Reinforcement Learning for Market Making: Dynamic Programming, Deep Q-Networks, and Soft Actor-Critic

Sam Layton Siddhant Sukhani Sondre Rogde

Abstract

This report studies market making as a finite-horizon control problem using Markov Decision Processes (MDPs), Dynamic Programming (DP), and Reinforcement Learning (RL). The methodology is organized as a comparison ladder: first solve a small discretized model exactly with DP, then train RL on the same task with a discrete state representation, and finally relax the representation to continuous state variables and, in the last experiment, continuous actions. Experiments 1-3 therefore compare exact planning against progressively more flexible learned policies, while Experiment 4 moves to a continuous state-action setting trained with Soft Actor-Critic (SAC). The central theme is that DP is the exact benchmark inside the discretized model, whereas RL becomes increasingly attractive as discretization becomes restrictive and continuous control becomes the more natural choice.

1 Introduction

A market maker repeatedly posts bid and ask quotes, earns spread revenue when orders arrive, and absorbs inventory risk after every fill. The core tradeoff is simple but dynamic: tighter quotes increase execution probability, while accumulated inventory makes future price moves more dangerous. This makes market making a natural finite-horizon MDP.

The methodological challenge is that the same control problem admits very different solution methods depending on how richly it is modeled. If the state, action, and transition law are all small and known, DP gives an exact solution. Once price, volatility, and eventually continuous quote placement and sizing are introduced, exact tabulation becomes less natural and RL becomes the practical alternative. Our report is built around that transition.

2 Phase 1: Problem Definition and Markov Description

2.1 Problem Specialization and Practical Relevance

We study single-asset market making. At each step the agent chooses bid and ask quotes, and in the most realistic version also chooses quote sizes. The objective is not just to maximize spread capture, but to do so while controlling inventory and limiting exposure to adverse price moves.

2.2 Three Versions of the Problem (Project Structure)

The report follows the course progression from **exact control to approximate control** through three nested versions of the same market-making problem. **Phase 1** formulates the problem as an MDP and clarifies the modeling levels considered. The **ideal full problem** represents the most realistic formulation, with a rich market state, continuous quote placement and sizing, and potentially additional latent signals such as order-flow imbalance or regime variables. The **Phase 3 course version** introduces a structured simplification in which the state includes inventory, price deviation, and stochastic volatility, ultimately leading to continuous control solved

with **SAC**. The **Phase 2 simplified version** reduces the environment to an inventory-only discrete model with constant volatility and an action grid small enough to be solved exactly using **dynamic programming (DP)**. Accordingly, **Phase 2** solves this discretized model exactly with DP, while **Phase 3** progressively enlarges the state description and compares three methodological levels whenever feasible: **DP on the discretized model**, **RL with the same discrete representation**, and **RL with continuous state inputs**. **Experiment 4** completes the progression by moving to **continuous state-action control with SAC**.

2.3 MDP Formulation

2.3.1 State

The state always includes the current inventory I_t. In richer versions we add the mid-price deviation and stochastic volatility:

$$s_t = \begin{cases} I_t, & \text{Phase 2 / Experiment 1,} \\ (I_t, S_t), & \text{Experiment 2,} \\ (I_t, S_t, \sigma_t), & \text{Experiment 3 \& 4} \end{cases} \tag{1}$$

In the DP baseline and in the discrete-RL baselines, these variables are placed on finite grids. In the continuous-RL baselines price and volatility are continuous, and in the SAC experiment (Experiment 4) all state variables are continuous.

2.3.2 Action

In Experiments 1-3 the action is a discrete pair of asymmetric bid and ask half-spreads: $a_t = (\delta_t^{\text{bid}}, \delta_t^{\text{ask}})$. The action grid is intentionally small enough for DP but still rich enough to allow meaningful inventory skew. In Experiment 4 the action becomes continuous and additionally includes quote sizes: $a_t = (\delta_t^{\text{bid}}, \delta_t^{\text{ask}}, q_t^{\text{bid}}, q_t^{\text{ask}})$, where both spread placement and order size are chosen continuously.

