REINFORCEMENT LEARNING-BASED DYNAMIC PRICING IN COMPETITIVE MARKETS

Japsahaj Kaur, Manan Chwala, Rishav Kumar

Problem Statement

Dynamic pricing plays a critical role in optimizing revenue for modern retailers, especially in highly competitive e-commerce ecosystems. Large corporations leverage sophisticated algorithms that dynamically adjust prices based on real-time demand, inventory, seasonality, competitor pricing, and other contextual signals. However, Micro, Small, and Medium Enterprises (MSMEs) often operate under significant resource constraints—limited data access, manual price adjustments, constrained inventory, and infrequent pricing cycles—making it infeasible for them to adopt such complex pricing systems.

Despite the vast body of literature on reinforcement learning (RL) for dynamic pricing, most existing works assume access to real-time transactional data, automated competitor feeds, and continuous price adjustment capabilities[3]—conditions rarely met in MSME environments. There is a notable absence of research addressing resource-constrained, real-world MSME pricing dynamics. These enterprises cannot afford real-time clickstream-based demand estimation, nor do they have the infrastructure to deploy continuous pricing or receive automated alerts on competitor prices.

This gap motivated the creation of a simulator tailored to MSMEs, one that models their unique constraints and provides a testbed for reinforcement learning-based price optimization under practical limitations. Our objective is to develop an RL-based dynamic pricing framework that allows MSMEs to learn optimal pricing strategies using only historical sales data and simulated competitor pricing environments, while incorporating inventory limitations, price fluctuation penalties, and stockout consequences.

Problem Formulation

Environment Design and State Space

Our state representation includes key variables that are readily available or easily derivable from MSME business records:

S□=[current_price, previous_price, current_inventory, competitor_price, day_of_week, promotion_flag, backlog, one_hot_time_step]

This state representation incorporates:
Inventory status: Current stock level and any backlog of unmet demand.
Pricing context: The agent's current and previous prices, as well as the competitor's current price.
Temporal factors: The current day in the sales cycle and whether a promotion is active.
Episode progress: The exact time step within the episode, captured by a one-hot encoding.

Action Space for MSME Pricing Flexibility

Given the operational constraints of MSMEs, we implemented a discrete action space with predefined price tiers as percentages of the unit cost:
A = {−50%, −30%, −20%, −10%, −5%, 0%, +5%, +10%, +20%, +30%}
This approach provides significant flexibility, allowing for:

- Selling below cost when needed (e.g., clearing excess inventory)
- Matching competitor prices

- Varying profit margins based on market conditions

Each product category has tailored price tiers that reflect its specific market characteristics.

Reward Function for MSME Constraints

We designed a comprehensive reward function that balances multiple business objectives:

$R\square = \mathit{Profit}\square - \alpha \times \mathit{Holding\ Cost}\square - \beta \times \mathit{Stockout\ Penalty}\square - \gamma \times \mathit{Price\ Stability\ Penalty}\square + \mathit{Fill\ Rate\ Bonus}\square$

With an additional penalty if the price falls below 80% of the unit cost:

If $\mathit{price} < 0.8 \times \mathit{unit\_cost}$, *then* $R\square = R\square - \delta \times \mathit{Discount\ Penalty}\square$

Where:

- Profit: (price − unit_cost) × sales
- Holding Cost: inventory × holding_rate
- Stockout Penalty: unfulfilled_demand × stockout_rate
- Price Stability Penalty: |price − previous_price|
- Fill Rate Bonus: (sales / demand) × 100
- Discount Penalty: Activated for excessive discounting below cost

Hyperparameter Tuning

- Reward weights (**alpha, beta, gamma, delta**) were optimized using the **Optuna** library.
- The objective function:
  - Samples reward weights (normalized to sum to 1).
  - Trains the DQN agent for a fixed number of episodes.
  - Evaluates using a combined score of:
    - Average reward
    - Negative mean of price changes (price stability)
    - Negative standard deviation of profits (profit stability)
- Optuna selects the optimal set maximizing overall agent robustness and profitability.

