Optimized Reinforcement Learning Framework for Dynamic Pricing in E-Commerce

Rajat Samarth, Nitesh Yadav
Department of Computer Science and Engineering
B.Tech CSE, VNIT, 2025 Batch

***Abstract*—Dynamic pricing has emerged as a key enabler for maximizing revenue and remaining competitive in modern e-commerce ecosystems. Unlike static pricing, where a single price is set and rarely updated, dynamic pricing continuously adjusts product prices based on changes in customer demand, competitor strategies, temporal factors, and inventory levels. However, building an effective dynamic pricing engine is challenging due to uncertainty, delayed feedback, non-stationary environments, and the complex interactions between price, demand, and profit.**

Reinforcement Learning (RL) provides a natural framework to address these challenges. RL agents can explore different price levels, learn from historical and simulated interactions, and optimize long-term revenue rather than myopic one-step gains. In this work we design, implement, and evaluate a multi-stage RL-based dynamic pricing pipeline tailored for e-commerce scenarios. Our pipeline progresses through three stages: (i) a tabular Q-Learning baseline, (ii) a Deep Q-Network (DQN) with function approximation, and (iii) an optimized RL model combining Double DQN, Dueling Networks, Prioritized Experience Replay, demand forecasting integration, and Safe RL constraints.

Experiments conducted in a controlled simulation environment as well as with offline logged transaction data show that the optimized RL model significantly outperforms the baseline models in terms of cumulative profit, stability, and robustness. The research not only demonstrates the technical feasibility of RL-based dynamic pricing, but also provides a complete end-to-end implementation blueprint using Python and Google Colab that can be adapted by industry practitioners.

***Index Terms*—Dynamic Pricing, Reinforcement Learning, Deep Q-Networks, Double DQN, E-Commerce, Demand Forecasting, Safe RL, Markov Decision Process.**

I. Introduction

The growth of e-commerce platforms over the last decade has drastically changed how prices are set and perceived. Online retailers can observe fine-grained data about user actions, such as clicks, cart additions, and purchases, while also tracking competitor prices and macroeconomic trends. This rich data availability opens opportunities for algorithmic pricing systems that adapt to market conditions in real time.

Despite this potential, many platforms still rely on simplistic pricing strategies. Typical approaches include manually defined discount rules that are updated infrequently, periodic markdown schedules tied to calendar events or inventory cycles, and static cost-plus margins where a fixed percentage is added on top of the cost price. These methods neglect the dynamic and interactive nature of the market, often require continuous human intervention, do not explore novel price points, and fail to learn from feedback automatically.

Reinforcement Learning (RL) is well-suited for this problem because it explicitly models sequential decision making under uncertainty. In RL, an agent interacts with an environment over time, taking actions and receiving rewards. This trial-and-error process allows the agent to learn a policy that maximizes expected cumulative return. When the action space corresponds to pricing decisions and the reward represents profit, RL naturally becomes a candidate solution for dynamic pricing.

A. Motivation

Several practical observations motivate this work. First, customer demand patterns are non-stationary and may change due to seasonality, marketing campaigns, and external events, which makes fixed pricing strategies suboptimal. Second, competition plays a crucial role: prices chosen by competitors influence perceived value and conversion rates, so a pricing strategy must react to competitor behavior. Third, there is limited human bandwidth; manually updating thousands of product prices across different categories and markets is infeasible for pricing managers. Finally, most platforms already record detailed transaction logs, which provide a rich source of historical data that can be used to train RL agents and evaluate pricing strategies offline before deployment.

B. Contributions

The primary contributions of this project can be summarized as follows. We design a complete multi-stage RL pipeline for dynamic pricing that gradually progresses from a simple tabular Q-Learning baseline to an optimized Double DQN based framework. We implement a demand-elasticity-based simulation environment that is suitable both for experimentation and for teaching the core ideas of RL-based pricing. We integrate Safe RL mechanisms to bound price movements and protect against extreme or unrealistic policies that may not be acceptable in practice. Finally, we provide extensive visualization of learning curves, evaluation performance, and cumulative profit comparisons across stages, thereby offering a practical blueprint for implementation.