2.3.3 Transition Dynamics

The market maker posts bid $(S_t - \delta_t^{\text{bid}})$ and ask $(S_t + \delta_t^{\text{ask}})$ quotes around the mid-price S_t. Customer orders arrive as independent Poisson processes with spread-dependent intensity $\lambda(\delta) = A\exp(-k\delta)$, so the per-step fill probability on one side is $P(\text{fill} \mid \delta) = 1 - \exp(-\lambda(\delta))$. Tighter spreads therefore attract more flow but expose the agent to greater inventory risk.

The mid-price evolves according to adverse selection plus noise: $\Delta S_t = \eta\,(\mathbf{1}_{\text{ask fill}} - \mathbf{1}_{\text{bid fill}}) + \sigma_t\varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0,1)$. An ask fill (someone buying from us) signals informed buying pressure and tends to push the price upward; a bid fill has the opposite effect.

In Experiment 3 and the continuous SAC experiment, volatility evolves stochastically via an Ornstein-Uhlenbeck-style process ($\Delta\sigma_t = \kappa(\bar{\sigma} - \sigma_t) + \xi\varepsilon_t^v, \varepsilon_t^v \sim \mathcal{N}(0,1),$) with a lower floor to prevent degenerate volatility values.

2.3.4 Reward and Constraints

At each step, the reward combines realized spread capture, mark-to-market inventory P&L, and a quadratic inventory penalty:

$$r_t = \underbrace{\delta_t^{\text{bid}}\mathbf{1}_{\text{bid}} + \delta_t^{\text{ask}}\mathbf{1}_{\text{ask}}}_{\text{spread capture}} + \underbrace{I_t\Delta S_t}_{\text{inventory P\&L}} - \underbrace{\alpha I_t^2}_{\text{inventory penalty}}. \tag{2}$$

At the terminal step, we apply an additional penalty proportional to the magnitude of remaining inventory to discourage carrying large positions to the end of an episode. Inventory is bounded by $|I_t| \le I_{\max}$.

3 Phase 2: Dynamic Programming Solution

3.1 Simplified Phase 2 Model and DP Method

The canonical Phase 2 model is the inventory-only formulation used in Experiment 1. The state is $s_t = I_t \in \{-I_{\max}, \ldots, I_{\max}\}$, and volatility is fixed at σ_{base}. With $I_{\max} = 5$, the state space contains only 11 inventory levels, so the problem is small enough to solve exactly.

Because the experiments have a finite episode length and an explicit terminal inventory penalty, we solve the discretized model by finite-horizon backward induction with terminal condition $V_T(s) = -\lambda|I_T|$.

$$V_t(s) = \max_a \mathbb{E}[r_t(s, a, s_{t+1}) + \gamma V_{t+1}(s_{t+1}) \mid s_t = s, A_t = a] \tag{3}$$

This makes the DP policy time-dependent near the episode end, which is exactly what we want in a market-making problem with terminal inventory control.

3.2 RL Baselines and the Comparison Ladder

In Experiments 1–3 we compare three methods within a consistent framework: exact dynamic programming (DP) on the discretized finite-horizon MDP, a DQN trained on the same discretized state representation, and a DQN trained on continuous state features when richer variables are introduced. This setup isolates two issues. First, how much performance is lost when exact planning is replaced by a learned value-based policy on the same grid. Second, how the policy changes when discretization is removed and RL consumes raw continuous inputs. DP therefore provides the exact benchmark only for the discretized MDP, while the continuous RL model represents the more flexible approximation once state discretization becomes restrictive.

3.3 Experiment 1: Inventory-Only DP vs. RL

Experiment 1 is the cleanest validation case. The state contains only inventory, so there is no meaningful distinction between a discretized market state and a continuous market state. For that reason, Experiment 1 compares dynamic programming (DP) against the discrete DQN only and uses the result as the anchor for the later richer experiments. Figure 1 shows a qualitative pattern that persists throughout all experiments which is the inventory-skew logic: high positive inventory incentivizes selling, so the ask spread is kept tight (to attract buyers) while the bid spread is widened (to discourage further purchases). High negative inventory produces the opposite behavior. Note also that the reward distributions in Figure 1 for DP and discrete DQN are quite similar indicating that the RL-agent is able to approximate the exact solution.