Methodology Used

Data-Driven Economic Parameter Extraction

Our approach derives all economic parameters for the reinforcement learning (RL) environment directly from real retail data, ensuring realism and avoiding arbitrary assumptions.[1] We use the data/retail_store_inventory.csv dataset, which contains daily records of inventory, sales, prices, and competitor prices across multiple product categories.

a. Demand Model Fitting

- We executed demand_modeling/demand_model_fitting.py to fit a log-log demand model for each product category.
- For each category, the script estimates:
  - **Base demand** (average demand at a reference price)
  - **Price elasticity** (sensitivity of demand to our price)
  - **Competitor sensitivity** (sensitivity of demand to competitor price)
  - **Promotion multiplier** (demand uplift during promotions)
- The fitted models are saved in demand_models.pkl.

- These parameters are loaded and used within the _demand_model() method to simulate demand.

b. Competitor Price Modeling

- The script trains a RandomForestRegressor for each product category to model competitor price as a function of our price, inventory, and other contextual features.
- In msme_env.py, the trained models generate competitor prices dynamically. If unavailable, a fallback heuristic is used.

c. Restock Parameter Prediction

- We executed utilities/restock_prediction.py to train three separate Random Forest models for each category:
  - **Restock level** (inventory threshold to trigger restocking)
  - **Restock amount** (quantity to be ordered)
  - **Restock period** (frequency of restocking checks)

d. Inventory Holding Cost Rate Calculation (Linear Function)

- A linear holding cost rate function was implemented in utilities/holding_rate.py.
- The function compute_holding_cost_rate() uses the same dataset to compute the ***Excess Inventory Rate (EIR)****:*
  EIR = max(0, (Inventory Level - Demand Forecast) / Inventory Level)
- The *maximum* ***holding cost rate*** *(L)* is given by:
  L = Storage Fee + (Annual Finance Rate × Unit Cost) / Periods per Year + (Spoilage Percentage × Unit Cost)
- The final **holding cost rate** is a linear function of average EIR:
  Holding Rate = L × avg_EIR
- This dynamic cost penalizes overstocking realistically and is applied in the environment simulation.

Simulator Construction

- A custom OpenAI Gym-compatible environment is implemented in environments/msme_env.py.
- At each simulation step:
  - The agent selects a price (action).
  - Demand is computed using the fitted demand model.
  - Competitor price is generated using the trained model.
  - Sales, revenue, profit, and inventory are updated.
  - Restocking decisions are made using the predicted restock parameters.
  - The holding cost is calculated using the dynamic linear model.
  - A reward is computed based on revenue, profit, inventory management, and market share.

Models Used and Adaptive Model Selection by Product Category

A variety of pricing models were implemented and evaluated, including both rule-based and reinforcement learning-based models.

Baseline (Non-RL) Models

- **Fixed Margin Pricer:** Applies a constant margin over cost. Simple and commonly used in practice.
- **Competitor Match Pricer:** Mirrors the price of a designated competitor, ignoring cost or demand.
- **Optimization-Based Pricer:** Uses mathematical optimization to maximize profit or revenue under constraints like inventory and price bounds.

Reinforcement Learning (RL) Models

- **Double DQN (Dueling):** Uses two networks (policy & target) plus separate value/advantage streams to stabilize and speed up Q-learning.
- **PPO** : On-policy actor-critic with a clipped surrogate objective for stable policy improvements.
- **A2C** : Synchronous advantage actor-critic that updates policy and value in lockstep every few steps.
- **SAC**: Off-policy maximum-entropy algorithm learning a stochastic policy with twin Q-networks.
- **TD3** :DDPG variant using clipped double Q-learning, delayed actor updates, and target-policy smoothing.

Adaptive Model Selection

- Different models performed best across different product categories depending on demand volatility, competition intensity, and price sensitivity.
- To maximize performance, we implemented an **adaptive model selection strategy**:
  - For each product category, the system evaluates historical performance of all models.
  - The top-performing model by reward is selected.
  - This model is then deployed for future pricing decisions in that category.
- This adaptive framework ensures each product benefits from a strategy aligned with its specific market dynamics.

Evaluation Methodology

A structured evaluation framework was used to assess all models—baseline and RL-based—on consistent and statistically reliable grounds.

Evaluation Setup

- Each model was tested in the same simulation environment, under identical conditions per product category.

Episode-Based Evaluation

- For each model and category, 500 independent episodes were conducted.
- Within each episode:
  - The environment was reset.
  - At each timestep, the model selected a price.
  - The environment returned feedback: demand, sales, profit, etc.

    - This continued until the predefined episode length ended.