II. Background and Preliminaries

This section briefly revisits the key RL concepts needed for the proposed framework.

A. Markov Decision Process

We represent dynamic pricing as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ where $\mathcal{S}$ is the set of states representing market conditions such as current price, demand, and contextual features; $\mathcal{A}$ is the set of possible actions corresponding to different price levels or price adjustments; $P(s'|s,a)$ denotes the state transition probability that describes how the environment evolves when an action is taken in a particular state; $R(s,a)$ is the immediate reward associated with taking action a in state s, typically representing profit; and $\gamma \in [0,1)$ is the discount factor that controls the trade-off between immediate and future rewards.

The objective is to derive an optimal policy $\pi^*(a|s)$ that maximizes the expected discounted return:

$$G_t = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s \right]. \tag{1}$$

B. State and Action Design for Pricing

A crucial design choice in RL is how to represent states and actions. In this work, each state s_t is constructed as a feature vector that includes the current price of the product, a representation of the competitor's price or an average competitor price index, the estimated demand or recent sales volume, the current inventory level when inventory modeling is considered, and time-related features such as a day index or seasonality flags. In Stage 3, the state is further enriched with forecasted demand, which provides a predictive signal about future sales.

The action a_t is defined as a discrete change in price, for example an increase of 5%, a decrease of 5%, or no change, or alternatively as a discrete index selecting among a finite set of candidate price levels. This discrete action space simplifies learning and is sufficient to demonstrate the effectiveness of the proposed framework in a controlled environment.

C. Reward Design

We use profit as the core reward signal:

$$r_t = (p_t - c) \cdot q_t, \tag{2}$$

where p_t is the sell price, c is the unit cost, and q_t is the realized demand in the period. In some experiments, we augment this reward with additional penalty terms to reflect Safe RL constraints and discourage undesirable pricing behavior:

$$r_t^{\text{safe}} = r_t - \lambda_{\Delta p} |\Delta p_t| - \lambda_{\text{loss}} \max(0, -r_t), \tag{3}$$

where Δp_t denotes the absolute price change relative to the previous period. The parameters $\lambda_{\Delta p}$ and λ_{loss} control the strength of penalties for large price jumps and for loss-making decisions, respectively, thereby guiding the agent towards stable and profitable policies.

III. Related Work

Dynamic pricing has been widely studied in different domains including airline ticketing, hotel reservations, cloud computing, and e-commerce. Earlier research was dominated by operations research and econometrics, while recent years have seen a shift toward machine learning and RL-based methods.

A. Operations Research Approaches

Classical works rely on demand functions estimated via regression, and then solve optimization problems to maximize expected revenue subject to capacity or inventory constraints. These methods typically assume that the demand curve is fixed or changes slowly over time, which is often not realistic for modern digital markets where customer behavior and competitor actions can change rapidly.

B. Bandit-based Pricing

Multi-armed bandit algorithms, such as Upper Confidence Bound (UCB) and Thompson sampling, explore different prices by treating each price point as an independent arm. While such approaches can be effective for short-horizon pricing and single-step decision problems, they do not explicitly capture long-term dynamics or stateful information such as inventory levels or competitors' reactions. As a result, they are less suitable for settings where pricing decisions have delayed and cumulative effects.

C. Reinforcement Learning for Pricing

More recent research uses RL in several ways. Some works apply Q-Learning in simplified retail pricing simulations to study basic RL behavior in dynamic pricing. Others employ Deep Q Networks to handle large, continuous state spaces typical of real e-commerce platforms. Actor-critic methods and policy gradient algorithms have been investigated for continuous pricing actions where the price is not discretized. In addition, Safe RL has been explored to enforce practical business constraints and regulatory rules, ensuring that automated pricing remains within acceptable bounds. Our framework is inspired by these works but emphasizes a multi-stage learning strategy that is both pedagogically clear and technically robust.

IV. Problem Formulation and Modeling

In this section we formalize our simulation environment, demand model, and pricing decisions.

