SDP 2025

User Design Document

Project Name	Reinforcement Learning on Sales & Distributions
Faculty Supervisor Name	Mr. Samir Mammadov
Team Lead Name & ID	Laman Panakhova 16882 Section1 Contribution
Team member #1 Name & ID What sections did this person contribute to?	Nargiz Aghayeva 16042 Section2 Contribution
Team member #2 Name & ID What sections did this person contribute to?	Aysu Azammadova 16113 Section1 Contribution
Team member #3 Name & ID What sections did this person contribute to?	Leyla Eynullazada 18033 Section2 Contribution

Guidelines:

1. In Section 1 you should create a journey map in Figma, for **each of** the **four** most important use-cases from the research document. Do not worry about trivial use cases like user sign-up and user password reset.
2. In Section 2 (Database Design), discuss the data model you propose to support your user journey maps. The database design may be relational (using ERDs) or document storage (using document collections)
3. Page Count: 15 Pages Maximum

Note: This project is not a mobile or web application. *It is a* research-focused simulation platform for testing RL models. Therefore, the Figma "journey maps" do not represent classical UI flows (sign-up, search, payment, etc.). Instead, they represent agent interaction cycles**,** data processing steps**,** and research pipelines**.**

Section 1 User Journey Maps

HERE IS AN EXAMPLE

Use-case 1: Book a flight on the app

- Persona: Alex, 35, frequent traveler, wants a quick booking process.
- Goal: Book a round-trip flight.

Figma Implementation:

1. Canvas Setup: Create a horizontal timeline in Figma or FigJam with stages (Search Flights, Select Flight, Enter Details, Confirm Booking).
2. Visual Elements: Use rectangles for stages, sticky notes for actions (e.g., "Enters destination"), and a line graph for emotions (e.g., frustrated at slow search).
3. Touchpoints: Add icons for app screens, email confirmations, or payment gateways.
4. Pain Points: Note issues like "Slow flight searches due to unoptimized database queries."
5. Collaboration: Share the file with the team; developers annotate database needs (e.g., "Index flight data for faster queries").
6. Prototype: Link stages to wireframes (e.g., click "Search Flights" to view a search results screen).

User Journey Maps (Based on Our RL Research System)

For our project, the User Journey Map doesn't follow the typical user flow you'd see in mobile or web apps. Instead, it focuses on how a researcher interacts with the RL experimentation platform throughout the entire process, starting from preparing their data all the way to evaluating the results. In Figma, we've laid this out as a horizontal timeline that captures the different stages of the process: 1. Data Preparation, 2. Parameter Setup, 3. Simulation Execution, 4. Agent Training, 5. Logging & Visualization, 6. Analysis of KPIs.

Each stage is represented by a simple block, and we use sticky notes to describe what the researcher does at each step. For example, some actions might include things like "uploads cleaned sales data," "sets episode length," or "starts training run." Instead of focusing on emotions, we've highlighted research pain points in the process, like slow training iterations, unstable hyperparameters, or high variance in rewards.

The touchpoints along the journey include the Jupyter environment, simulation logs, MongoDB storage where the research data is kept, and the dashboards used for visualizing results. Developers can also add notes directly inside Figma, explaining issues like where logging might overload storage or where preprocessing pipelines could be improved.

We've also added links to prototype samples, such as notebooks or visual mock-ups, that show how the researcher interacts with the experiment results. The idea is that the User Journey Map

helps us better understand how a researcher works through the RL experimentation lifecycle, making sure each step connects smoothly with the next.

