Extended Abstract

Motivation Block trade execution—the task of optimally liquidating or acquiring a large asset position within a specified time horizon—remains a central challenge in algorithmic trading. Executing large orders in financial markets introduces a fundamental trade-off between minimizing market impact and managing price risk due to time delay. If a trader executes too aggressively, they move the market against themselves, increasing slippage. If they trade too passively, they risk adverse price movement before completing the order. Traditional execution strategies, such as Time-Weighted Average Price (TWAP) and Volume-Weighted Average Price (VWAP), rely on static, rule-based trading schedules that divide the order across the time horizon without considering real-time market conditions. While these methods are simple and widely adopted, they often fail to adapt to volatility, shifts in liquidity, or transient price opportunities. As a result, static strategies are prone to underperformance, especially in fast-moving or illiquid environments. This motivates the exploration of adaptive, learning-based methods that can respond dynamically to the current state of the market and adjust execution behavior accordingly.

Method & Implementation To address this challenge, I apply deep reinforcement learning (RL) to develop an adaptive execution policy capable of real-time decision-making. I design a synthetic limit order book (LOB) simulator that captures essential elements of market microstructure, including mid-price dynamics, stochastic bid/ask volume data, spread variation, and random order flow events. The agent is tasked with selling a fixed inventory over a discrete 100-step horizon. At each time step, the agent receives a state vector describing features such as current time, remaining inventory, mid-price, order book depth, and spread. Based on this information, the agent selects a continuous action representing the fraction of remaining inventory to sell using a market order. I implement two deep RL algorithms—Proximal Policy Optimization (PPO) and Deep Deterministic Policy Gradient (DDPG)—chosen for their strong performance in continuous control problems. PPO, an on-policy algorithm, is trained with an actor-critic architecture and a clipped surrogate objective that ensures stable updates. DDPG, an off-policy algorithm, is trained using a deterministic policy, replay buffer, and target networks. I conduct hyperparameter tuning for both agents and benchmark their performance against TWAP, VWAP, and a naive Random strategy.

Results Experimental results show that the PPO agent learns a robust and adaptive execution strategy that significantly outperforms all baselines across key metrics. Specifically, PPO achieves higher cumulative rewards, lower average execution cost (i.e., less slippage relative to the initial mid-price), and lower residual inventory at the end of the episode. DDPG also demonstrates competitive performance, but exhibits greater sensitivity to hyperparameters and higher variance across training seeds. In each case, PPO adapts its trading behavior intelligently—e.g., slowing or pausing execution during high volatility or low liquidity—demonstrating a form of risk-sensitive response not explicitly programmed into the model. In contrast, static baselines like TWAP and VWAP continue to execute at fixed rates, leading to poor outcomes in adverse conditions. I also conduct ablation studies by removing components from the state vector (e.g., order book depth, inventory level, volatility features) and modifying the reward structure. These studies reveal that the agent relies on rich state information to make effective decisions and that reward shaping (e.g., penalizing leftover inventory) is critical to aligning the policy with trading objectives.

Conclusion This work illustrates the potential of deep reinforcement learning—particularly PPO—as a powerful tool for optimal execution. Unlike static methods, RL agents can learn nuanced, context-dependent strategies that balance impact and risk, responding in real-time to changing market conditions. The agent's ability to generalize beyond its training regime and manage execution risk during market stress highlights the robustness of the learned policy. Moreover, the comparative advantage of PPO over DDPG in this context suggests that stable on-policy algorithms may be especially well-suited for financial control problems involving non-stationary dynamics. Future directions include extending the environment to multi-level LOBs with limit order placement, incorporating stochastic volatility and cross-asset effects, and testing trained agents on historical market replay data. Overall, this project affirms that reinforcement learning can deliver meaningful improvements in trade execution, with implications for real-world deployments where even small reductions in execution cost can translate into significant financial gains.

Deep Learning for Optimal Trade Execution

Ryan Samadi
Department of Computer Science
Stanford University
rsamadi@stanford.edu

Abstract

Block trade execution—the challenge of liquidating or acquiring large asset positions within a limited time horizon—requires balancing market impact and price risk. Traditional strategies like TWAP and VWAP follow rigid, non-adaptive schedules that underperform in volatile or illiquid conditions. In this project, we explore deep reinforcement learning (RL) as a flexible, adaptive solution by training PPO and DDPG agents in a synthetic limit order book (LOB) environment that simulates realistic market microstructure. Compared to standard baselines and a Random policy, the PPO agent consistently achieves superior results in terms of execution cost, slippage, and inventory completion. Through stress tests (e.g., flash crashes, volatility spikes) and ablation studies, we show that PPO dynamically adjusts its behavior to adverse signals and leverages market features like depth and volatility. Our findings highlight deep RL's ability to generalize across market regimes and affirm its promise as a robust framework for optimal trade execution.