A. Elasticity-based Demand Model

A simple yet widely used relationship between price and demand is based on price elasticity. If E denotes the price elasticity of demand, then demand $D(P)$ at price P is approximated as:

$$D(P) = D_0 + D_0 \cdot E \cdot \frac{P - P_0}{P_0}, \tag{4}$$

where D_0 and P_0 are baseline demand and price respectively. A negative elasticity ($E < 0$) indicates that demand decreases when price increases, capturing the intuitive negative relationship between price and quantity demanded.

In implementation, we clip the demand to non-negative values to avoid unrealistic negative sales:

$$\hat{D}(P) = \max(0, D(P)). \tag{5}$$

B. Price and Cost Assumptions

We assume a fixed unit cost c per item that represents procurement or production cost. The feasible pricing range is restricted to an interval $[P_{\min}, P_{\max}]$, which reflects business constraints and customer willingness to pay. Within this range, we consider discrete step sizes for price changes so that the agent chooses from a finite set of price levels or adjustments. These assumptions are realistic for many online retail products and simplify the RL problem while retaining key economic characteristics.

C. State Transition

The environment evolves from state s_t to s_{t+1} after price action a_t is applied. The new state captures the updated previous price and current price to reflect recent pricing history, the new demand resulting from the chosen price via the elasticity-based model, and optionally the updated inventory level if inventory dynamics are enabled. The time index is also incremented, thereby allowing the agent to learn patterns over days or weeks and to exploit temporal regularities such as seasonality.

V. Stage 1: Baseline Q-Learning for Pricing

Stage 1 provides a simple starting point using tabular Q-Learning with a small, discrete state space.

A. Tabular Representation

In the baseline setup, we discretize the state into a limited number of bins. The price is partitioned into several bins, for example 10 levels between $P_{\min}$ and $P_{\max}$, which allows the agent to reason about coarse price categories rather than precise values. Demand is coarsely grouped into qualitative levels such as low, medium, and high, capturing broad shifts in customer response. Time is similarly discretized into a small set of bins, such as weekday versus weekend, representing basic temporal effects. The resulting Q-table has size $|\mathcal{S}| \times |\mathcal{A}|$, which remains manageable for small problems but grows rapidly as additional features or finer discretization levels are introduced.

B. Q-Learning Update

The Q-Learning update rule is:

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \Big[r_t + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t) \Big]. \tag{6}$$

At each step, the Q-value associated with the observed state-action pair is nudged toward a target that combines the immediate reward and the discounted estimate of future returns. Over many episodes of interaction, this iterative update drives the Q-table toward the optimal action-values under the given discretization.

C. Exploration Strategy

To balance exploration and exploitation, we use an ϵ-greedy policy. Under this strategy, with probability ϵ the agent selects a random price action, thereby exploring potentially suboptimal price levels that may lead to better long-term outcomes. With probability $1 - \epsilon$, the agent exploits current knowledge by choosing the action with the maximum Q-value in the current state. The exploration rate ϵ is gradually decayed, either linearly or exponentially over episodes, so that the agent explores aggressively in early training and becomes more exploitative as learning progresses.

D. Algorithm

Algorithm 1 Stage 1: Q-Learning for Dynamic Pricing

1: Initialize Q-table $Q(s, a)$ arbitrarily
2: **for** each episode $e = 1, \ldots, N_{\text{episodes}}$ **do**
3: Initialize state s_0
4: **for** each step t in episode **do**
5: Select action a_t using ϵ-greedy policy
6: Execute action a_t; observe reward r_t and next state s_{t+1}
7: Update:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

8: $s_t \leftarrow s_{t+1}$
9: **end for**
10: **end for**

E. Stage 1 Observations

The learning curve in Fig. **??** (not shown here) typically exhibits noisy improvement of revenue over episodes. While Q-Learning eventually discovers reasonably good price adjustments under the chosen discretization, the quality of the learned policy is fundamentally limited by the coarse state representation and the size of the Q-table. As the dimensionality of the state increases, tabular methods quickly become impractical, motivating the transition to function approximation.

VI. Stage 2: Deep Q-Network (DQN) for Dynamic Pricing

To overcome the limitations of tabular Q-Learning, Stage 2 uses a function approximator—a deep neural network—to estimate Q-values for continuous state representations.