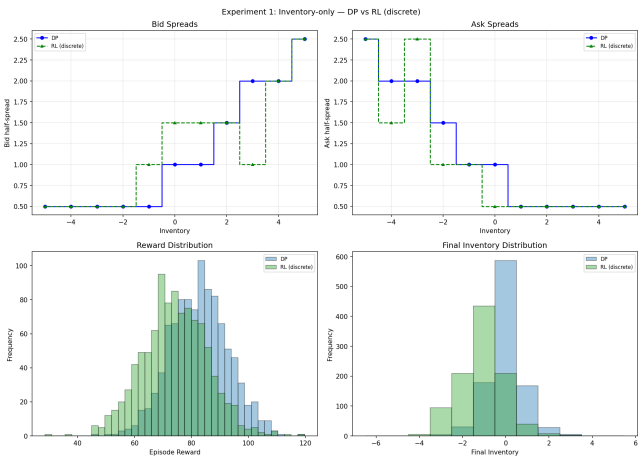


Figure 1: Experiment 1 comparison on the (inventory) state. The figure reports policy slices at the mid-price bin, reward distributions, and final inventory distributions for DP and RL with a discretized state.

4 Phase 3: Reinforcement Learning in Richer Environments

4.1 RL Setup

Experiments 2 and 3 use two DQN baselines with the same discrete action set but different state representations. The **discrete** DQN sees the same binned state that DP sees, so it is the closest learned approximation to the exact benchmark. The **continuous** DQN instead receives the underlying state variables as continuous features, avoiding state discretization while keeping the action space discrete. Experiment 4 then replaces the DQN family with **Soft Actor-Critic (SAC)**. This is the appropriate shift once the action itself becomes continuous: value-based methods are natural when the action set is enumerable, whereas actor-critic methods are better suited to learning in a continuous action space without an explicit discrete maximization over actions.

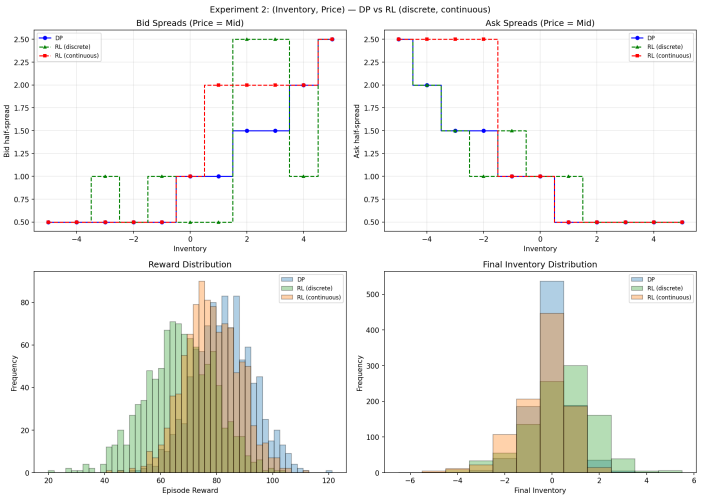


Figure 2: Experiment 2 comparison on the $(\text{inventory}, \text{price})$ state. The figure reports policy slices at the mid-price bin, reward distributions, cumulative P&L, and final inventory distributions for DP, RL with a discretized state, and RL with continuous state inputs.

4.2 Experiment 2: Inventory and Price

Experiment 2 extends the state to include both inventory and price deviation, following $s_t = (I_t, S_t) \in \{-I_{\max}, \ldots, I_{\max}\} \times \mathcal{G}_S$. The price grid has 21 bins over $[-10, 10]$, so the DP model contains $11 \times 21 = 231$ states. Experiment 2 provides the first direct test of the discrete-versus-continuous representation question in the RL setting.

Figures 2 and 3 show the first clear benefit of the comparison ladder. Inventory remains the dominant control signal, but the problem is now genuinely two-dimensional. Dynamic programming (DP) is still the exact benchmark on the binned model. From the reward and P&L distribution in Figure 2 we can see that the continuous DQN performs better than the discrete DQN which is to be expected as it is a richer model. The continuous DQN could potentially also be better than DP if the bins are too wide or do not cover the appropriate range.