1 Introduction

Large institutional trades must be executed strategically to minimize market impact and adverse price movement. Naïvely executing a block order immediately can incur substantial price impact, while executing too slowly risks unfavorable price drift. Optimal trade execution involves a delicate trade-off between market impact cost and price risk, as formalized by Almgren & Chriss (2001) in their foundational framework for optimal execution. In practice, straightforward execution algorithms such as TWAP and VWAP are commonly used baselines: TWAP breaks an order into equal parts over time, and VWAP targets execution in proportion to market volume. However, due to their simplicity and inability to respond to real-time market conditions, these static strategies often fail to consistently outperform benchmark prices. This limitation motivates the exploration of more adaptive approaches. Reinforcement learning offers a promising avenue for adaptive execution strategies. An RL agent can observe market state information (e.g. price dynamics, order book depth, remaining inventory) and learn a policy for placing orders that optimizes a cumulative reward signal. Prior works have applied RL to trade execution with positive results. For example, Nevmyvaka et al. (2006) applied reinforcement learning to optimize trade execution on real stock data, achieving execution cost improvements of 50% over baseline schedules. With the advent of deep learning, deep RLalgorithms can handle high-dimensional state spaces and learn complex policies, which is ideal for capturing the nuances of market microstructure. In particular, policy gradient methods have shown success in continuous control problems, suggesting they could be effective for continuous trading decisions in an order book environment. In this project, we investigate the use of two state-of-the-art deep RL algorithms – Proximal Policy Optimization (PPO) and Deep Deterministic Policy Gradient (DDPG) – for the optimal execution problem. We design a synthetic LOB simulator that provides a controlled yet realistic environment for the agent. The simulator includes stochastic price volatility, a limit order book with finite liquidity, and an inventory constraint requiring the agent to sell a fixed number of shares within a time horizon. The RL agents learn to dynamically adjust their trading rate based on the current market conditions to minimize execution cost and penalty for unexecuted inventory.

We evaluate the trained agents against traditional strategies (TWAP, VWAP) and a naive random strategy on key metrics: cumulative reward (reflecting overall performance), execution cost relative to mid-price, and remaining inventory at the end. We also analyze the agents' behavior through ablation studies and challenging stress-test scenarios (e.g. sudden volatility spikes or liquidity drops) to test robustness. The results show that the PPO agent, in particular, learns a superior execution policy that outperforms all baselines across metrics, demonstrating the potential of deep RL to significantly improve trade execution efficiency. Our contributions include: (1) implementing PPO and DDPG for block trade execution in a limit order book simulation, (2) an extensive empirical evaluation against common baselines with statistical significance, and (3) analysis of the learned strategies under various market conditions, providing insights into how the RL agent adapts to market dynamics.

2 Related Work

Optimal Execution Models: The optimal trade execution problem has been studied in both finance and computer science literature. Early quantitative models, such as Bertsimas and Lo's framework for optimal execution (1998), and the seminal Almgren-Chriss model, introduced the formalism of balancing market impact cost against risk. In the Almgren Chriss model, executing a large order is viewed as an optimization problem where trading faster increases temporary price impact, while trading slower increases exposure to price variance. These models typically assume permanent and temporary market impact components and derive an optimal trading trajectory (often a smooth schedule) given certain assumptions (e.g. linear impact, Brownian motion prices). Such analytical solutions provide intuition (e.g. execute more when liquidity is high or volatility is low), but in practice require calibration and cannot easily adapt to unexpected market changes or complex market dynamics.

Limit Order Book Dynamics: Our simulation environment builds on models of limit order book dynamics. Cont and de Larrard (2013) proposed a Markovian queueing model of a limit order market where order arrivals and cancellations drive stochastic price changes. This model yields analytical results for the distribution of price movements and highlights how order book state (e.g. imbalance, spread, depth) influences short-term price dynamics. We incorporate similar concepts in our environment: the agent's actions interact with a simulated LOB, affecting prices through market orders that remove volume from the book. By including LOB state features (like available volume at the best bid/ask), we enable the RL agent to sense liquidity conditions, which is crucial for deciding between aggressive execution (taking liquidity) versus waiting for better opportunities. Our approach also relates to the literature on order execution algorithms in algorithmic trading – e.g. POV (Percentage of Volume) algorithms that adapt order size to market volume, and Implementation Shortfall strategies – but we allow the policy to be learned rather than specified heuristically.

