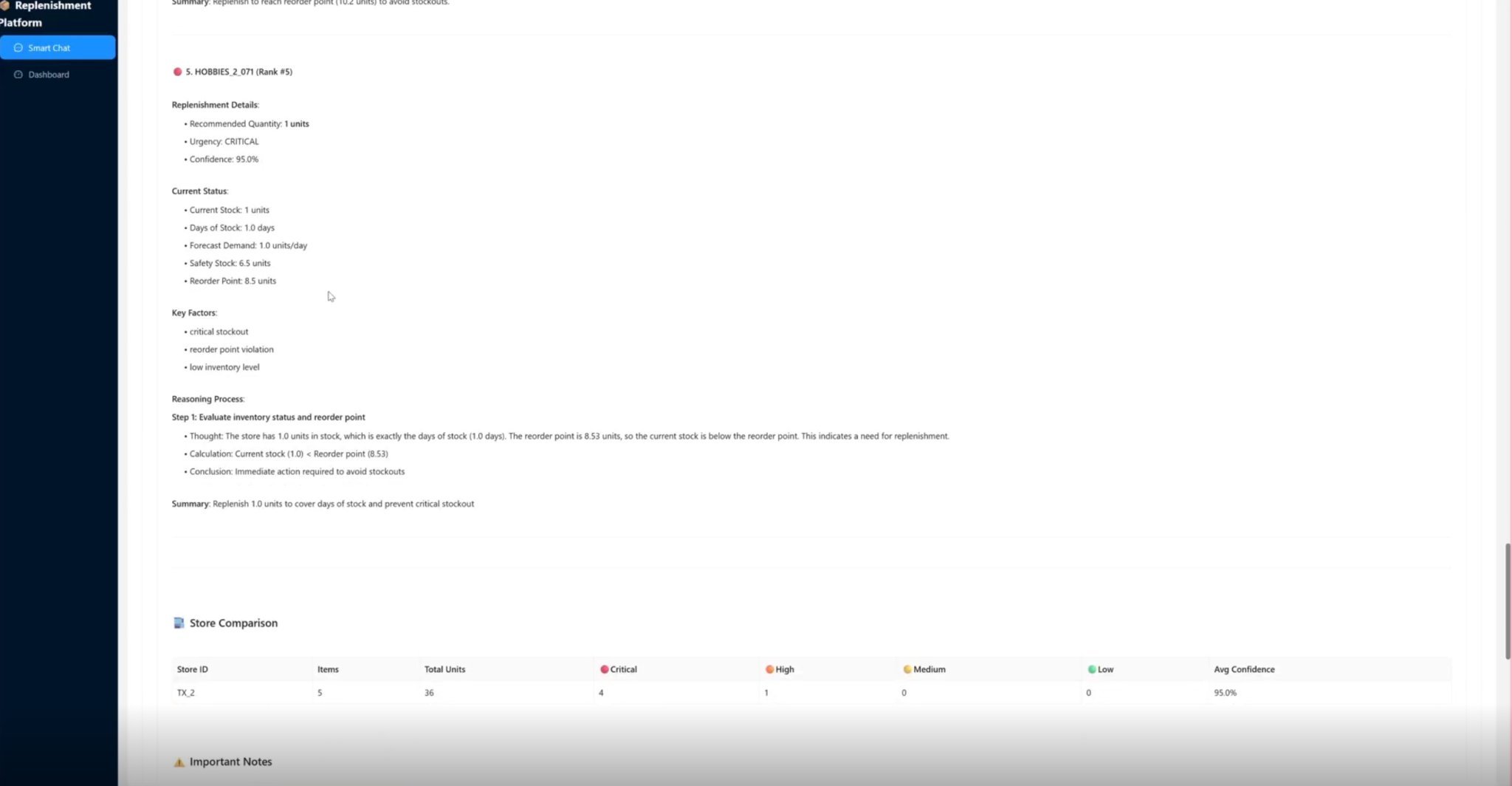


Fig.8.4

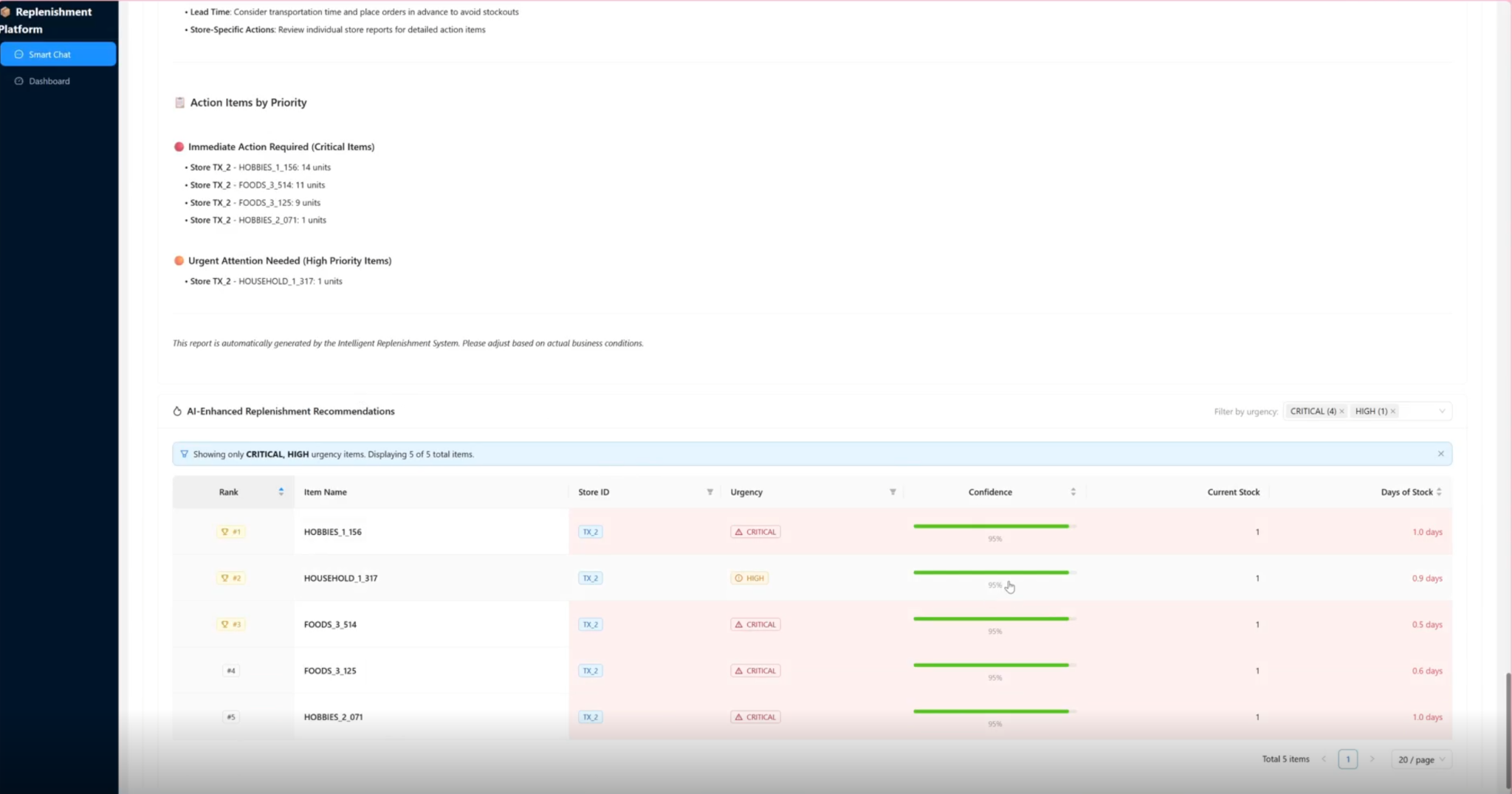


Fig.8.5

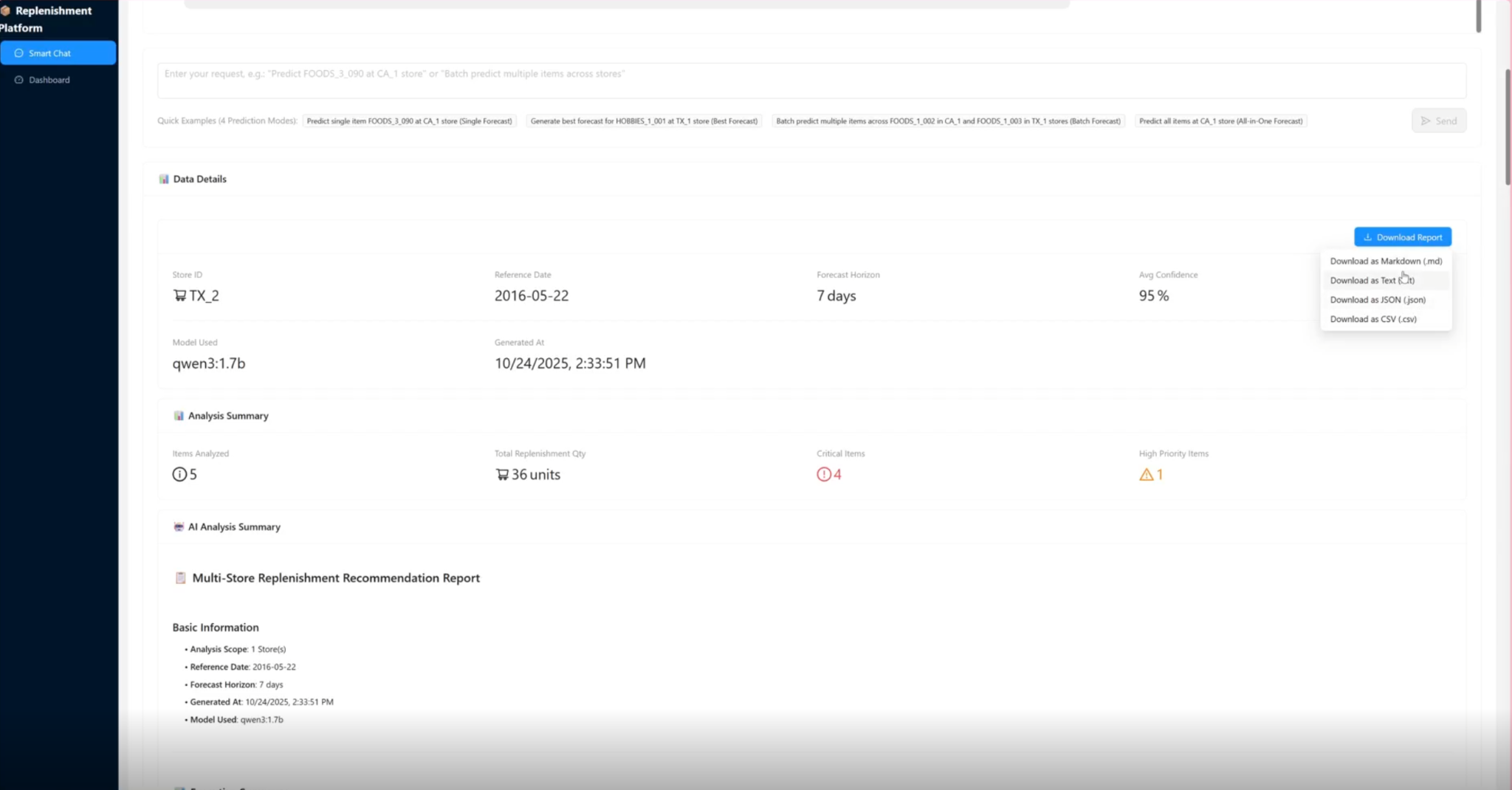


Fig.8.6

