Capstone Project: Agentic AI for Inventory Optimization

LLM-Agent Inventory Replenishment Optimization with Forecasting, Demand Simulation, and Policy Optimization

Problem statement

Retailers and distributors routinely face uncertain demand and variable lead times. Static replenishment rules (fixed reorder points / fixed safety stock) often cause either excess inventory (high holding cost) or stockouts (lost sales + service failures). The goal is to build a decision-support system that forecasts demand, simulates uncertainty, optimizes replenishment plans, and exposes the results in a cloud/web app.

Project objective

Build a practical "inventory control tower" that recommends replenishment parameters (e.g., **(s, Q)** or **(R, S)** policies) that **minimize total expected cost** while meeting a **target service level**, using **three AI agents**:

1. Forecast Agent
2. Demand Simulation Agent
3. Optimization Agent

Data Sources

Primary:

- **M5 (Walmart) sales + calendar + prices** (multi-SKU daily demand)

Optional:

- Weather (NOAA CSV/API) to create demand shocks or lead-time risk scenarios
- Google Trends CSV export for "demand sensing" signal

System design: 3 AI agents shared architecture

All three agents will be LLM-driven, that call deterministic tools (ML model inference, simulator, optimizer) and write decisions to a shared state.
LLM will be used for orchestration + explanation; all numeric outputs will come from ML/simulation/optimization functions.

Agent 1: Forecast Agent (LLM + ML method)

Purpose: Produce demand forecasts with uncertainty for each SKU.

Functions it can call:

- **train_forecast_model()** (e.g., LightGBM/XGBoost)
- **predict_quantiles()** → outputs mean, P50, P90 (or intervals)
- **explain_forecast_drivers()** → tells which features mattered (promotions, seasonality, price, etc.)

The **Forecast Agent** predicts future demand for each SKU using historical sales and signals like seasonality, calendar effects, and price/promotions. It outputs both a typical forecast and an uncertainty range (e.g., P50/P90), which the Simulation and Optimization Agents use to test and choose inventory policies that meet the service target at low cost.

Agent 2: Demand Simulation Agent (LLM + simulation method)

Purpose: Generate demand scenarios and simulate inventory outcomes under uncertainty.

Functions it can call:

- **sample_demand_paths(forecast_quantiles, n_sims)** → creates many demand scenarios (low/typical/high)
- **simulate_inventory(policy_params, demand_paths, lead_time_dist, costs)**
- **summarize_kpis()** → service level, stockouts, cost breakdown, inventory turns

The **Simulation Agent** runs many "what-if" inventory simulations using the forecast uncertainty and lead-time variability, then reports KPIs like service level, stockouts, and total cost so policies can be compared and improved.

Agent 3: Optimization Agent (LLM + optimization method)

Purpose: Find policy parameters that minimize expected cost subject to service constraints.

It tries different settings of the following policies:

- **(s, Q)**: reorder point s, order quantity Q
- or **(R, S)**: review every R days, order up to S

Functions it can call:

- **evaluate_policy(s, Q) or evaluate_policy(R, S)** → runs simulation pipeline for a candidate policy
- **search_policy_space()** → smart search (Optuna/Bayesian optimization)
- **generate_recommendation()** (LLM writes final recommendation)

The **Optimization Agent** searches for the best inventory policy parameters (e.g., (s, Q) or (R, S)) by repeatedly evaluating candidates through the Simulation Agent, then selecting the lowest-cost policy that still meets the required service target.

MCP connection

MCP will be used to pass demand forecasts and uncertainty estimates as structured context between agents. Simulation outcomes such as service level and cost distributions will be shared across agents via MCP. Optimized replenishment policy parameters ((s, Q) or (R, S)) will be updated in agent context using MCP. All agent context updates and decisions will be logged through MCP for traceability and reproducibility.

Evaluation

Splitting the data like a real deployment

Using the first 4 years to build the system (train models, tune policies).
Using the last 1 year as "new/unseen" test data to evaluate performance.

Evaluating forecasting on the 1-year test set

Measuring how accurate forecasts are using MAE/RMSE.
Checking if the forecast "range" is reliable (interval coverage/calibration).

Evaluate the full inventory system on the 1-year test set (main result)

Running the full pipeline: forecast → simulation → optimization → policy output.
Comparing against simple baselines like EOQ + fixed safety stock.

Deliverables (web app)

User selects SKUs + service target → Forecast Agent produces quantiles → Simulation Agent estimates KPI distributions → Optimization Agent searches policy space → dashboard shows recommended policy+ expected cost/service.

1. **Dashboard**

- Top SKUs by predicted stockout risk
- Current recommended policy vs baseline
- Cost and service KPIs

2. **SKU detail**

- Forecast chart with uncertainty bands
- Simulated inventory trajectory
- Policy recommendation (s, Q) with explanation

3. **Scenario tab**

- sliders/toggles: service target, lead time variability, demand spike
- “Re-optimize” button → runs Optimization Agent

Required Tech Stack

- Backend: FastAPI (agent endpoints + model inference)
- Data Analysis: Pandas, Numpy
- AI/ML: Scikit-Learn, Langchain, LLMs (OpenAI or Google Gemini or Groq)
- Frontend: Streamlit