Use Case 1 — RL for Stockout / Overstock Management

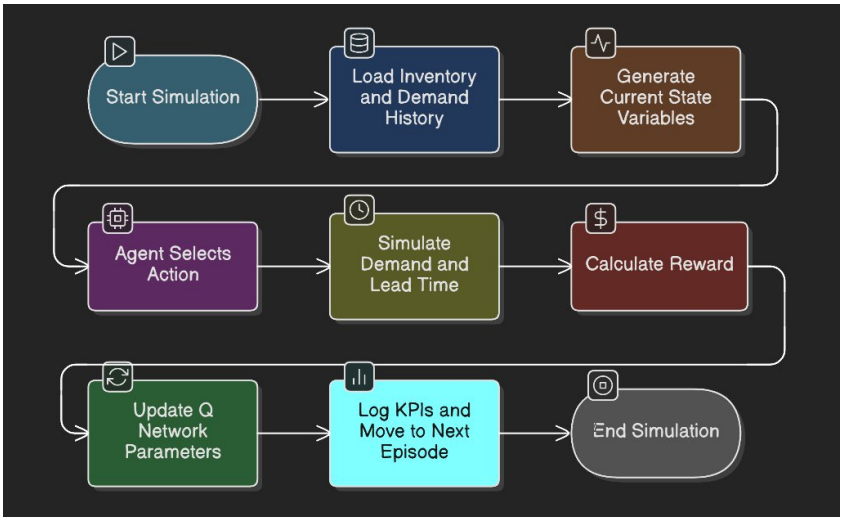
User: Supply Chain Analyst / Researcher
Goal: Needs to run RL simulations to minimize stockouts and holding costs.

State: current inventory level, forecast demand, lead time, cost parameters

Action: order quantity

Reward: penalties for stockout, holding cost, service-level achievement

Outcome: RL agent learns dynamic ordering policy that balances service level and cost.



Use Case 2 — RL for Competitive Dynamic Pricing

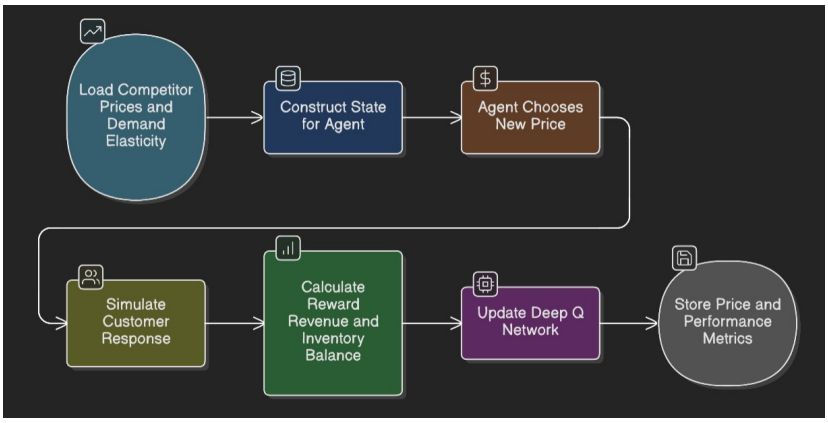
User: Pricing Analyst / Revenue Manager
Goal: Learn optimal prices in competitive markets where competitors' pricing actions influence demand.

State: competitor prices, demand signals, inventory, seasonality

Action: price point selection

Reward: profit, market share, long-term customer value

Outcome: Adaptive pricing strategy that responds to real-time market shifts.



Use Case 3 — RL for New Product Recommendation / Promotion

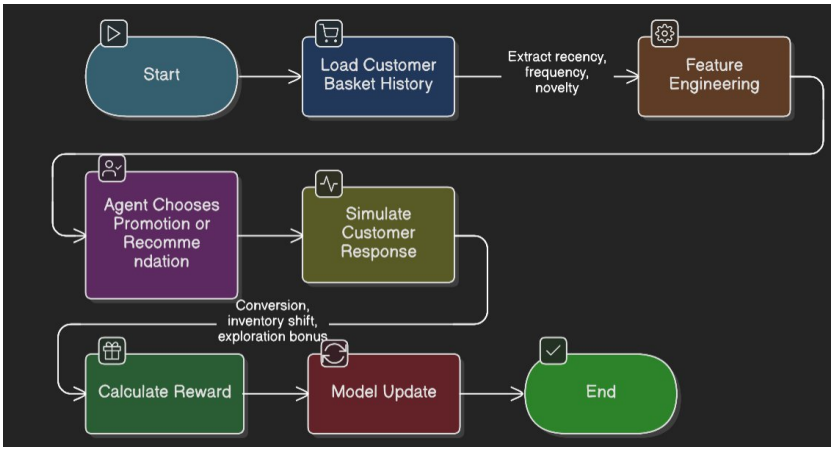
User: Marketing Analyst / Growth Manager
Goal: Optimize personalized recommendations or promotions for new products to maximize conversion and long-term retention.