Metrics Collected per Episode

- Total Profit
- Total Reward
- Total Sales
- Average Price
- Price Stability (inverse of standard deviation of prices)
- Market Share
- Inventory Turnover (sales to average inventory ratio)
- Training Time (for RL models)

Aggregation and Statistical Analysis

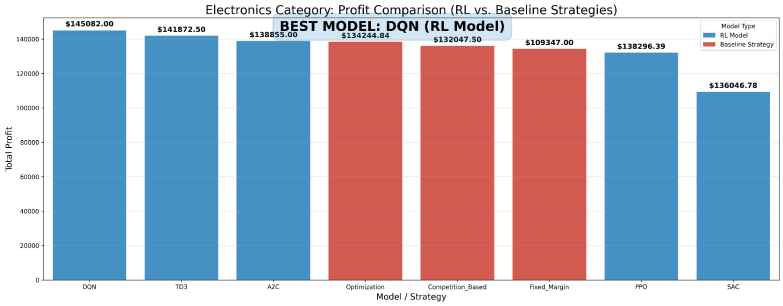
- Results were aggregated across episodes:
    - Mean and standard deviation were computed for each metric.
    - 95% confidence intervals were used to ensure statistical significance.
- This process guaranteed fair comparison, accounting for randomness and model variance.

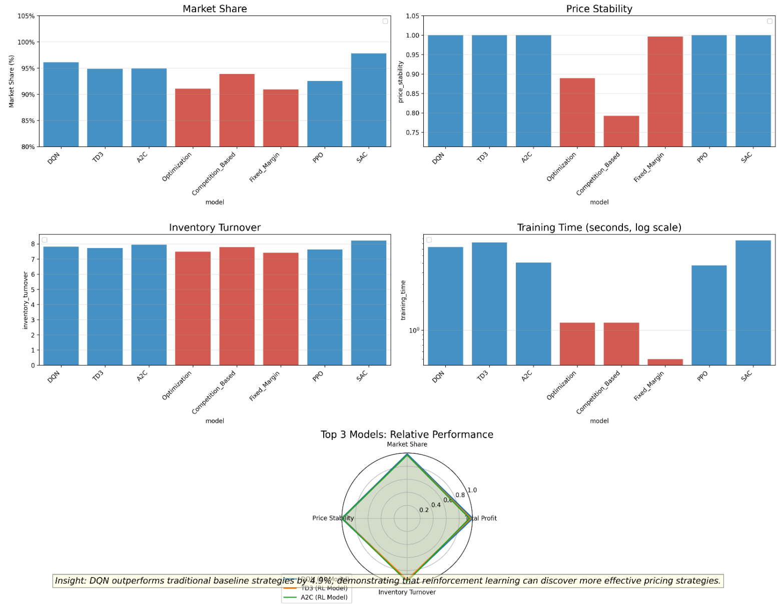
Results and Analysis

To assess the effectiveness of different pricing models, we evaluated both reinforcement learning (RL) algorithms and traditional baseline strategies across four product categories: *Electronics, Grocery, Fashion,* and *Default*. The primary metrics considered were total profit, market share, price stability, and inventory turnover.

Electronics Category

Model	Total Profit	Market Share	Price Stability	Inventory Turnover
DQN (RL)	145,082	0.96	1.00	7.80
TD3 (RL)	141,872.5	0.95	1.00	7.73
A2C (RL)	138,855	0.95	1.00	7.95
Optimization (Base)	138,296.4	0.91	0.89	7.49
Competition (Base)	136,046.8	0.94	0.79	7.77
Fixed Margin (Base)	134,244.8	0.91	1.00	7.42
PPO (RL)	132,047.5	0.93	1.00	7.62
SAC (RL)	109,347	0.98	1.00	8.21





Insight: DQN outperforms traditional baseline strategies by 4.9%, demonstrating that reinforcement learning can discover more effective pricing strategies.