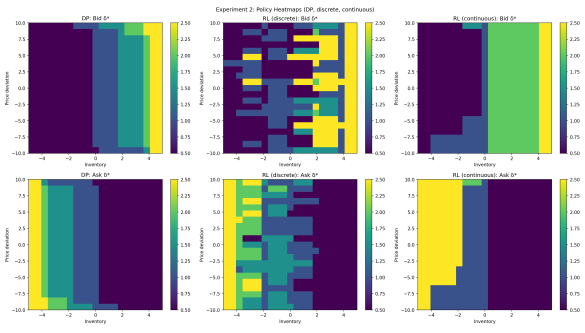


Figure 3: Experiment 2 policy heatmaps over inventory and price deviation. DP provides the exact benchmark on the grid, the discrete DQN shows how well RL can match that benchmark, and the continuous DQN illustrates how the policy changes once price is no longer discretized for the learner.

Price deviation itself has only a modest effect on policy behavior in our calibration. A deviation of ± 10 corresponds to a 10% move from the starting level of 100, which is already absorbed by the spread-capture and inventory P&L components. As a result, the optimal spreads change only slightly across price levels. The practical implication is that normalization becomes

more important when comparing assets with very different price scales; within a single asset calibrated around a fixed starting price, inventory remains the dominant dimension.

As seen in Figure 3, the continuous DQN yields a smoother policy because Q-values are modeled as a continuous function of the raw state rather than constant within each bin.

4.3 Experiment 3: Inventory, Price, and Volatility

Experiment 3 adds stochastic volatility to the state:

$$s_t = (I_t, S_t, \sigma_t) \in \{-I_{\max}, \dots, I_{\max}\} \times \mathcal{G}_S \times \mathcal{G}_\sigma. \tag{4}$$

In the implementation, the price grid has 21 bins and the volatility grid has 15 bins, producing a total discretized state space of $11 \times 21 \times 15 = 3{,}465$ states. Figures 4 and 5 largely replicate the findings from Experiment 2, but with one notable difference. Adding stochastic volatility makes the reward signal noisier, since inventory P&L now depends on the joint interaction of inventory, adverse selection, and a time-varying volatility. This is visible in the reward distributions: the gap between the continuous DQN and DP is wider in Experiment 3 than in Experiment 2.

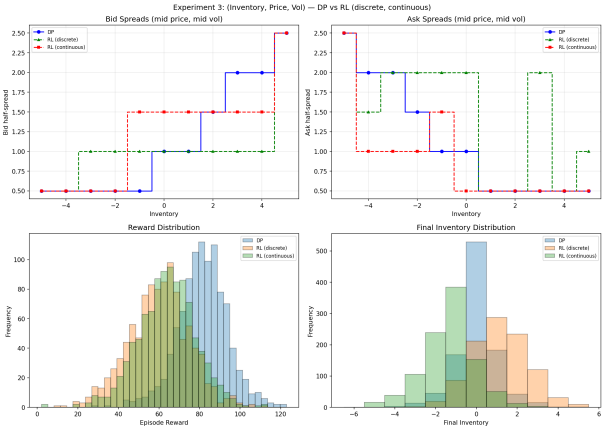


Figure 4: Experiment 3 comparison on the (inventory, price, volatility) state. The policy slice is shown at the mid-price and mid-volatility bin, together with reward and inventory diagnostics for DP, discrete-state RL, and continuous-state RL.

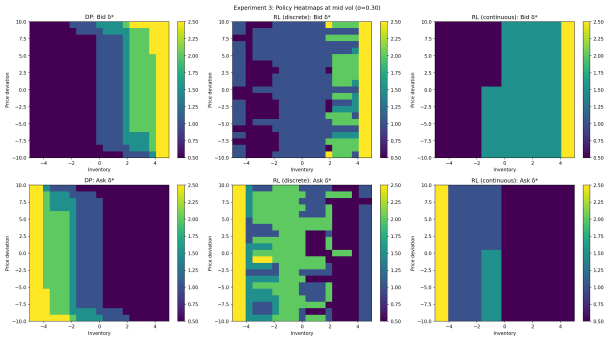


Figure 5: Experiment 3 policy heatmaps at the mid-volatility slice. The figure highlights how policy structure changes as we move from exact DP to discrete-state RL and then to continuous-state RL.