State: user profile, browsing/purchase history, product attributes

Action: promotion or recommendation choice

Reward: conversion, engagement, downstream purchases

Outcome: RL-driven personalization that improves adoption of new products.



Use Case 4 — RL for Supply Allocation with Limited Supply

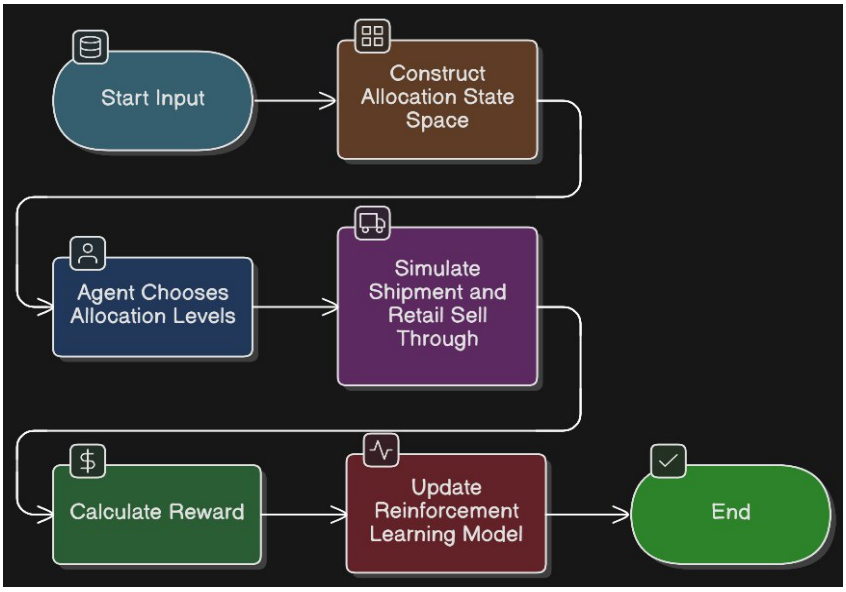
User: Supply Planner / Operations Researcher
Goal: Use reinforcement learning to allocate constrained supply across multiple regions, channels, or customer segments to maximize service levels and business impact.

State: available supply, forecasted demand by region/channel, priority rules, capacity constraints

Action: allocation vector (how much to assign to each region or channel)

Reward: service-level improvement, revenue maximization, penalty for unmet high-priority demand

Outcome: RL agent learns an adaptive allocation policy that optimizes limited supply distribution under fluctuating demand conditions.



Section 2: Database Design

In this section there are two choices: either choose a relational database design or a non-relational database design. In the rationale section, explain your reasoning.

Rationale

Here you should explain why you selected a Relational or a Non-Relational Database.

Relational Database Design

1. **Schema Structure**
    a. Entity-Relationship Diagrams (ERDs) for Figma Use Cases
    b. Tables and Relationships (e.g., Users, Roles, Transactions)

    c. Normalization to Ensure Data Consistency
2. **Query Optimization**
    a. Indexing for Fast Retrieval in Figma-Defined Flows (e.g., Search Queries)
    b. Joins and Aggregations for Data Display in UI
3. **Use Case Alignment**
    a. Supporting Structured Data Needs (e.g., Authentication, Reporting)
    b. Examples from Figma Prototypes (e.g., SQL Queries for User Profiles)