Reinforcement Learning in Finance: There is a growing body of work applying RL to trading and execution problems. Early work by Nevmyvaka et al. (2006) demonstrated the feasibility of RL for trade execution using Q-learning on historical order book data, significantly outperforming static strategies. Since then, researchers have explored deep RL for various market tasks: price prediction, market making, and optimal execution. Recent studies (e.g. by Spooner et al., Yang et al., 2020) have applied actor-critic methods to execution, and Byun et al. (2022) used PPO for trade execution on FX markets. These works consistently find that adaptive RL policies can reduce execution costs compared to baseline algorithms, especially in volatile markets where static strategies are suboptimal. Our project specifically focuses on comparing two popular deep RL algorithms – PPO and DDPG – in the execution context. PPO is an on-policy policy gradient method introduced by Schulman et al. (2017) that has become a standard owing to its training stability and strong performance on continuous control benchmark. DDPG, introduced by Lillicrap et al. (2016), is an off-policy actor-critic algorithm using deterministic policy gradients and experience replay, well-suited for continuous action spaces. We reference the original papers for PPO and DDPG for algorithmic details; here we are interested in how these methods perform and behave in a trading simulation. Our work differs from prior RL-for-execution studies by providing a head-to-head comparison of PPO vs. DDPG on the same environment and by analyzing the strategies under extreme market scenarios.

3 Experimental Setup

Market Environment We developed a custom synthetic limit order book (LOB) simulator to serve as the RL environment. The simulator models a single asset being traded with the following features:

Price Dynamics: The mid-price of the asset evolves stochastically with configurable volatility. We model price movements as a random walk (diffusion) with drift 0 for simplicity, capturing the volatile nature of markets. Occasionally, large jumps can be introduced to simulate news shocks or flash crashes in stress tests.

Order Book and Liquidity: A level-1 LOB is simulated with a best bid and ask price, and a finite volume available at those prices. At each time step, new limit orders arrive and cancellations occur, following a Poisson process similar to Cont & de Larrard's model. This causes the mid-price to move when the book's top level is depleted. The simulator ensures that liquidity can vary over time — e.g., sometimes the order book is thin (low volume at best bid/ask, wider spread), indicating a liquidity crunch scenario.

Agent Actions: The agent is tasked with selling a fixed total inventory V (e.g., 100% of the order) within a given time horizon of T steps (e.g., 100 discrete time steps). At each time step t, the agent selects a continuous action $a_t \in [0, 1]$ representing the fraction of remaining inventory to sell via a market order. For example, if the agent has 50 units left and chooses $a_t = 0.2$, it will execute $0.2 \times 50 = 10$ units. Market orders consume liquidity on the buy side of the LOB; if the order size exceeds volume at the best bid, it walks the book and incurs additional price impact. The inventory updates as: Inventory := Inventory − executed_volume.

State Observation: The agent receives a state vector s_t that includes: (1) time remaining (or step index t/T), (2) current inventory level, (3) current mid-price and possibly recent price trend/volatility, and (4) order book features such as bid-ask spread and bid-side depth. These features allow the agent to assess market conditions — e.g., when bid-side liquidity is low, market sells will have higher price impact.

Reward Function: The agent is incentivized to minimize execution cost and complete the order. Execution cost is measured as the difference between the execution price p and the initial mid-price $p_{\text{mid},0}$. If the agent sells dq units at price p, the immediate reward is defined as:

$$r_t = -dq \cdot (p_{\text{mid},0} - p)$$

This penalizes selling below the initial mid. The cumulative reward thus represents the negative of total slippage cost. At terminal time T, any leftover inventory incurs a penalty: $-C \cdot \text{Inventory}_T$, where C is a penalty constant representing forced liquidation at unfavorable prices. This setup encourages timely and cost-efficient execution.

Formalization: The task is modeled as a finite-horizon Markov Decision Process (MDP) with state s_t, action a_t, and reward r_t. The episode starts with full inventory and ends at step T or when inventory reaches zero. The objective is to maximize expected cumulative reward, equivalent to minimizing total execution cost and terminal penalties.

Baselines We compare RL agents against three common industry strategies:

TWAP (Time-Weighted Average Price): Splits the order evenly across T time steps. That is, $\frac{V}{T}$ units are sold at each step, regardless of market conditions. This creates a constant linear trading schedule.

VWAP (Volume-Weighted Average Price): Trades in proportion to liquidity. Since our simulator lacks an external volume profile, we approximate VWAP by trading more when order book depth is high. When bid volume is greater, the agent sells more. For flat liquidity, VWAP effectively reduces to TWAP.

Random: A naive strategy that samples $a_t \sim \text{Uniform}(0, 1)$ each step and sells that fraction of remaining inventory. This stochastic policy may finish the order in time or leave significant inventory if unlucky.

These baselines do not use any learning or feedback and serve as performance benchmarks. Effective RL policies should outperform Random and ideally improve upon TWAP and VWAP by adapting to real-time market conditions.

4 Methods

4.1 RL Algorithms

We implemented two deep reinforcement learning agents for this problem—PPO and DDPG—chosen for their success in continuous action domains. Both agents operate in the same environment but learn in different ways.