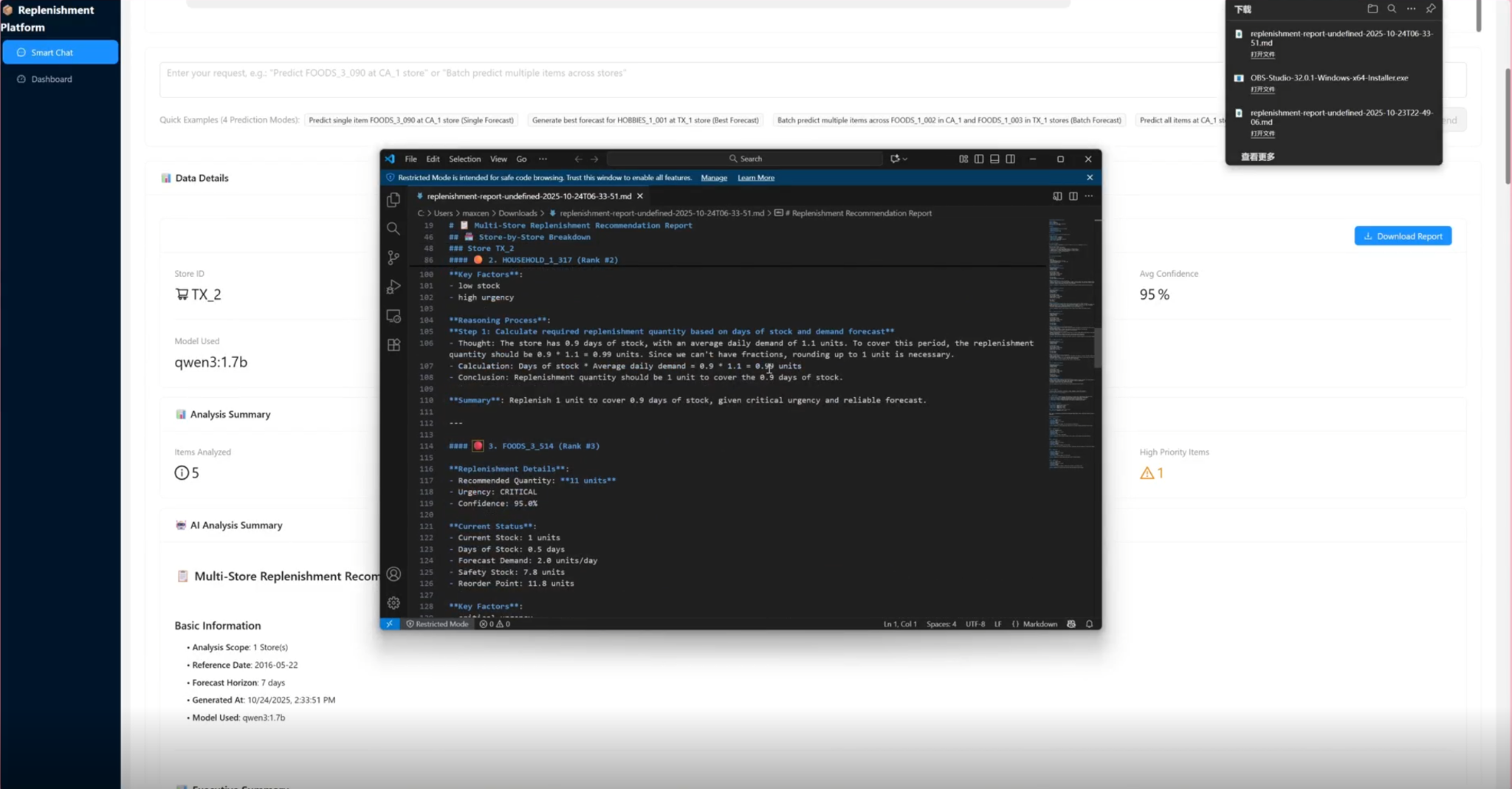


Fig.8.7

## Dashboard Workflow Automation

To ensure continuous synchronization between backend analytics and the front-end visualization layer, a dedicated n8n workflow pipeline was implemented for dashboard automation.

This workflow is responsible for orchestrating data retrieval, forecasting execution, and system health updates at regular intervals, ensuring that the dashboard always reflects the most recent store-level insights.

