

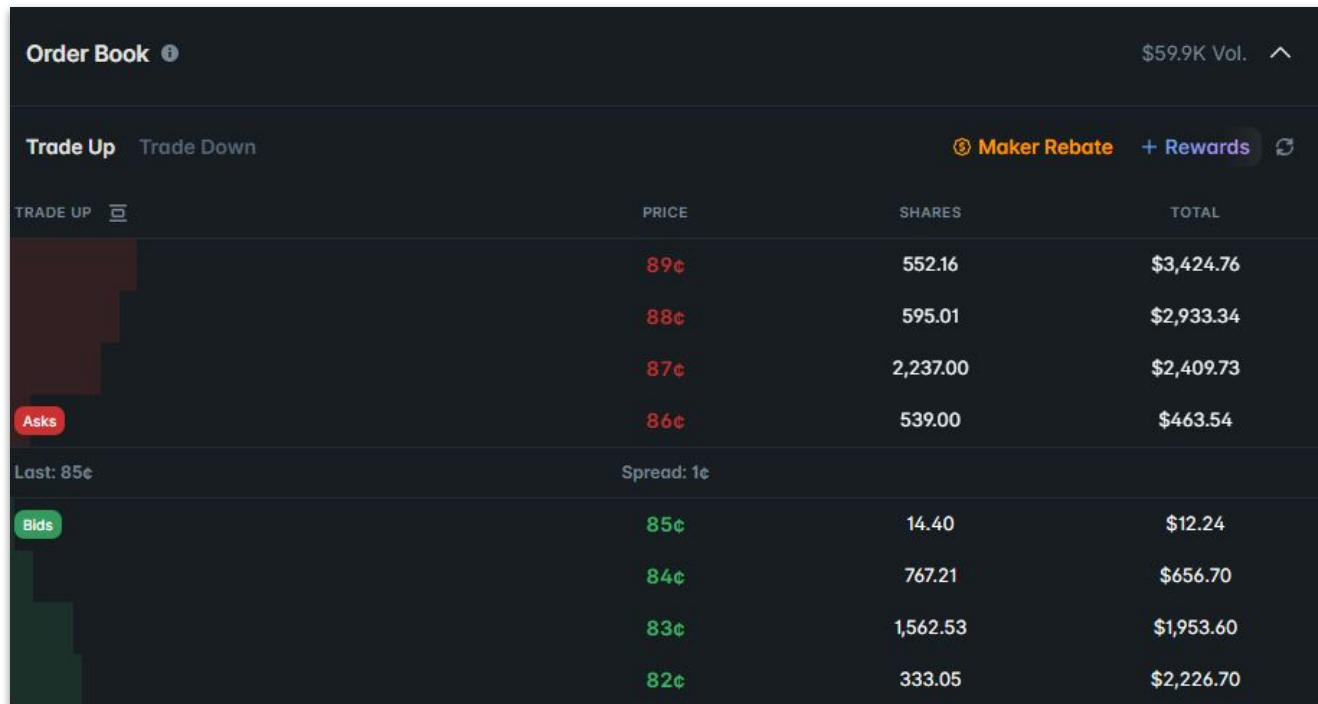
CME 241 - Optimal Market Making

TOMMASO SINI & ANTHONY VIVERITO

Binary Prediction Markets: Setting

- A binary prediction market trades contracts that settle to either \$1 or \$0 depending on whether an event occurs.
 - Contract payoff at resolution: $X_T \in \{0, 1\}$, $X_T = \begin{cases} 1 & \text{if the event occurs} \\ 0 & \text{otherwise} \end{cases}$
 - Because the payoff is binary, the market price can be interpreted as a market-implied probability: $p_t \approx \mathbb{P}(X_T = 1 \mid \mathcal{F}_t)$
- Polymarket BTC 1-Hour Market (Will Bitcoin be higher in 1 hour?)
- Two complementary contracts exist: YES/NO
 - $p_t^{YES} + p_t^{NO} \approx 1$
- We trade only the YES token as a simplified abstraction.
- Market makers quote around the mid-price:.
 - $m_t = \frac{\text{bid}_t + \text{ask}_t}{2}$
- Market spread: $s_t = \text{ask}_t - \text{bid}_t$
- A market maker profits by earning the spread.