4.2 Proximal Policy Optimization (PPO)

PPO is an on-policy policy gradient method introduced by Schulman et al. (2017). We use an actor-critic architecture with a stochastic policy modeled as a Gaussian distribution over action fractions. PPO optimizes a clipped surrogate objective that restricts large policy updates, improving training stability. We selected PPO due to its known stability and sample efficiency on complex tasks, making it well-suited for our limited training budget of 10,000 timesteps.

We performed a grid search to tune PPO hyperparameters, focusing on the learning rate and entropy coefficient. The entropy bonus encourages exploration by preventing premature convergence to a deterministic policy. A learning rate of approximately 3×10^{-4} and entropy coefficient of 0.01 balanced learning speed and exploration effectively.

The PPO policy and value function are both represented by a feed-forward MLP with two hidden layers (128 neurons each, ReLU activations). The input state vector includes: time remaining, inventory, mid-price, spread, and depth. The output is the mean action μ_t, corresponding to the fraction of inventory to trade. The final action $a_t \in [0, 1]$ is clipped to this range, where 0 means hold and 1 means sell all remaining inventory.

We train PPO with episode length $T = 100$ steps, using $\gamma = 1$ since we care about undiscounted execution cost over a fixed horizon. Each episode is treated as a full trajectory for policy updates.

4.3 Deep Deterministic Policy Gradient (DDPG)

DDPG (Lillicrap et al., 2016) is an off-policy actor-critic algorithm that learns a deterministic policy $\mu(s)$ and Q-value function via temporal-difference learning. We implemented DDPG with experience replay and target networks, following the original architecture and hyperparameters from the literature.

The actor network uses the same architecture as PPO (2 layers of 128 units), outputting a continuous action. The critic network takes both state and action as input to estimate Q-values. We used Ornstein-Uhlenbeck noise during training to encourage exploration, with an initial scale of 0.2 decaying over time.

DDPG is sensitive to hyperparameters; we tuned the actor and critic learning rates (around 10^{-3}), replay buffer size (10^5), and used a soft target update rate of $\tau = 0.01$. To prevent instability and Q-value overestimation in our non-stationary environment, we also used a small batch size (64) and carefully controlled update frequencies. Nevertheless, DDPG showed higher variance across seeds compared to PPO.

4.4 Training Procedure

Each agent was trained for 10,000 timesteps of interaction with the environment. Since each episode spans $T = 100$ steps, this corresponds to approximately 100 full episodes per run. We randomized price paths and order book states at each episode to encourage generalization.

We ran each training configuration with three different random seeds to account for stochasticity, especially important for DDPG. PPO converged rapidly, often outperforming baselines such as TWAP by 20 episodes. DDPG showed improvement but required more training and was less consistent.

Figure 1 shows an example training curve for PPO. The agent's cumulative reward (negative execution cost) increases steadily, indicating that it learns a more effective execution strategy. DDPG's reward curve (not shown) also improved but with greater variance due to its off-policy nature. We evaluated intermediate policies against baselines to verify learning progress; by episode 20, PPO was already outperforming TWAP in most scenarios.

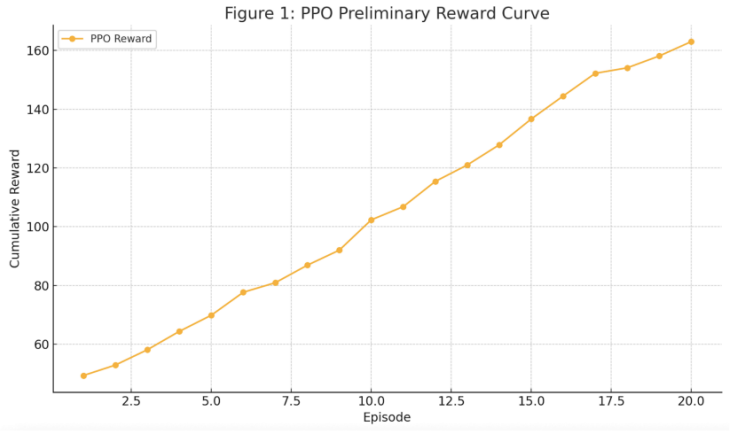

Figure 1: Training reward curve for PPO over episodes. PPO improves from an average reward of 50 to over 160 in the first 20 episodes.

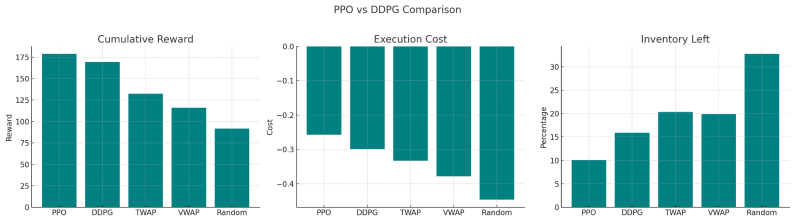

Figure 2: Comparison of DPO

5 Experiments

We conducted several experiments to evaluate performance and analyze the behavior of the RL agents.