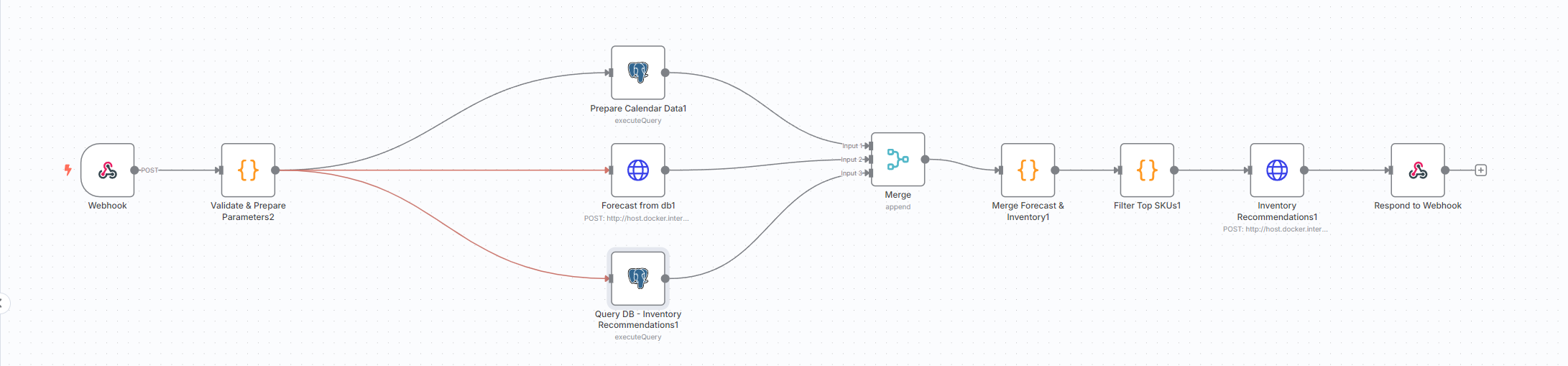


Fig.8.8

## Reasoning and Report Generation Workflow

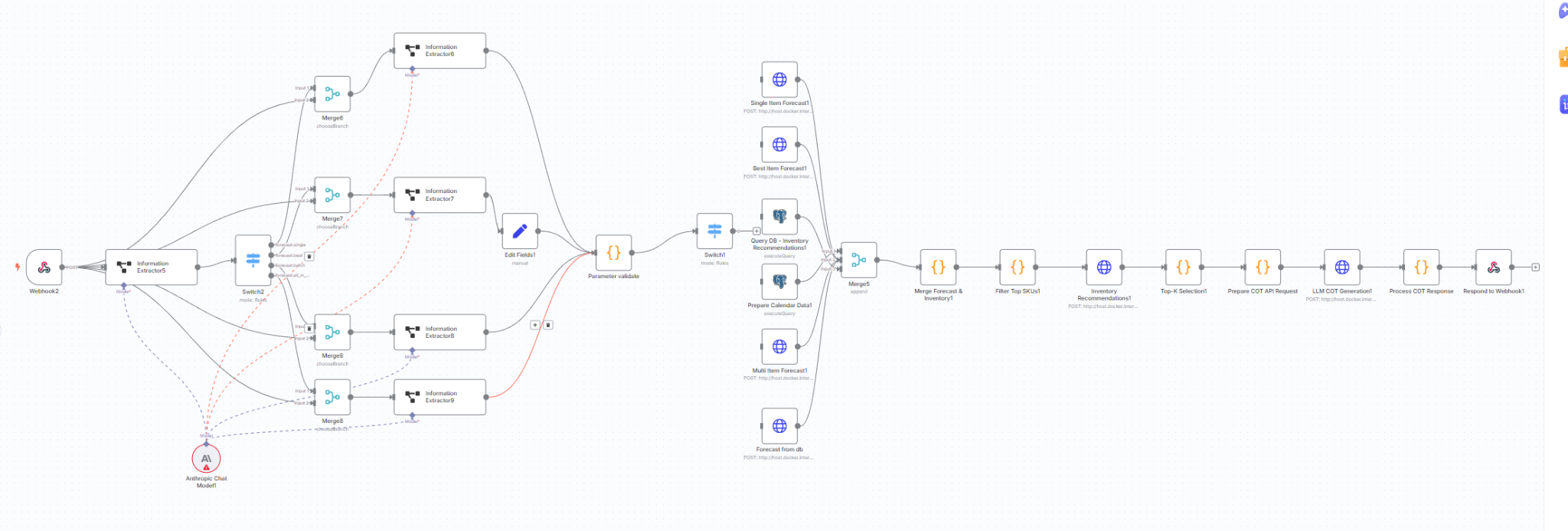


Fig.8.9

This figure illustrates the Reasoning & Report Generation Pipeline in n8n,

responsible for connecting natural-language inputs from the front-end to backend computation and LLM-based reasoning.