Non-Relational Database Design

4. **Schema Structure**
    a. Flexible Data Models (e.g., JSON Documents, Key-Value Stores)
    b. Collections for Dynamic Figma Use Cases (e.g., Product Catalogs, User Preferences)
    c. Denormalization for Performance in Unstructured Data
5. **Query Optimization**
    a. Aggregation Pipelines for Complex Data Retrieval
    b. Caching for Real-Time Interactions in Figma Prototypes
6. **Use Case Alignment**
    a. Supporting Scalable, Flexible Data (e.g., Real-Time Notifications, Analytics)
    b. Examples from Figma Journey Maps (e.g., MongoDB for User Activity Logs)

Research Database Design Overview

For our system, since it's more of a research platform than a transactional application, we don't use a traditional relational database. Instead, we go with a flexible, document-based model that works well for storing things like simulation parameters, training logs, experiment metadata, and RL outputs. The reason we don't use a relational database is that RL experiments naturally produce data that's complex and constantly changing. For example, there are sequences of states, actions, rewards, hyperparameters, and environment responses, and it's hard to fit all of these into fixed relational tables. A MongoDB-style design fits better because it lets us store data in a way that's more flexible, without needing to redesign the schema every time we add new data or modify the structure.

Since our system doesn't have a user interface, no authentication, and no business transactions, using a relational model just doesn't make sense. Instead, we organize the database into several main collections. A) Experiments collection: This holds high-level metadata about the experiment, like the algorithm type, hyperparameters, start/end times, and notes. B) Simulation_states collection: This stores the state vectors created at each step in the environment and links back to the corresponding experiment. C) Agent_actions collection: This one holds the RL agent's actions, the values of those actions, and the Q-value vectors. D) Rewards collection: This stores the reward signals and derived metrics like service levels, stockout counts, and costs.

We also have some domain-specific collections, like inventory_snapshots, pricing_logs, and recommendation_logs, which store observations specific to inventory, pricing, and recommendation experiments. All of these collections together form a kind of research journal that tracks everything from the initial experiment parameters to the final performance metrics.

In our conceptual database diagram, the experiments collection is at the center. It links down to the simulation_states collection, which then connects to agent_actions and rewards. We also have lateral links to domain-specific snapshots, like the inventory logs. This layout reflects the hierarchical and step-by-step nature of RL, where each experiment can generate thousands of states, actions, and rewards.

We also have several additional diagrams that explain how the whole system works.

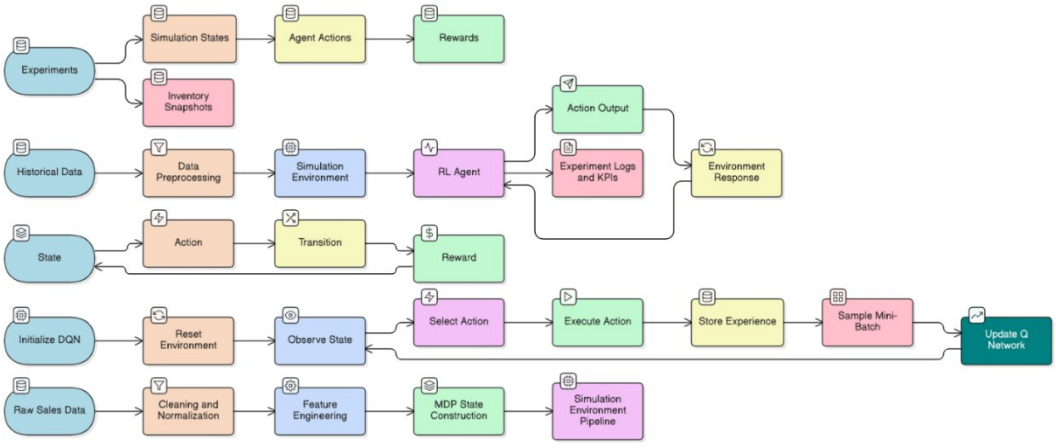
The System Architecture Diagram shows how historical datasets flow through preprocessing and into a simulation environment (like Gymnasium), where DQN-based agents generate actions, receive rewards, and store experiment logs and KPIs.