In Experiment 3 the continuous DQN barely achieves better rewards than discrete DQN, suggesting that the richer representation helps less when the reward process itself is noisy. The continuous DQN still learns a smoother policy than the discrete DQN which has a more fragmented policy. Overall the policy learned by continuous DQN more closely resembles the DP solution. Comparing the heatmaps of Experiments 2 and 3 (Figures 3 and 5), the overall policy structure is very similar: inventory remains the dominant control signal and the same directional skew is present across all three methods. The main visible difference is that the Experiment 3 policies exhibit somewhat smoother transitions across the state space, which is consistent with the noisier reward signal washing out sharper action boundaries during training.

4.4 Experiment 4: Continuous State-Action Control with SAC

The final experiment removes the grid-based action simplification and moves to a continuous state-action formulation. The state is $s_t = (I_t, S_t, \sigma_t) \in [-1, 1] \times \mathbb{R} \times \mathbb{R}_+$, where inventory

is continuous in $[-1, 1]$, price deviation is normalized by a fixed scale, and volatility is normalized by its long-run mean. The action includes both quote placement and quote size: $a_t = (\delta_t^{\text{bid}}, \delta_t^{\text{ask}}, q_t^{\text{bid}}, q_t^{\text{ask}})$, with continuous spread bounds $\delta \in [0.1, 3.0]$ and continuous quote-size bounds $q \in [0, 0.15]$. The environment keeps the same economic structure as the earlier experiments: spread-dependent Poisson fills, adverse selection, OU-style stochastic volatility, quadratic inventory penalty, and a terminal inventory penalty.

This is the closest setting to the full objective of the project. DP is no longer a natural comparator because both the state and action spaces are continuous, and a DQN is no longer the right tool because the action is not enumerable. We therefore use the Soft Actor-Critic algorithm.

The trained SAC policy shows positive but modest performance. Mean reward and episode P&L are positive, indicating profitability on average, though high reward variability keeps the Sharpe ratio low. Inventory remains well controlled, with mean absolute inventory of 0.255 and terminal inventory of 0.292, suggesting the policy has learned inventory mean reversion rather than simply maximizing fills.

Metric	Value
Mean reward	1.727
Reward standard deviation	4.369
Sharpe-like ratio	0.395
Mean episode P&L	2.183
Mean $\lvert I_t \rvert$	0.255
Mean $\lvert I_T \rvert$	0.292

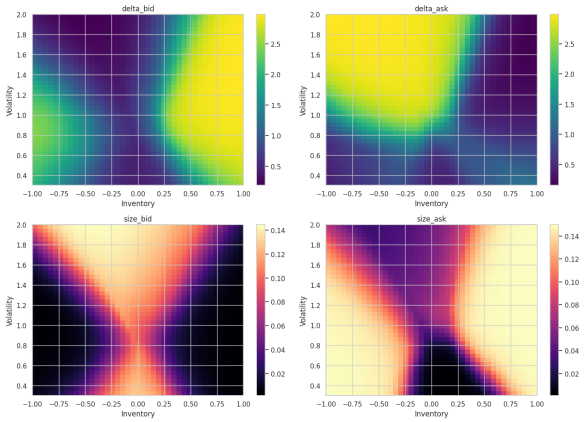
Table 1: Experiment 4 continuous-control SAC evaluation metrics, recorded over 500 episodes.

Inspecting the learned policy functions in Figure 6a reveals several additional structural patterns. First, ask quote size is approximately the mirror image of bid quote size across inventory levels and volatility levels: when the agent posts large ask quantities (eager to sell), it posts small bid quantities (reluctant to buy), and vice versa. This near-inversion is consistent with a policy that is simultaneously managing inventory in both directions. Second, the bid spread δ^{bid} displays a similar approximate inversion with bid quote size: when the agent wants to buy aggressively it both tightens the bid spread and increases bid size. This pattern is somewhat less pronounced for the ask dimension, and could be because inventory enters the reward symmetrically but the adverse-selection term introduces an asymmetry. Third, the effect of stochastic volatility is even more pronounced than in Experiment 3. In Figure 6b at high inventory and volatility the agent dramatically increases the bid spread and tightens the ask spread, becoming more urgently willing to unwind. The reverse holds at low inventory. Fourth, the agent also responds to price level: when prices are low relative to the starting level the agent is less willing to buy, reflecting that a further price decline would amplify mark-to-market losses on a long position.