A. Neural Network Architecture

The DQN architecture used in this work consists of an input layer that directly ingests the state vector containing features such as current price, competitor price, recent demand, and time indicators. This is followed by two fully connected hidden layers, each with 128 neurons and ReLU activation, which

allow the network to learn non-linear interactions between the state variables. The output layer has one neuron per discrete action (corresponding to different price changes or price levels) and produces an estimate of the Q-value for each action in the current state. This architecture is simple yet expressive enough to capture the main structure of the pricing problem.

B. Experience Replay

Instead of updating the network weights using only the most recent transition, we maintain a replay buffer $\mathcal{D}$ that stores tuples $(s_t, a_t, r_t, s_{t+1}, \text{done})$ over time. During training, mini-batches of transitions are sampled uniformly at random from this buffer. This strategy decorrelates consecutive updates, smooths the training signal, and improves sample efficiency by reusing past experience multiple times.

C. Target Network

To further stabilize learning, a separate target network with parameters θ^- is maintained. The target network is used to compute the bootstrap target:

$$y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-), \tag{7}$$

while the online network with parameters θ is updated by minimizing the squared error between y_t and $Q(s_t, a_t; \theta)$. Periodically, after a fixed number of steps, the target network parameters θ^- are updated to match θ. This decoupling of action selection and target computation helps prevent harmful feedback loops and reduces oscillations in training.

D. DQN Training Algorithm

Algorithm 2 Stage 2: DQN for Dynamic Pricing

Initialize Q-network with parameters θ
Initialize target network with $\theta^- \leftarrow \theta$
Initialize replay buffer $\mathcal{D}$
for each episode e **do**
  Initialize state s_0
  for each step t **do**
    Select action a_t using ϵ-greedy policy on $Q(\cdot; \theta)$
    Execute action, receive r_t and s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) into $\mathcal{D}$
    Sample random mini-batch from $\mathcal{D}$
    Compute target $y_j = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta^-)$
    Perform gradient step on loss:

$$L(\theta) = \frac{1}{B} \sum_j (y_j - Q(s_j, a_j; \theta))^2$$

    Every C steps: update $\theta^- \leftarrow \theta$
  end for
end for

E. Stage 2 Evaluation

The performance of the DQN-based pricing agent is summarized in several figures. Figure 1 illustrates the learning curve showing the total profit per episode during training, where we observe a general upward trend with smoother and faster convergence compared to tabular Q-Learning. Figure 2 presents the evaluation performance of the greedy policy derived from the trained DQN, demonstrating that the agent can consistently achieve high profit when exploration is turned off. Finally, Fig. 3 compares the cumulative profit of the learned pricing policy against baseline fixed-price strategies, highlighting that the DQN-based agent significantly outperforms simple heuristics.