The Orderbook



Market-Making Problem / Goal

- Goal: Learn an approximately optimal market-making policy for the BTC 1-hour binary market.
- A market maker continuously posts quotes to:
 - a. provide liquidity / capture bid-ask spread / manage inventory risk
- We want a policy that decides:
 - a. Quote width / order sizing / adjustment according to inventory imbalance
- Our approach:
 - a. start with a simplified model of the market
 - b. solve a benchmark policy with Dynamic Programming
 - c. train an RL agent in the same simulated environment
 - d. compare RL vs DP using Monte Carlo simulation

Data, & Limitations

- Source: Polymarket websocket stream; we use 500 ms snapshots of the order book (best bid/ask).
- Markets: Full day of 1-hour BTC up/down markets. We train on these markets and evaluate on an out-of-sample market so simulation is not on training data.
- Mid filter: We only use snapshots with $\text{mid} \in [0.10, 0.90]$.
 - Cuts compute: when mid is in the tails, the market is mostly inactive for long stretches.
- State: We discretize into 16 mid bins and 10 spread bins and estimate $P(m', s' | m, s)$ from the training data.
- Single day, specific BTC 1h markets; order-book microstructure and edge cases are simplified.

Assumptions & Models

- Trading abstraction
 - a. We trade only the yes token (buy on one side, sell on the other). In practice you'd market-make by buying both yes and no; Polymarket doesn't allow shorting, so real MM is "buy yes + buy no," not "short yes."
- Fill model
 - a. We use a parametric fill probability per side:
 - b. $P(bidfill) = (q_b > 0) \cdot \min(0.3 \cdot e^{-0.5 \cdot w_b}, 1)$
 - c. $P(askfill) = (q_a > 0) \cdot \min(0.3 \cdot e^{-0.5 \cdot w_a}, 1)$
- We use an arbitrary λ for inventory penalty; penalizes large $|inventory|$ so the policy doesn't accumulate too much risk.
- $P(m', s' \mid m, s)$; no flow or other order-book structure in the state.

Markov Decision Process

- State
 - a. $S = (m, s, i), \quad m \in \{0, \dots, 15\}, \quad s \in \{0, \dots, 9\}, \quad i \in \{-40, -35, \dots, 40\}$
- Action
 - a. $A = (w_b, w_a, q_b, q_a), \quad w_b, w_a \in \{0, 1, 2\}, \quad q_b, q_a \in \{0, 5, 10, 15\}$
- Inventory
 - a. $i' = \text{clip}(i + F_b q_b - F_a q_a, -40, 40)$
- Reward
 - a. $R = F_b q_b \cdot \text{edge}_{bid} + F_a q_a \cdot \text{edge}_{ask} - \lambda (i')^2$
- Edge Definitions
 - a. $\text{edge} = \frac{\text{spread}}{2} + w \cdot \text{tick}, \quad \text{tick} = 0.01$
- Transitions and discount
 - a. $(m', s') \sim P(\cdot \mid m, s), \quad \gamma = 0.95, \quad \lambda > 0$

Training Simulator

- At each step, the simulator state is: $s_t = (m_t, s_t, i_t)$
 - a. m_t = discretized mid-price bin, s_t = discretized spread bin, i_t = inventory
- The agent chooses one discrete action: $a_t = (w_b, w_a, q_b, q_a)$
 - a. buy/sell sizes: $q_b, q_a \in \{0, 5, 10, 15\}$
 - b. buy/sell widths: $w_b, w_a \in \{0, 1, 2\}$
- Buy and sell fills as stochastic events with probabilities: $p_b = \mathbf{1}_{\{q_b > 0\}} \min(\text{BASE_FILL} e^{-\alpha_{\text{fill}} w_b}, 1)$
 - a. $\text{BASE_FILL} = 0.3$, $\alpha_{\text{fill}} = 0.5$
- Inventory then updates as: $i_{t+1} = \text{clip}(i_t + q_b \mathbf{1}_{\{\text{buy fill}\}} - q_a \mathbf{1}_{\{\text{sell fill}\}}, -I_{\text{max}}, I_{\text{max}})$
- Next market state is sampled from the empirical training transition matrix:
 - a. $(m_{t+1}, \sigma_{t+1}) \sim P_{\text{train}}(m', \sigma' \mid m_t, \sigma_t)$
- Gymnasium: defines environment