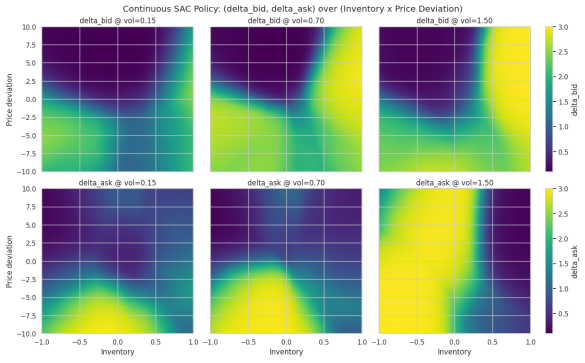
4.5 From DP to RL as Complexity Grows

The progression from Experiments 1 through 4 clarifies the methodological ladder. DP is appropriate when the model is known and the discretized state–action space is small enough to enumerate. Discrete-state RL serves as the closest learned approximation to that benchmark. Continuous-state RL then relaxes the representation while keeping the same economic task, and SAC finally removes the discrete action grid entirely.

The computational tradeoff appears clearly in the convergence times from Experiment 2. Exact DP converges in about 1.8 seconds, while the discrete DQN requires roughly 57.9 seconds and the continuous DQN about 52.0 seconds. DP is therefore extremely attractive when the discretized model remains small enough to solve directly. However, its computational cost grows rapidly as state dimensions and discretization resolution increase, whereas RL scales through sampled interaction rather than exhaustive enumeration. Experiment 3 already pushes this



(a) Experiment 4 policy heatmaps over inventory, price deviation, and volatility.



(b) Bid policies with varying volatility.

Figure 6: Results from Experiment 4.

tradeoff further, and Experiment 4 removes tabulation entirely by moving to continuous state and action spaces.

Table 2 summarizes how the role of each method evolves as model complexity increases. When the state space is small, dynamic programming (DP) remains the most direct solution because exact planning is feasible. Discrete-state RL then acts as the closest learned approximation to that benchmark.

Exp.	State	$\lvert\mathcal{S}\rvert$	DP tractable?	RL role
1	I	11	Yes	Validation
2	(I, S)	231	Yes	Comparison
3	(I, S, σ)	3,465	Slower	Preferred
4	cont. (I, S, σ)	∞	No	Required

Table 2: How the role of DP and RL changes as model complexity grows.

As additional state variables are introduced, discretization becomes increasingly restrictive and the computational burden of DP grows. Continuous-state RL therefore becomes the more scalable approach once the state space is too large to enumerate.

Across Experiments we observe a consistent performance ordering: DP $\geq$ continuous-state RL $\geq$ discrete-state RL. DP is the exact benchmark for the discretized MDP and therefore constitutes an upper bound for the discrete DQN on the same grid. The continuous-state DQN escapes that grid entirely and in practice sits between the two. Crucially, this ordering reverses when grids are very coarse: a coarse grid makes the DP solution less faithful to the underlying continuous task, so the continuous-state RL can outperform DP on true-environment metrics even though it cannot beat DP within the discretized model. The finer the grid, the more accurate DP becomes as a representation of the real task, and the more the ordering DP $\geq$ continuous RL is restored. However, the finer the grid the bigger the state space becomes as well, which makes DP computationally expensive.

5 Discussion

5.1 Main Lessons Across Experiments

Several qualitative patterns are consistent across the experiments. First, inventory mean reversion is the dominant structural feature of the policy throughout the project. Second, once price is added to the state, the representation itself matters: discrete and continuous RL are no longer

solving the same approximation problem. Third, once volatility is added, the reward becomes materially noisier and the gap between exact planning and approximate learning becomes more informative.

The project also shows that "comparing against DP" must be interpreted carefully. DP is an exact upper bound only for the same discretized MDP. The discrete DQN is therefore the closest learned comparator, while the continuous DQN and the SAC policy trade exact comparability for a representation that is closer to the eventual realistic control problem.