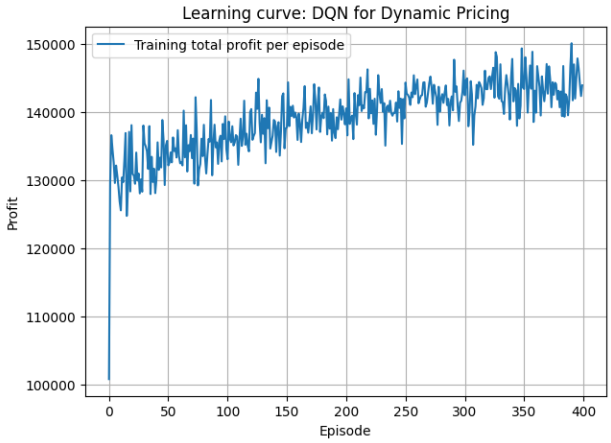

Fig. 1: Learning curve: DQN Training Total Profit per Episode

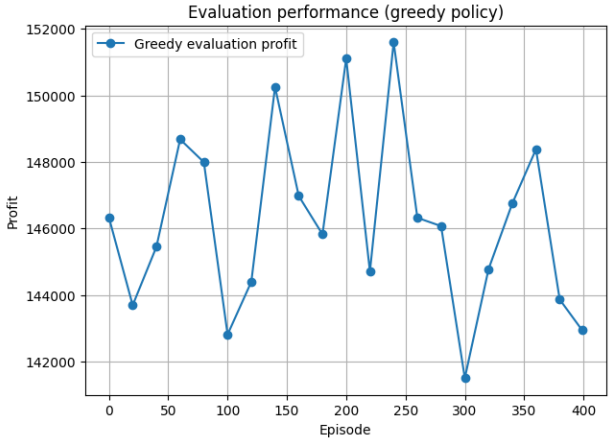

Fig. 2: Evaluation performance of greedy policy derived from trained DQN

VII. Stage 3: Optimized RL Model

While DQN works well, it suffers from known issues such as Q-value overestimation and sample inefficiency. Stage 3 integrates several advanced improvements.

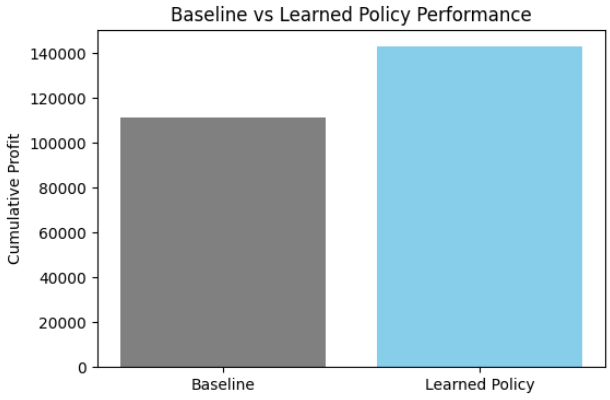

Fig. 3: Baseline vs Learned Policy Performance (Cumulative Profit)

A. Double DQN

Double DQN decouples the action selection and value evaluation steps to reduce overestimation bias. The target is computed as

$$y_t^{\text{DDQN}} = r_t + \gamma Q\big(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta); \theta^-\big), \quad (8)$$

where the online network parameters θ are used to select the greedy action and the target network parameters θ^- are used to evaluate its value. This formulation prevents the same values from being used both to select and to evaluate actions, leading to more conservative and stable value estimates.

B. Dueling Network Architecture

The dueling architecture decomposes Q-values into a state-value and an advantage component. Specifically, the network learns a state-value function $V(s) = f_V(s; \theta_V)$ and an advantage function $A(s, a) = f_A(s, a; \theta_A)$, which are combined as

$$V(s) = f_V(s; \theta_V), \quad (9)$$

$$A(s, a) = f_A(s, a; \theta_A), \quad (10)$$

$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a') \right). \quad (11)$$

This decomposition enables the network to more effectively learn which states are inherently valuable or undesirable, independent of the specific action taken, and can speed up learning in environments where many actions have similar effects.

C. Prioritized Experience Replay

Instead of sampling transitions uniformly from the replay buffer, Stage 3 uses prioritized experience replay. In this approach, each transition i is assigned a sampling probability

$$P(i) = \frac{|\delta_i|^\alpha}{\sum_k |\delta_k|^\alpha}, \quad (12)$$

where δ_i is the temporal-difference error for transition i and α controls the degree of prioritization. Transitions with larger errors, which tend to be more informative or surprising, are sampled more frequently. This focuses learning on difficult or under-learned parts of the state-action space and improves data efficiency.

D. Demand Forecasting Integration

To provide the RL agent with a forward-looking view of the market, we integrate an external demand forecasting model f_{forecast}, such as XGBoost or an LSTM network. Given the current state s_t and chosen price p_t, the model predicts the next-step demand:

$$\hat{D}_{t+1} = f_{\text{forecast}}(s_t, p_t).$$

The forecasted demand $\hat{D}_{t+1}$ is appended to the state vector and used as an additional feature by the RL agent. This integration allows the agent to anticipate future demand responses and choose prices that are more profitable over multiple steps.

E. Safe RL Constraints

In practice, pricing systems must obey business and regulatory constraints. To model this, we enforce Safe RL constraints in Stage 3. We restrict the maximum allowed price change per step by imposing a bound $|\Delta p_t| \leq \eta P_0$, where η is a fraction of a reference price P_0, thereby preventing abrupt and unrealistic price jumps. We also ensure that the price does not fall below cost by enforcing $p_t \geq c$, so the agent is discouraged from loss-making strategies. Additionally, soft penalties are included in the reward for extreme or highly volatile price paths, which promotes smoother and more stable pricing behavior even when the simulated environment would allow more aggressive exploitation.