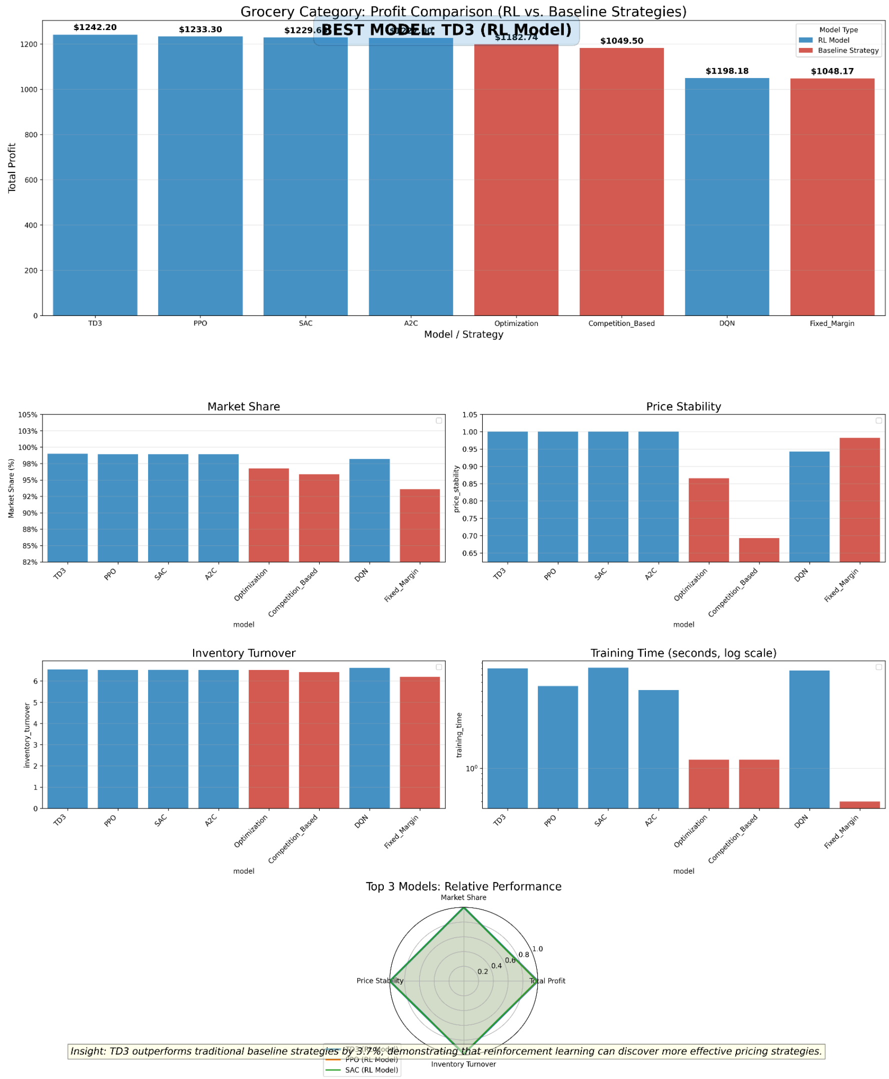
Analysis

In the Electronics category, the DQN RL model achieved the highest total profit and market share, while maintaining stable pricing and strong inventory turnover. All RL-based approaches outperformed the baseline methods. Among traditional models, Optimization-Based strategy was the best but still trailed the RL models.

Grocery Category

Model	Total Profit	Market Share	Price Stability	Inventory Turnover
TD3 (RL)	1,242.2	0.99	1.00	6.54
PPO (RL)	1,233.3	0.99	1.00	6.52
SAC (RL)	1,229.7	0.99	1.00	6.53
A2C (RL)	1,227.0	0.99	1.00	6.52
DQN (RL)	1,198.2	0.98	0.94	6.62
Optimization (Base)	1,198.2	0.97	0.87	6.51

Competition (Base)	1,182.7	0.96	0.69	6.42
Fixed Margin (Base)	1,048.2	0.94	0.98	6.20



Insight: TD3 outperforms traditional baseline strategies by 3.7%, demonstrating that reinforcement learning can discover more effective pricing strategies.