6 Conclusion

This project implements a coherent progression from exact planning to continuous-control RL. We begin with a small finite-horizon market-making MDP that can be solved exactly with DP, use that as the benchmark for discrete RL, then relax the state representation to continuous inputs, and finally move to continuous actions with SAC. The main takeaway is not that one method dominates everywhere, but that each method is most appropriate at a different level of model complexity: DP for exact small models, discrete RL for approximate learning under the same action grid, and continuous RL once the richer formulation becomes the real object of interest.

7 Appendix

7.1 DQN Agent Architecture

Network. For Experiments 1-3, both the online network Q_θ and the target network Q_{θ^-} are multilayer perceptrons with two hidden ReLU layers:

$$\text{Linear}(d_s, h) \to \text{ReLU} \to \text{Linear}(h, h) \to \text{ReLU} \to \text{Linear}(h, |\mathcal{A}|)$$

where d_s is the state dimension, $|\mathcal{A}| = n^2$ is the number of joint bid/ask spread actions, and the hidden width h is chosen per experiment (larger in the richer state settings).

Double DQN. The online network selects the greedy next action; the target network evaluates it:

$$y_t = r_t + \gamma \cdot Q_{\theta^-}\Big(s_{t+1},\ \arg\max_a Q_\theta(s_{t+1}, a)\Big)$$

This decouples action selection from value estimation, reducing overestimation bias.

Loss and replay. Huber (Smooth L_1) loss is minimized over mini-batches drawn from an experience replay buffer of capacity 50 000:

$$\mathcal{L}(\theta) = \text{SmoothL}_1(Q_\theta(s, a),\ y_t)$$

Optimizer and target network. Parameters are updated with Adam. The target network is updated by soft Polyak averaging after every gradient step:

$$\theta^- \leftarrow \tau\,\theta + (1 - \tau)\,\theta^-, \qquad \tau = 0.005.$$

Exploration and training schedule. Action selection uses ε-greedy exploration with linear decay from $\varepsilon = 1.0$ to $\varepsilon = 0.02$. As is standard in DQN, gradient updates are delayed until a minimum number of transitions has been collected (`learning_starts`), after which replay-buffer samples are used for off-policy learning. In the richer experiments, the implementation can also use prioritized replay to place extra weight on informative transitions.

Inventory masking. At inventory boundaries ($|I| = I_{\max}$), Q-values for actions that would deepen the position are set to $-\infty$ before the arg max, forcing the agent to quote on the unwinding side only.

7.2 SAC Agent Architecture

Policy and critic. Experiment 4 uses Stable-Baselines3 SAC with the default multilayer-perceptron policy (`MlpPolicy`). The actor outputs a stochastic continuous policy over the four-dimensional action vector

$$(\delta^{\text{bid}}, \delta^{\text{ask}}, q^{\text{bid}}, q^{\text{ask}}),$$

while the critic learns soft Q-values for continuous state-action pairs.

State and action. The observation is the three-dimensional continuous feature vector

$$(I_t,\ (S_t - S_0)/10,\ \sigma_t/\bar{\sigma}),$$

and the action bounds are

$$\delta^{\text{bid}}, \delta^{\text{ask}} \in [0.1, 3.0], \qquad q^{\text{bid}}, q^{\text{ask}} \in [0, 0.15].$$

Reward and environment. The reward is the same economic objective used throughout the report:

$$r_t = \text{spread P\&L} + \text{inventory P\&L} - \alpha I_t^2,$$

with an additional terminal inventory penalty $-\lambda|I_T|$. Fills are generated from spread-dependent Poisson intensities, price dynamics include adverse selection plus noise, and volatility follows an OU-style process.

Training setup. The reported run uses a replay buffer of size 100,000, learning rate 3×10^{-4}, batch size 128, soft target update $\tau = 0.005$, discount factor $\gamma = 0.99$, automatic entropy tuning, and a total budget of 20,000 environment steps. An evaluation callback tests the deterministic policy every 10,000 steps and stores the best checkpoint.

Interpretation. Unlike DQN, SAC does not need a discrete action grid and therefore scales naturally to the continuous quote-placement and quote-sizing problem. This is the main methodological reason it is the right final-stage algorithm for the project.