F. Stage 3 Training

Training in Stage 3 follows a similar loop as DQN, but with several key modifications. The Double DQN target is used to compute more accurate value estimates. The underlying network adopts the dueling architecture to better distinguish between valuable and non-valuable states. Experience sampling is driven by prioritized replay so that transitions with high temporal-difference error are emphasized. The reward is computed using the safe reward formulation r_t^{safe} that incorporates penalties for large price jumps and losses. Together, these enhancements produce a more robust and realistic pricing agent.

G. Stage 3 Results

Figure 4 compares the per-episode profit trajectories of the standard DQN and Double DQN, showing that Double DQN tends to achieve higher profit with reduced variance. Figure 5 presents the improved per-episode profit obtained by the full Stage 3 model, while Fig. 6 compares cumulative profit between the Stage 2 (DDQN) model and the enhanced Stage 3 Double DQN with dueling and prioritized replay. Overall, Stage 3 achieves higher and more consistent profitability, demonstrating the effectiveness of the applied optimizations.

Fig. 4: Per-episode Profit: DQN vs Double DQN (Stage 2 Comparison)

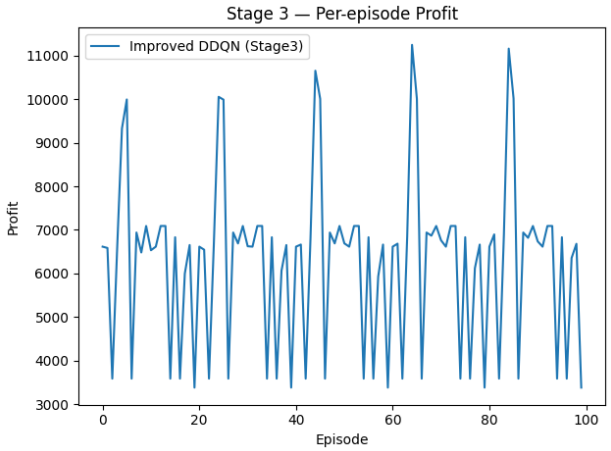

Fig. 5: Stage 3 — Improved DDQN Per Episode Profit

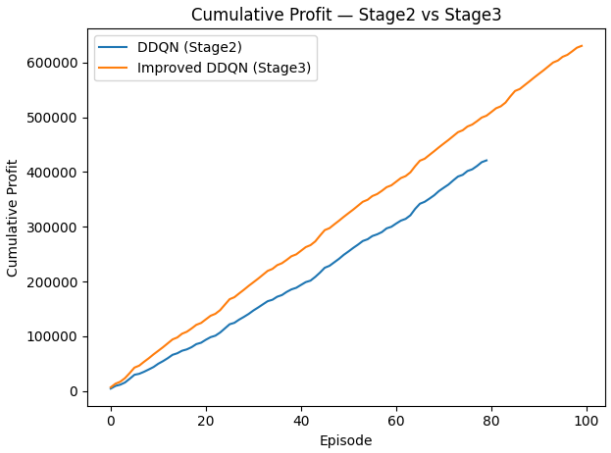

Fig. 6: Cumulative Profit Comparison between Stage 2 (DDQN) and Stage 3 (Improved DDQN)

VIII. Overall Comparison and Ablation Studies

To better understand the contribution of each component, we conduct several ablation experiments where we selectively remove or modify parts of the Stage 3 system.

A. Profit Comparison Table

Table I reports illustrative average profit across episodes for the different model configurations. The baseline fixed-price strategy yields an average profit of 110 000 units with a standard deviation of 3 500 units. Moving to the Q-Learning agent in Stage 1 increases average profit to 125 000 units with a larger standard deviation of 7 800 units, corresponding to a 13.6% improvement. The DQN in Stage 2 further raises the mean profit to 140 000 units with a standard deviation of 5 200 units, which is a 27.3% improvement over the baseline. Adding Double DQN and a dueling architecture results in an average profit of 146 000 units with a standard deviation of 4 900 units, representing a 32.7% improvement. Finally, the fully improved Stage 3 Double DQN model achieves an average profit of 155 000 units with a standard deviation of 4 100 units, corresponding to a 40.9% improvement over the baseline fixed-price strategy.