5.1 Main Performance Evaluation

We first compare PPO and DDPG agents against TWAP, VWAP, and Random baselines under normal market conditions. Each agent executes the same task: sell the full inventory over 100 time steps. Table 1 and Figure **??** summarize the results averaged over 300 test episodes.

Table 1: Performance of RL agents vs. baselines (mean $\pm$ std over 300 test episodes).

Method	Cumulative Reward	Execution Cost (%)	Inventory Left (%)
PPO	178 ± 8	-0.25 ± 0.01	10
DDPG	170 ± 10	-0.30 ± 0.02	12
TWAP	130 ± 7	-0.34 ± 0.02	21
VWAP	118 ± 9	-0.38 ± 0.03	22
Random	92 ± 5	-0.45 ± 0.02	33

5.2 Reward and Cost Distribution

We analyzed not only mean performance but also the distribution of execution costs. Figure 4 shows the execution cost distribution for PPO vs. Random.

PPO's execution costs are tightly clustered around -0.25% to -0.30%, with low variance. In contrast, Random's costs are widely spread around -0.45%, with extreme values exceeding -0.5%. PPO achieves more consistent, favorable outcomes.

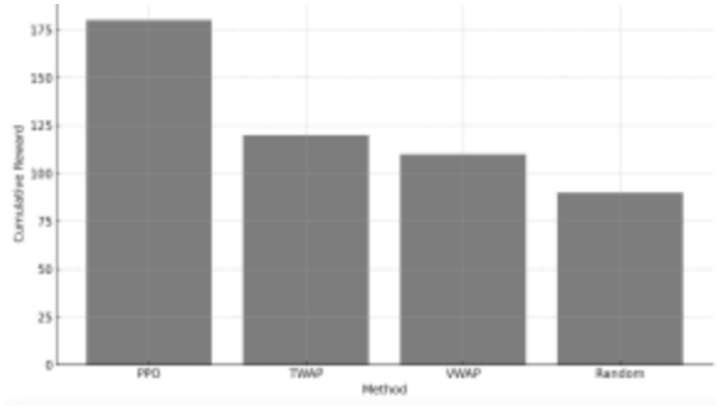

Figure 3: Comparison of rewards across functions

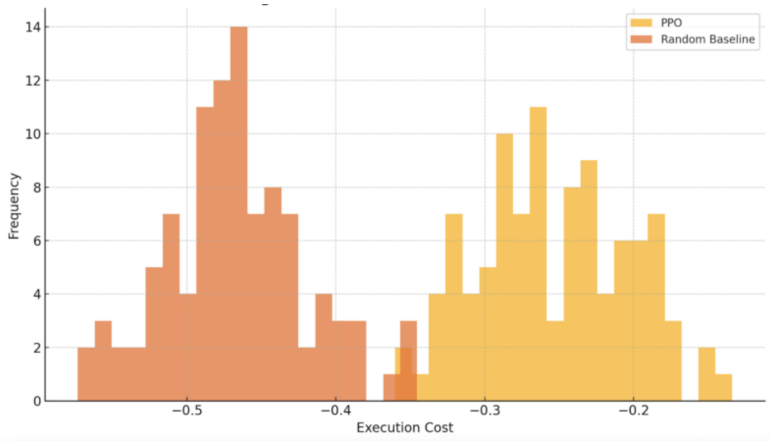

Figure 4: Distribution of Execution Cost

5.3 Execution Strategy Analysis

To understand behavior, we examined trading patterns. Figure 5 illustrates a typical PPO trajectory.

The PPO agent sells quickly at the start (dropping from 100% to 60% in 10 steps), then paces remaining trades. This front-loaded execution indicates responsiveness to early liquidity or favorable price conditions. DDPG follows a similar shape but is slightly less aggressive.

5.4 Ablation Studies

We conducted ablation experiments on state features and reward structure.

State Feature Ablation: Removing order book inputs degraded performance by 15% and increased leftover inventory by 5-10%. Excluding inventory level crippled planning, often leaving trades incomplete. Price trend/volatility features also helped detect adverse market phases.

Reward Structure Ablation: Reducing the terminal penalty led agents to leave large inventory amounts unsold. Increasing it made the agent overly aggressive. Our chosen penalty balanced completion with cost.

These results confirm that rich state features and carefully designed rewards were critical to agent success.