The Markov Decision Process (MDP) Diagram breaks down the math of the environment, showing how states (like demand, stock levels, lead times, and prices) interact with actions (like reorder or price changes), stochastic transitions, and reward calculations.

The RL Training Loop Diagram explains the iterative process of training the DQN, including state observation, action selection with the epsilon-greedy policy, experience storage, batch sampling, and Q-network updates.

Finally, the Data Pipeline Diagram describes how raw sales and inventory data are cleaned, normalized, turned into MDP-compatible state vectors, and fed into the simulation environment.

All of these diagrams and explanations work together to paint a complete picture of how our RL research pipeline is structured and how data flows through the system.



Experiment Workflow and Architecture Diagrams

1. Researcher Journey Map Diagram (Experiment Lifecycle)
Shows the full workflow a researcher follows, from loading datasets and setting parameters to running simulations, training the agent, and reviewing results. Highlights key touchpoints like Jupyter notebooks, simulation logs, and MongoDB. Also marks pain points such as unstable training or long run times.

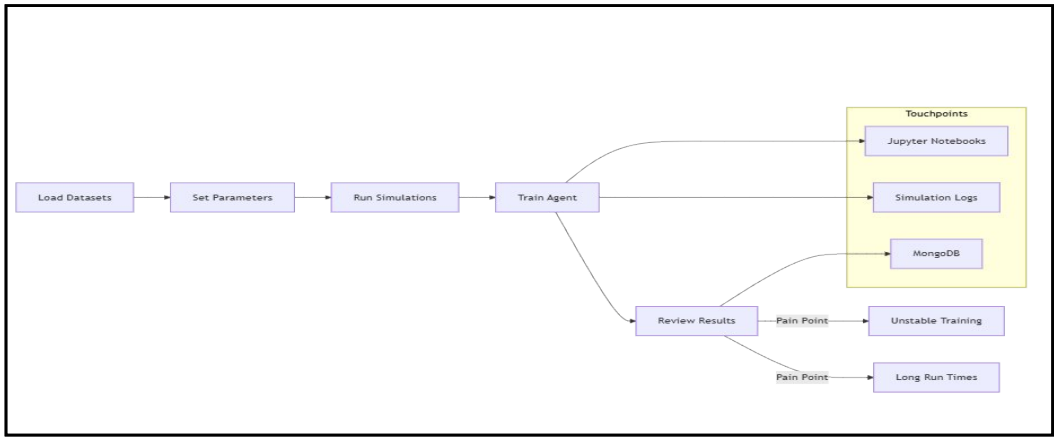
2. System Architecture Diagram
Illustrates the entire RL pipeline beginning with raw data preprocessing and moving into the Gymnasium simulation environment. The agent interacts with the environment, receives rewards, and logs all results into MongoDB. Shows how data and decisions flow continuously through the system.

3. MDP Framework Diagram (Core RL Logic)
Represents the fundamental State, Action, Transition, Reward, Next State loop. States include stock levels, demand, pricing, and lead times, while actions represent reorder or price decisions. Demonstrates how rewards and transitions drive policy learning.