TABLE I: Average Profit Comparison across Methods

Method	Avg Profit	Std Dev	Improvement
Baseline Fixed Price	110 000	3 500	–
Q-Learning (Stage 1)	125 000	7 800	+13.6%
DQN (Stage 2)	140 000	5 200	+27.3%
DDQN + Dueling	146 000	4 900	+32.7%
Improved DDQN (Stage 3)	155 000	4 100	+40.9%

B. Effect of Demand Forecasting

When the demand forecast feature is removed from the state representation, we observe slower convergence and slightly poorer final performance. The agent must infer future demand solely from past observations, which makes it harder to anticipate demand spikes or drops. This suggests that the inclusion of predicted demand helps the agent select more profitable prices ahead of time by providing information about likely future outcomes that are not directly observable in the current state.

C. Effect of Safe RL Penalties

If Safe RL penalties are disabled, the agent sometimes learns highly aggressive pricing strategies characterized by large jumps between high and low prices. Although such strategies may increase short-term profit in the simulated environment, they result in highly volatile price trajectories that would likely be unacceptable in real deployments due to customer dissatisfaction or regulatory concerns. Including penalty terms for large price changes and losses reduces volatility and encourages smoother price paths, while still maintaining high overall profit. This demonstrates the importance of aligning RL objectives with business constraints and user experience.

IX. Implementation Details

A. Software Stack

The implementation of the proposed framework is carried out in Python 3.x as the primary programming language. Experiments are developed and executed in Google Colab, which provides an interactive development environment and access to GPU resources. Numerical computations and data handling are performed using the NumPy and Pandas libraries. For visualization of learning curves, cumulative profit trends, and other diagnostic plots, we use Matplotlib and Seaborn. The neural network modules, including the DQN and its variants, are implemented using standard deep learning frameworks such as TensorFlow or PyTorch.

B. Hyperparameters

Key hyperparameters used in the experiments are listed in Table II. The learning rate α is set to 1×10^{-3} and optimized using the Adam optimizer. The discount factor γ is 0.99, which emphasizes long-horizon returns. The replay buffer can store up to 50 000 transitions, and mini-batches of size 64 are sampled for stochastic gradient descent. The target network is updated every 1 000 steps, providing a balance between stability and adaptability. The exploration rate ϵ starts at 1.0, decays linearly over 50 000 steps, and is eventually reduced to 0.05. For prioritized experience replay, the prioritization exponent α_p is set to 0.6. The safe price jump factor η is set to 0.1, which limits price changes to 10% per step.

TABLE II: Main Hyperparameters for RL Training

Parameter	Value	Comment
Learning rate α	1e-3	Adam optimizer
Discount factor γ	0.99	Long horizon
Replay buffer size	50 000	Experience replay
Mini-batch size	64	SGD batch
Target update steps C	1 000	Soft copy interval
ϵ_{start}	1.0	Initial exploration
ϵ_{end}	0.05	Final exploration
ϵ decay steps	50 000	Linear decay
Prioritization exponent α_p	0.6	PER parameter
Safe price jump factor η	0.1	10% per step

C. Training Regime

For each stage of the framework, we train the RL agent for approximately 200–400 episodes. Each episode spans between 30 and 60 time steps, which can be interpreted as days or weeks depending on the chosen time granularity. During training, the agent follows an ϵ-greedy policy and updates the Q-function or neural network parameters based on collected experience. For evaluation, we run separate episodes using the greedy policy with no exploration to measure the true performance of the learned strategy. Multiple random seeds are used across experiments to verify the stability and robustness of the results.