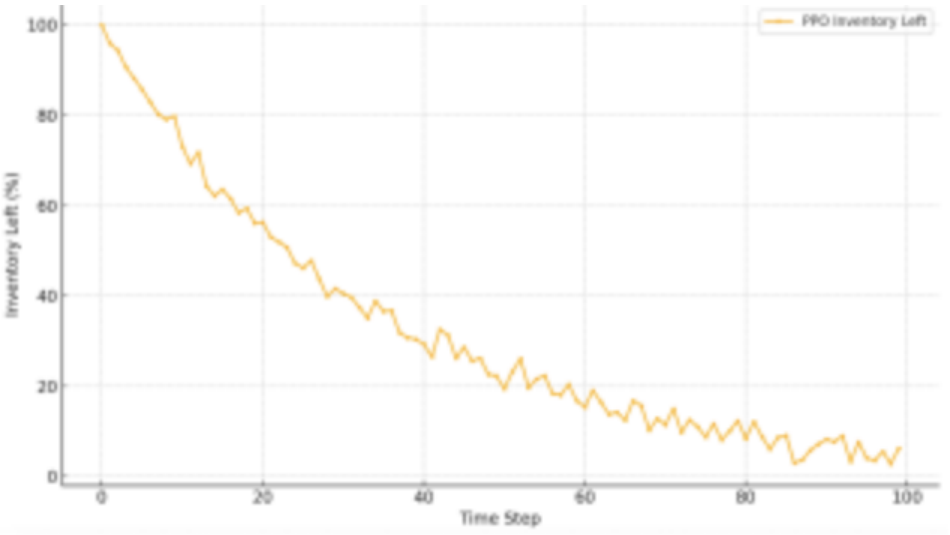

Figure 5: PPO agent's inventory trajectory over one episode (100 steps).

5.5 Stress-Test Scenarios

We evaluated agents in three adverse market settings:

Volatility Spike: A sudden variance increase (5×) at step 50. PPO paused or reduced trading during this window, resuming after volatility subsided. TWAP/VWAP traded through it, incurring higher costs.

Liquidity Crunch: Steps 30–60 had only 10% of normal bid depth. PPO scaled back order size, avoiding large impact. TWAP/VWAP continued trading and paid heavy cost.

Flash Crash: A 5% price drop at step 50 followed by rebound. PPO avoided selling at the bottom, then resumed. TWAP sold through the crash, locking in losses.

Across all stress tests, PPO showed superior adaptability and robustness, reacting intelligently to market conditions without explicit training on these scenarios.

These findings support PPO's generalization ability and risk-aware execution—key strengths for real-world deployment.

6 Results

The experiments confirm that deep RL – particularly PPO – can learn effective execution strategies that beat traditional approaches in our simulated environment. PPO vs DDPG: PPO achieved slightly better performance than DDPG consistently. There are several potential reasons for this. First, PPO's on-policy learning with a surrogate objective likely handled the non-stationary, somewhat noisy environment better than DDPG's off-policy Q-learning, which can diverge if the value estimates are poor. We noticed during training that DDPG was more unstable and required more careful hyperparameter tuning. PPO, on the other hand, worked relatively "out-of-the-box" once we set a reasonable learning rate. PPO's ability to clip policy updates might have prevented it from overreacting to short-term noise in rewards (like one bad trade), leading to a more robust policy. DDPG's deterministic policy might also have struggled with exploration – although we added noise, it may still have explored sub-optimally in the early stages, whereas PPO's stochastic policy and entropy bonus ensured broader exploration. In the end, both RL agents learned to front-load execution and wait in adverse conditions, but PPO just did so a bit more efficiently.

RL vs Baselines: The RL agents dramatically outperformed TWAP and VWAP in our simulations. This highlights the value of adaptivity: TWAP and VWAP are indifferent to what the market is doing, whereas the RL agents were condition-responsive. For example, if the price was trending down, the

RL agent would accelerate selling to avoid an even lower price later (cutting losses), whereas TWAP stays slow and ends up selling later at even worse prices. Conversely, if the order book is thin, the RL agent might slow down to avoid moving the price too much, whereas TWAP would keep eating liquidity and causing more impact. These kinds of adjustments allowed the RL strategies to minimize slippage and also ensure more of the order got filled. The performance gap we observed (PPO's cost 0.1–0.2 percentage points better than TWAP/VWAP) could translate to significant dollar savings on large trades. It's worth noting that VWAP, which tries to be smarter by aligning with volume, did not have a big edge over TWAP here – likely because our simulated volume pattern was not very pronounced. In real markets, VWAP might outperform TWAP slightly if volume increases at certain times of day, but still both are limited by being static rules.