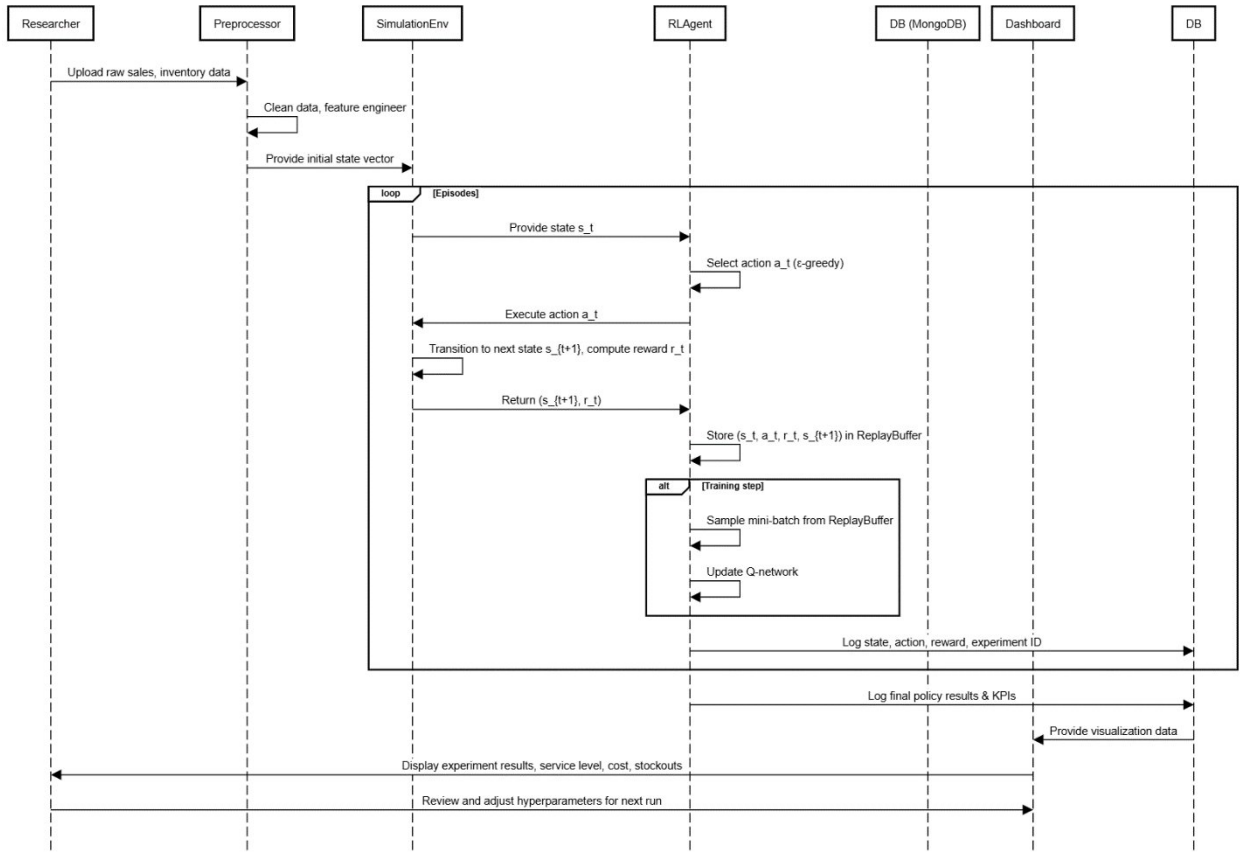
4. RL Training Loop Diagram
Visualizes the DQN training cycle, starting with environment reset and state observation. Includes epsilon-greedy action selection, reward collection, replay buffer storage, and Q-network updates. Highlights mechanisms that stabilize training like replay memory and target networks.

5. Data Pipeline Diagram
Shows how raw sales and inventory data are cleaned, engineered, and converted into MDP-compatible state vectors. Connects preprocessing directly to the simulation environment. Identifies where processed data and logs are stored.

6. Research-Oriented Database Diagram (MongoDB Collections)
Depicts how experiments link to simulation states, actions, rewards, and domain-specific snapshots. Each collection stores structured experiment logs such as Q-values, metrics, and timestamps. Demonstrates why a hierarchical document database fits RL workflows better than relational schemas.



System Architecture Diagram



References:

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.

Mnih, V., et al. (2015). *Human-level control through deep reinforcement learning*. Nature, 518(7540), 529-533. https://doi.org/10.1038/nature14236

Chodorow, K., & Dirolf, M. (2019). *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage* (3rd ed.). O'Reilly Media.

Martin, R. C. (2008). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.