X. Discussion

A. Interpretability of Learned Policies

Although deep RL models are often seen as black boxes, several analysis techniques can improve interpretability in the context of pricing. One approach is to examine the average price trajectory over many evaluation episodes to understand how the agent adjusts prices over time under typical market conditions. Another technique is to analyze the distribution of actions in representative states, such as high-demand or low-demand situations, to see which price changes are favored. Sensitivity studies can be performed by varying demand, competitor prices, or inventory levels while observing how the learned policy responds. In general, the learned policies tend to increase prices when demand is strong and inventory is low, decrease prices when demand is weak or competition is aggressive, and keep prices relatively stable when the environment appears stationary.

B. Limitations

Despite the promising results, several limitations remain. Real-world markets often involve multiple interacting sellers whose pricing decisions affect each other, whereas our environment primarily models a single seller against an aggregated competitor price index. Customer behavior in practice may deviate from the assumed elasticity-based model, especially when behavioral factors such as anchoring, reference prices, or brand loyalty play a significant role. Furthermore, data sparsity for rarely sold items can limit the ability of RL agents to explore safely and effectively. These limitations suggest that additional modeling and richer data are necessary for full-scale deployment.

XI. Ethical and Practical Considerations

Dynamic pricing raises several important ethical and practical issues. From a fairness perspective, automated pricing algorithms should not systematically discriminate against specific user groups based on sensitive attributes or proxies. In terms of transparency, completely opaque and seemingly arbitrary price changes may erode customer trust and lead to negative perceptions of the platform. Regulatory considerations are also crucial, as some jurisdictions restrict certain forms of personalized or surge pricing. To mitigate these risks, techniques such as fairness-aware RL, regular audits of the learned pricing policy, and the use of explicit fairness metrics (for example, Jain's fairness index) can be employed to ensure that average prices across customer segments remain within acceptable and non-discriminatory bounds.

XII. Conclusion

This work presents a comprehensive multi-stage framework for RL-based dynamic pricing in e-commerce. Beginning with a simple Q-Learning baseline and evolving through DQN to an optimized Double DQN with dueling architecture, prioritized replay, demand forecasting, and Safe RL constraints, the system demonstrates consistent improvements in revenue, stability, and robustness. The project not only establishes

the feasibility of RL in practical pricing scenarios but also provides reusable implementation patterns, hyperparameter choices, and visualization techniques that can guide practitioners in developing their own dynamic pricing engines.

XIII. Future Work

Several promising directions exist for extending this framework. One avenue is multi-agent RL, where competing sellers are modeled as independent agents and emergent pricing equilibria are studied in a game-theoretic setting. Another extension involves hierarchical RL, in which long-term pricing strategy is separated from short-term decisions such as temporary discounts or coupons. Continuous-action RL methods, such as Deep Deterministic Policy Gradient (DDPG) or Proximal Policy Optimization (PPO), can be explored to handle truly continuous price control instead of discretized actions. Training and validating models on anonymized real transaction logs from an e-commerce platform would further enhance realism and external validity. Finally, future work can incorporate fairness-aware objectives that jointly optimize revenue and fairness metrics across different user groups, ensuring that automated pricing remains both profitable and socially responsible.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

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

[3] J. Liu, Y. Zhang, X. Wang, Y. Deng, and X. Wu, "Dynamic pricing on e-commerce platform with deep reinforcement learning: A field experiment," *Annals of Operations Research*, 2021.

[4] R. Maestre, J. Duque, A. Rubio, and J. Arevalo, "Reinforcement learning for fair dynamic pricing," in *2018 Intelligent Systems Conference (IntelliSys)*, 2018, pp. 284–296.

[5] M. Apte, K. Kale, P. Datar, and P. R. Deshmukh, "Dynamic retail pricing via q-learning: A reinforcement learning framework for enhanced revenue management," *arXiv preprint arXiv:2411.18261*, 2024.

[6] K. Segbenu *et al.*, "Reinforcement learning in dynamic pricing models for e-commerce: A systematic review," *Journal of AI Applications*, 2025.

[7] R. Kumar and A. Singh, "Leveraging reinforcement learning and bayesian optimization for dynamic pricing strategies in e-commerce," *International Journal of Data Science*, 2020.

[8] A. Gupta and R. Mehta, "Reinforcement learning in dynamic pricing models for e-commerce," *European Journal of Management and Business*, 2022.