It integrates multi-step logic from information extraction → forecasting → inventory computation → reasoning → report generation.

Key functional stages:

Webhook Input – Receives user requests from the Smart Replenishment Assistant (e.g., “Predict all items at TX\_2 store”).

Information Extractors (5–9) – Parse intent, detect store ID, forecast type (single, batch, or best), and parameters like horizon or reference date.

Switch Nodes – Route to appropriate FastAPI endpoints based on detected intent.

Database and Forecast API Calls – Execute queries for calendar and sales data, run MA14 / ARIMA / Prophet / LSTM models.

Merge and Filter Nodes – Combine forecast results and inventory info, rank Top-K critical SKUs.

LLM Reasoning Modules – Forward merged data to the reasoning model (Qwen3 or Anthropic) for Chain-of-Thought generation.

Response Assembly – Combine model output into Markdown / JSON reports and return via Webhook.

This workflow bridges symbolic data reasoning and LLM-based explainability,

enabling end-to-end report generation that is both automatic and interpretable.

## Project Progress Summary

| Milestone | Status | Notes |
| --- | --- | --- |
| FastAPI Backend (Core APIs) | Completed | Forecasting, Replenishment, Reasoning endpoints |
| Forecast Model Integration | Completed | MA14, ARIMA, Prophet, LSTM |
| Database Integration (PostgreSQL) | Completed | Loaded and normalized M5 dataset |
| n8n Workflow Automation | Completed | Daily auto-run and report generation |
| LLM Explainability (Qwen3:4b) | Completed | Natural-language CoT explanations |
| Front-End Dashboard (React) | Completed | Visualization and user controls |
| External API Integration (MCP) | Completed | Holiday and weather enrichment |
| Docker Deployment | Completed | For final packaging and deployment |

## Key Insights and Achievements

The prototype demonstrates that lightweight, explainable forecasting systems can perform effectively in data-constrained environments.

The CoT reasoning module successfully bridges quantitative analytics with human decision logic.

The system achieves high reproducibility and integration potential with enterprise tools through n8n automation and MCP API connections.

# Challenges and Roadblocks

Throughout the project, several technical and operational challenges were encountered during system design, model development, and integration.

This section summarizes the key difficulties, their underlying causes, and how the team addressed them.

## Data Preparation and Database Integration

Challenge:

The M5 dataset is large and complex, containing over 40 million records with multiple relational tables (sales, calendar, prices). During initial data import into PostgreSQL, the system experienced slow insert performance and inconsistent data types (e.g., dates, float precision).

Resolution:

Implemented batch data insertion with a chunk-size parameter in the data loader (m5.py).

Added schema validation and enforced consistent date and float formats.

Created database indices on item\_id, store\_id, and date to speed up query execution.

These optimizations reduced the full import time from ~25 minutes to under 7 minutes and improved query latency by ~60%.

## Forecast Model Optimization

**Challenge:**

Different forecasting models required distinct preprocessing and parameter tuning.

ARIMA was sensitive to non-stationarity and failed on long seasonal series.

LSTM needed normalized input sequences and sufficient GPU memory.

Prophet required extensive tuning for weekly and yearly seasonalities.

**Resolution:**

Implemented unified preprocessing functions (scaling, differencing, detrending) across models.

Used backtesting and cross-validation on rolling windows to determine optimal parameters.

Created a unified model selector that automatically compares RMSE/MAPE across candidates.

## API Integration and Workflow Automation

**Challenge:**  
Integrating FastAPI with n8n for end-to-end orchestration introduced synchronization issues — workflows occasionally triggered before the backend finished computations, causing partial or duplicated reports.