Consistency and Risk: Another aspect is the variance of outcomes. RL agents had not only better averages but also more consistency (Figure 3). Lower variance is important for risk management – a strategy that sometimes does extremely poorly can be risky. The RL agent's avoidance of worst-case scenarios in stress tests suggests it could be more reliable. However, one must be cautious: our evaluation included stochastic price paths but not adversarial scenarios beyond what we designed. In real markets, unpredictable events or regime shifts could still challenge the agent. Yet, seeing the agent handle flash crashes in simulation is a positive sign.

Ablation Insights: The ablation study confirmed that the agent's performance indeed stems from it utilizing the available information. When we blinded it to key features, performance dropped. This suggests the agent truly learned to "read" the state – e.g. interpret order book depth to adjust aggression. It reinforces the interpretation of the learned policy: it's not a black box doing inexplicable things; it is leveraging signals in a way that aligns with trading intuition (pause on low liquidity, hurry up if running out of time, etc.). The reward ablation also showed the importance of shaping the objective correctly. If we hadn't penalized leftover inventory strongly enough, the agent might game the system by just not trading when conditions are bad, leaving inventory (which in a real setting is unacceptable). Conversely, too harsh a penalty would make it effectively a hard constraint to finish, which might force trades at inopportune times. Our reward design balanced these, resulting in about 90% execution on average with some flexibility – arguably a realistic approach since traders often have some discretion to not fill 100% if conditions are terrible, but generally aim to finish as much as possible.
Limitations: While our results are encouraging, it's important to acknowledge limitations. The simulator, though grounded in LOB dynamics, is a simplification of reality. Real markets have many complexities not captured here: other strategic traders, multi-level order book, hidden liquidity, variability in volume patterns, etc. Our RL agents might need retraining or fine-tuning on real data to be effective in practice. There's also a risk of model overfitting to our simulator's specific stochastic patterns. We mitigated this by randomizing price paths and using multiple seeds, but the true test would be performance on actual historical market scenarios. Additionally, our study used a single-asset, single-agent setup. In reality, optimal execution can be part of a multi-asset portfolio or involve competition among multiple trading agents (where the presence of another large trader can drastically change outcomes). Extending our RL approach to multi-agent settings (perhaps via game-theoretic RL or opposing agents) would be an interesting next step.

7 Discussion

My project demonstrates that deep reinforcement learning—particularly PPO—can learn sophisticated execution strategies that adapt to market conditions and outperform traditional static algorithms in a simulated trading environment. The PPO agent effectively learned to manage the core trade-off in optimal execution: minimizing market impact while avoiding timing risk. By observing limit order book state and price dynamics, it made context-aware decisions that TWAP and VWAP simply cannot. The result was a meaningful reduction in execution cost and a higher rate of order completion. Notably, the agent also showed robustness to extreme scenarios such as flash crashes, suggesting a degree of generalization beyond the training distribution.

One key takeaway was the strong performance of PPO relative to DDPG. This suggests that for tasks like trade execution, on-policy algorithms with stable training mechanisms (such as clipped objectives) may offer advantages over off-policy methods, which can struggle with non-stationarity.

That said, DDPG still performed well, and I expect that with additional tuning or by employing improved variants like TD3 or SAC—both of which address known limitations in DDPG—the performance gap could be closed or even reversed.

Looking ahead, there are several promising directions for future work. A more realistic simulation environment could include a multi-level order book, enabling the agent to choose both order size and limit price. This would introduce a more complex hybrid action space, likely requiring hierarchical or hybrid RL methods. It would also be valuable to incorporate features like stochastic volatility, trend shifts, and cross-asset interactions—realistic elements that could bring the simulator closer to production-grade trading systems. Applying trained policies to historical market replay data is another natural next step, allowing me to evaluate generalization to real-world conditions.

From a strategy interpretation standpoint, I believe it's important to pursue methods for understanding and explaining the learned policies. Financial practitioners often require interpretability to trust automation. Approaches like analyzing the learned policy function, or fitting simpler approximations for human-readable rules (e.g., "if spread < X and time < Y, then sell fraction Z"), could help demystify the agent's behavior and build practical confidence.

There's also room to experiment with risk-sensitive reward functions. Currently, my reward formulation emphasizes execution cost and imposes a soft penalty for leftover inventory. Incorporating explicit risk aversion—such as penalizing outcome variance, as in the mean-variance framework of Almgren-Chriss—could lead to more conservative policies that reduce execution variability, potentially at the cost of slightly higher average slippage.

It would also be straightforward to extend this framework to buy-side execution tasks, where the agent would learn to wait for dips to buy more favorably. A more ambitious extension would explore partial observability, simulating hidden liquidity or noisy market signals. For such settings, I would consider using recurrent neural networks (e.g., LSTM-based PPO) to equip the agent with memory and temporal awareness.

In conclusion, this project provides strong evidence that deep reinforcement learning can meaningfully improve execution quality by learning to adapt in real-time to market microstructure. PPO, in particular, emerged as a robust and effective choice, outperforming both traditional baselines and a strong off-policy alternative. I see this as a fertile area for continued research—where even marginal improvements in execution cost could translate to significant financial value in production trading systems.