Dynamic Programming

- What it is
 - a. Optimal policy from value iteration on the same MDP. We are using value_iteration_result from the RL-book. State includes mid, spread, and inventory; reward trades off edge and inventory risk.
- How we get it
 - a. Build the full finite MDP (~1000 states, ~150 actions). For each state–action pair enumerate transitions and rewards from the empirical transition model and the fill model, then solve with value iteration.
 - b. $V^*(s), \pi^*(s) \ (m', s') \sim P(m', s' \mid m, s)$
- What it does
 - a. Chooses bid/ask width and size at each step to balance spread capture and inventory risk; when inventory is long it biases toward asks, when short it biases toward bids.
 - b. $a = (w_b, w_a, q_b, q_a)$
- Why we use it
 - a. Dynamic programming gives the exact optimal policy for the same MDP, allowing a direct benchmark comparison.
 - b. π_{DQN} vs π^*

Algorithm Choice: Dueling DQN

- Why this algorithm is a good fit
 - a. Our action space is discrete, not continuous
 - b. We want a policy that chooses among 144 quoting actions
 - c. DQN is a natural fit for finite action selection problems
 - d. Dueling DQN improves learning when many actions have similar value.
- That matches our environment, since the action is chosen from a finite grid: $a_t = (w_b, w_a, q_b, q_a)$
 - a. Total number of Actions: $|\mathcal{A}| = 3 \times 3 \times 4 \times 4 = 144$
- DQN is appropriate because it learns the optimal action-value function:
 - a. $Q^*(s, a) = \max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^T \gamma^t r_t \mid s_0 = s, a_0 = a \right]$
- The optimal policy is then: $\pi^*(s) = \arg \max_a Q^*(s, a)$

How Dueling DQN learns

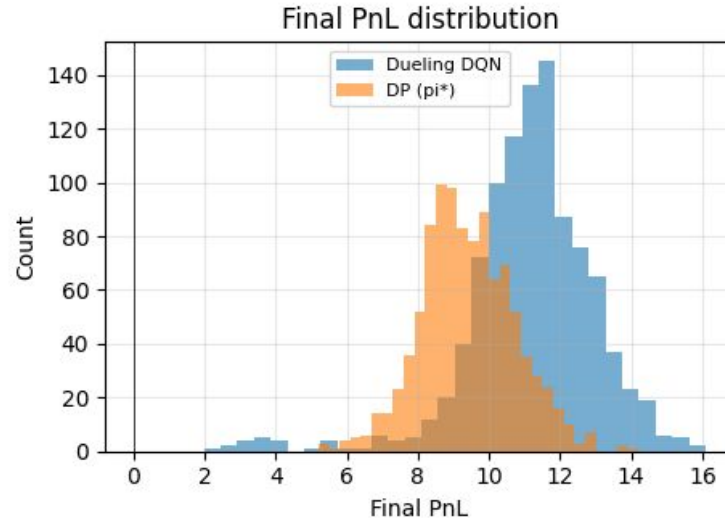
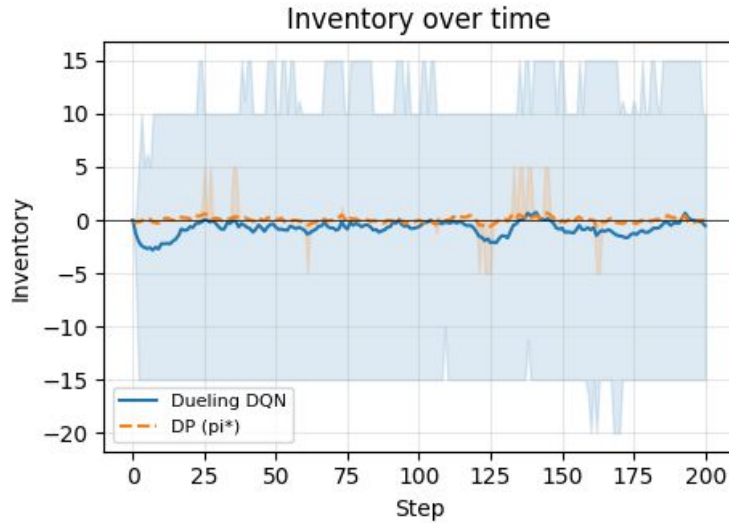
- DQN learns an approximation to the action-value function: $Q(s, a; \theta)$
 - a. $Q(s, a) = V(s) + A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a')$
- Instead of storing a full Q-table, neural network maps state to estimated Q-values for all actions.
- At each step, the agent observes a transition: (s_t, a_t, r_t, s_{t+1})
- It then forms the temporal-difference target:
 - a. $y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-)$
 - b. θ^- denotes the target network parameters
- The network is updated to reduce prediction error:
 - c. $L(\theta) = (y_t - Q(s_t, a_t; \theta))^2$
- It then adjusts $Q(s_t, a_t)$ toward:
 - d. $r_t + \gamma \max_{a'} Q(s_{t+1}, a')$
- Over many episodes, this allows the network to learn which quoting actions lead to higher long-run reward.
- Ray's RLlib