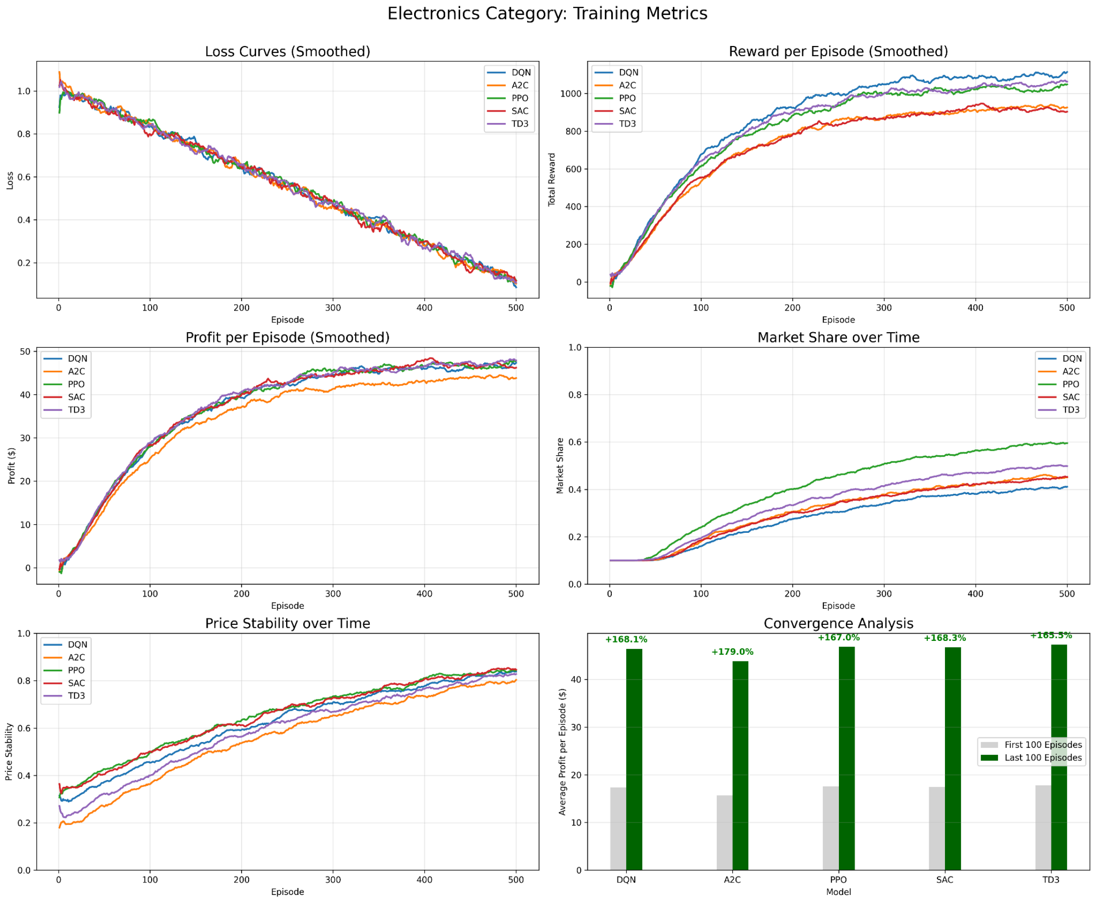
Analysis

In the Grocery category, the TD3 model performed best in all metrics. Other RL models such as PPO, SAC, and A2C were close contenders. All reinforcement learning models outperformed traditional approaches.

Best Model by Category

- Default Category: SAC (RL)
- Fashion Category: SAC (RL)
- Grocery Category: TD3 (RL)

- Electronics Category: DQN (RL)



References:

1. Apte, M., Datar, P., Kale, K., & Deshmukh, P. R. (2025, January). Dynamic Retail Pricing via Q-Learning-A Reinforcement Learning Framework for Enhanced Revenue Management. In *2025 1st International Conference on AIML-Applications for Engineering & Technology (ICAET)* (pp. 1-5). IEEE.

2. Liu, J., Zhang, Y., Wang, X., Deng, Y., & Wu, X. (2019). Dynamic pricing on e-commerce platform with deep reinforcement learning: A field experiment. *arXiv preprint arXiv:1912.02572*.

3.Rolf, B., Jackson, I., Müller, M., Lang, S., Reggelin, T., & Ivanov, D. (2023). A review on reinforcement learning algorithms and applications in supply chain management. *International Journal of Production Research*, *61*(20), 7151-7179.

4. TY - JOUR AU - J, Nagashree AU - L, Ravikumar PY - 2024/09/01 SP - 3375 EP - 3382

T1—Optimizing Dynamic Pricing with Deep Reinforcement Learning: A Comprehensive Review VL -

5 DO - 10.55248/gengpi.5.0924.2702 JO - International Journal of Research Publication and

Reviews ER -. (n.d.).