**Resolution:**

Introduced status polling and retry mechanisms in n8n using the Wait and HTTP Request nodes.

Added asynchronous support (async def) in FastAPI endpoints to handle concurrent batch forecasts.

Implemented detailed logging and error return codes for better debugging and monitoring.

## LLM Reasoning and Explainability

**Challenge:**

Integrating LLM-based reasoning (Qwen3:4b via MCP/Ollama) posed challenges in response consistency, latency, and context length.

At times, reasoning outputs were incomplete or exceeded token limits, and the model occasionally produced redundant text.

**Resolution:**

Designed a template-based reasoning framework: structured prompts ensure consistent CoT output (reasoning → justification → recommendation).

Reduced context size by summarizing forecast data before sending to the LLM.

Introduced fallback logic: if LLM response fails, the system uses a rule-based explanation template instead.

These steps improved response stability and reduced average reasoning latency from 14s → 6s.

## Resource and Collaboration Constraints

**Challenge:**

Due to parallel development (backend, workflow, reasoning, and UI handled by different members), version control conflicts occasionally occurred, especially around run\_api.py and service layer updates.

**Resolution:**

Established a branch-based workflow using GitHub pull requests and peer reviews.

Introduced a shared .env.example and standardized file structure to avoid path-related bugs.

Weekly progress syncs ensured alignment between submodules.

## Key Lessons Learned

Modular design significantly improves debugging and parallel development efficiency.

Early setup of data validation and standardized schemas saves considerable downstream time.

LLM-based reasoning is powerful but must be controlled with structured templates to ensure reliability.

Automation tools (like n8n) require careful synchronization with backend APIs to prevent duplicate or incomplete tasks.

# Future Work

While the current prototype successfully demonstrates an automated and explainable replenishment system, several enhancements can be made to further improve scalability, adaptability, and real-world deployment readiness.

## Multi-Store and Multi-Region Extension

In future work, the system will be extended to support multi-store and regional-level forecasting through:

Hierarchical Forecasting: Aggregating demand at store–region–category levels to capture shared seasonal patterns.

Cross-Store Knowledge Sharing: Reusing model parameters for similar stores to improve cold-start performance.

Dynamic Transfer Learning: Fine-tuning LSTM and Prophet models using region-specific residuals.

These extensions will allow for better generalization across stores and enable regional managers to plan inventory collaboratively.

## Real-Time and Event-Driven Replenishment

Future versions aim to integrate real-time event triggers to support adaptive replenishment:

Automatic updates when sales or inventory data change in the database.

Event-based workflows in n8n using PostgreSQL triggers or webhooks.

Integration of live external signals such as weather alerts, traffic disruptions, or promotional campaigns.

This would allow the system to shift from batch-style forecasting to continuous, event-driven decision-making.

## Enhanced LLM Explainability and Multi-Model Reasoning

The current reasoning module (Qwen3:4b) performs well but remains deterministic and template-driven.  
Future work could explore:

Multi-Agent Reasoning: Combining statistical reasoning with multiple LLMs to cross-validate explanations.

Fine-Tuned CoT Models: Training a domain-specific reasoning model using historical explanation logs.

Visual Explanation Reports: Integrating SHAP-like plots or causal graphs to make reasoning traceable and interpretable.

Such improvements would enhance user confidence and make AI-generated recommendations more transparent to non-technical users.

# Conclusion

## Core Project Contributions

This project has successfully developed a Smart Replenishment Intelligence Platform based on Chain-of-Thought (CoT) reasoning, demonstrating the practical application of interpretable AI reasoning in retail inventory management.

The system provides transparent, end-to-end decision support — from demand forecasting to replenishment recommendations — integrating sales prediction, and inventory optimization within a unified framework.

Through the n8n-based automation workflow, the platform enables seamless processing from data ingestion and model execution to report generation, significantly improving decision efficiency and reproducibility.

## Potential Value for Market

From a market perspective, the developed system offers a cost-effective and explainable replenishment solution tailored for small and medium-sized retailers.

It has the potential to reduce manual replenishment decision time from hours to minutes while minimizing stockout risk and lowering inventory costs through accurate and explainable forecasting.