DQN Implementation

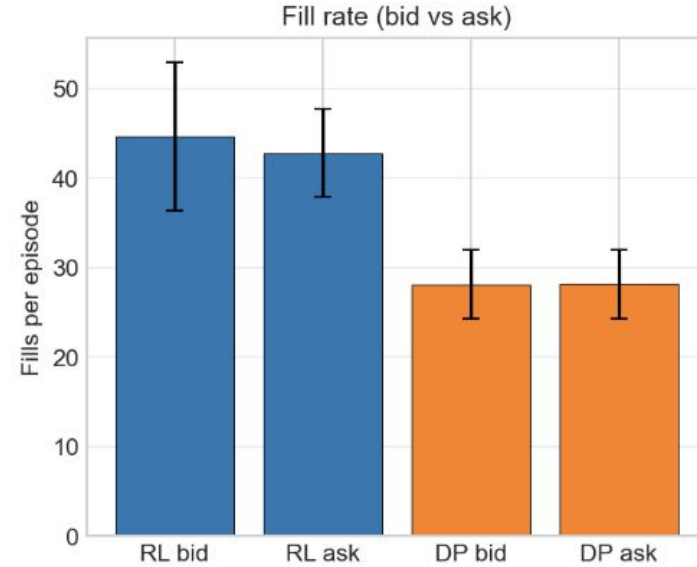
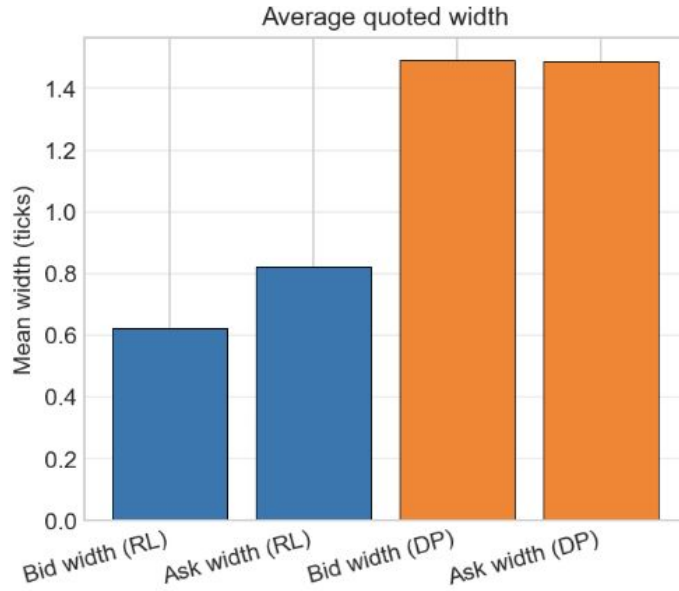
- The environment observation is a normalized version of the simulator state:
 - a. $o_t = \left[\frac{m_t}{M-1}, \frac{\sigma_t}{S-1}, \frac{i_t}{I_{\max}} \right]$
- The network receives this 3-dimensional observation and outputs Q-values over all actions:
 - b. $Q(o_t, a; \theta), \quad a \in \mathcal{A}, \quad |\mathcal{A}| = 144$
- The simulator then generates:
 - stochastic buy/sell fills / updated inventory / next market state / reward
- Dueling DQN decomposes the Q-function into:
 - a state-value term
 - i. $V(s)$
 - and an advantage term
 - ii. $A(s, a)$
- Combined as: $Q(s, a) = V(s) + A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a')$