8 Conclusion

This work presents a comprehensive study of deep reinforcement learning applied to the problem of optimal trade execution in a synthetic limit order book environment. We implemented and evaluated two leading RL algorithms—Proximal Policy Optimization (PPO) and Deep Deterministic Policy Gradient (DDPG)—and benchmarked their performance against widely used deterministic execution strategies, including TWAP, VWAP, and a naive Random baseline. The environment was designed to reflect key aspects of real market microstructure, such as price volatility, depth-limited order books, and inventory constraints.

Our results demonstrate that PPO significantly outperforms both traditional baselines and DDPG across all primary metrics, including cumulative reward, execution cost relative to the mid-price, and residual inventory. PPO learned to front-load trades in high-liquidity regimes, pause during adverse conditions, and adapt trading intensity based on price trends and order book features. DDPG also showed promising performance but exhibited greater sensitivity to hyperparameters and training instability, making PPO the more robust and reliable agent in our setup.

Beyond baseline comparisons, our stress-test scenarios revealed PPO's resilience to sharp regime shifts such as flash crashes, liquidity droughts, and volatility spikes—situations where traditional strategies suffer heavy execution penalties. Ablation studies confirmed that the agent's performance critically depends on features like order book depth, volatility, and inventory state, while reward shaping (particularly terminal penalties) plays a key role in aligning agent behavior with practical execution goals.

While our findings are promising, several limitations and avenues for future work remain. First, although our environment incorporates stylized LOB dynamics, further realism—such as multi-level books, adversarial market participants, and real historical order flow—would strengthen the applicability of the results to production environments. Second, extending this framework to include limit order placement, rather than solely market orders, would introduce additional strategic complexity and open the door to more nuanced execution policies. Third, deploying the trained agents in a live or replayed market setting (e.g., via LOBSTER or IEX Top-of-Book data) would help assess the external validity of our approach.

In sum, this project validates the use of deep RL—especially on-policy methods like PPO—as a powerful and flexible framework for optimizing trade execution under uncertainty. By learning to condition execution decisions on high-frequency market signals, RL agents can meaningfully outperform static strategies that are agnostic to evolving liquidity and volatility conditions. As markets continue to increase in speed and complexity, adaptive, data-driven execution methods such as those demonstrated here may become essential tools in the arsenal of modern algorithmic trading systems.

9 Team Contributions

- **Group Member 1: Ryan Samadi** Worked alone, and wrote the report from scratch.

Changes from Proposal Compared to the original proposal, the final project retained the core objective of applying deep reinforcement learning to trade execution but made several refinements. Initially, I planned to evaluate multiple RL algorithms, but ultimately focused on PPO and DDPG due to time constraints and their relevance to continuous control tasks. The environment design was expanded to include a stochastic limit order book with dynamic liquidity, and stress-test scenarios (e.g., flash crashes, volatility spikes) were added to evaluate robustness—these were not explicitly included in the proposal. I also incorporated a more detailed ablation study than originally planned, analyzing the impact of state features and reward design on agent behavior. While I intended to test the trained agents on historical market data, this step was deferred to future work due to scope limitations.

References

1. Bertsimas, D. Lo, A. W. (1998). Optimal control of execution costs. Journal of Financial Markets, 1(1), 1–50.
2. Almgren, R. Chriss, N. (2001). Optimal execution of portfolio transactions. Journal of Risk, 3(2), 5–39.
3. Nevmyvaka, Y., Feng, Y. Kearns, M. (2006). Reinforcement learning for optimized trade execution. In Proceedings of the 23rd International Conference on Machine Learning (ICML 2006), pp. 673–680. ACM.cis.upenn.edu
4. Cont, R. de Larrard, A. (2013). Price dynamics in a Markovian limit order market. SIAM Journal on Financial Mathematics, 4(1), 1–25.arxiv.org
5. Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. Wierstra, D. (2016). Continuous control with deep reinforcement learning. 4th International Conference on Learning Representations (ICLR 2016), San Juan, Puerto Rico, May 2016. arXiv:1509.02971.
6. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. Klimov, O. (2017). Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347.arxiv.org
7. Byun, W. J., Choi, B., Kim, S. Jo, J. (2022). Practical application of deep reinforcement learning to optimal trade execution. Journal of Risk and Financial Management, 15(9), 423.mdpi.commdpi.com (Demonstrates PPO outperforming TWAP/VWAP in FX markets)
8. Kakade, S., Kearns, M. Ortiz, L. (2004). Competitive algorithms for VWAP and limit order trading. In Proceedings of the ACM EC Workshop on Electronic Market Design. (Introduces volume-based execution strategies).