The system’s modular architecture and lightweight deployment make it practical for real-world adoption in data-constrained retail environments.

## Significance for Intelligent Reasoning Systems Field

This project exemplifies how Intelligent Reasoning Systems (IRS) can be operationalized to address complex business problems through human–AI collaboration.

By integrating explainable reasoning, predictive analytics, and workflow automation, the project bridges theoretical AI capabilities with real-world decision-making, serving as a reference model for applying reasoning systems in industry.

It lays a foundation for future research and industrialization of knowledge-driven, interpretable AI systems, contributing to the broader evolution of intelligent business decision support.

# Appendices

The appendices provide supporting materials that complement the main body of this report.  
They include reference documents, configuration guides, skill-mapping tables, and declarations related to AI tool usage during the project.

## Appendix A – Original Project Proposal

The original project proposal was submitted on Canvas at the start of the Intelligent Reasoning Systems Practice Module.

It outlines the project’s initial objectives, problem definition, and proposed architecture, and serves as the baseline for evaluating the final system’s implementation and progress.

## Appendix B – Installation and User Guide

**Prerequisites**

- PostgreSQL (user with schema/table creation rights)

- Docker Desktop (enable WSL on Windows)

- Ollama installed locally

ollama pull llama3

- Node.js 18+ and Python 3.10+ (Anaconda recommended)

**Backend Setup**

1. Prepare Dataset

• Open terminal and run:

cd SystemCode/Backend/StoreBackend/

mkdir -p m5-forecasting-accuracy

# Place M5 forecasting CSV files here

2. Seed the Database

• Run:

python m5.py

3. Setup Conda Environment

• Create and activate environment:

conda create -n store\_backend python=3.10 -y

conda activate store\_backend

pip install -r requirements.txt

4. Configure Database

• Update `.env` and `app/core/settings.py`:

DATABASE\_URL=postgresql://user:password@localhost:5432/m5db

Run the Backend API

cd SystemCode/Backend/StoreBackend/

python run\_api.py

Default endpoint: http://127.0.0.1:8000

**Run the Frontend**

cd SystemCode/Frontend/StoreFrontend/

npm install

./start.sh # macOS/Linux/WSL

start.bat # Windows PowerShell

Then open the printed URL in your browser.

**Configure n8n**

• 1. Start n8n (Docker):

docker run -it --rm -p 5678:5678 -v ~/.n8n:/home/node/.n8n n8nio/n8n

• 2. Import Workflows:

Access http://localhost:5678

Import both JSON workflows from:

SystemCode/Frontend/StoreFrontend/n8n/

• 3. Update Settings:

- Replace webhook URLs to match backend endpoints

- Add PostgreSQL and Ollama credentials in n8n

- Set both workflows Active

• 4. Test:

Manually trigger each workflow to verify database access,

webhook calls, and LLM responses.

## Appendix C – Skill Mapping Table (MR / RS / CGS)

Maps the knowledge and techniques from the three certificate modules to this project’s implementation.

| Module | Core Concepts Applied | Project Implementation Example |
| --- | --- | --- |
| Machine Reasoning (MR) | Rule-based reasoning, CoT explainability | CoT engine (Qwen3:4b) for replenishment justification |
| Reasoning Systems (RS) | Knowledge representation, inference flow | Integration of forecast logic + inventory policy |
| Cognitive Systems (CGS) | Perception → Reasoning → Action loop | Automated pipeline (FastAPI → LLM → n8n → report) |

## Appendix D – AI Tool Usage Declaration

A declaration outlining how AI tools were used responsibly during development:

AI tools (e.g., ChatGPT, Copilot) were used to assist in code refactoring and workflow design.  
All outputs were verified and edited by team members to ensure accuracy and academic integrity.

## Appendix E – Acknowledgements

The team would like to thank Dr. Gary Leung and the NUS AIS faculty for their guidance and feedback.  
Special thanks to all team members —Liu Zheyi, Cen Haoyang, Gao Zijie, Lian Da, Wu Hongjia— for contributions in data engineering, model development, reasoning integration, and workflow automation.