Results

- Results from 1000 Monte Carlo episodes on an out-of-sample market

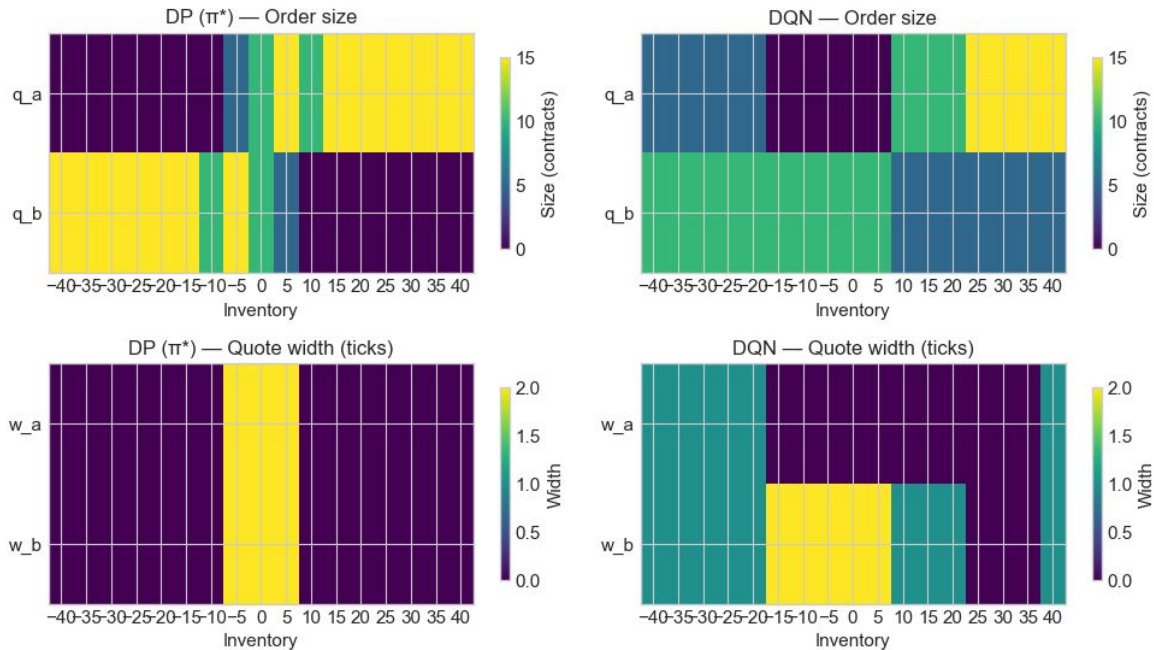


Results



Results

Policy at (m=9, s=0)



Extensions & Future Steps

- Robustness
 - a. Use more markets and longer history to improve $(P(m',s'|m,s))$ and reduce overfitting.
 - b. Train across regimes
- Simulator & microstructure
 - a. Replace the reduced-form fill model with an orderbook-based simulator (depth, queue position).
 - b. Add microstructure state (order flow, imbalance, recent trades) and model adverse selection.
- Broader RL scope
 - a. Richer state/action (time-to-resolution, volatility; finer/continuous actions).
 - b. Multi-market or multi-asset policies