Project Report: Deep Reinforcement Learning for Robust Optimal Execution (BTC/USDT)

Cameron Mouangue, Guicheney Jacques
M1 Applied Mathematics and Statistics
Institut Polytechnique de Paris

December 18, 2025

1 Introduction

1.1 Motivation: Institutional Liquidity Friction in Crypto Markets

Executing institutional-sized liquidation orders in cryptocurrency markets is fundamentally constrained by market microstructure. When a participant must sell a large inventory (e.g., on the order of \$60M+ notional BTC/USDT), the execution itself becomes a dominant source of cost: consuming order book depth induces adverse price moves, widens effective spreads, and generates significant Implementation Shortfall. In other words, the act of trading changes the price at which the trade can be completed.

This project studies the optimal execution problem under realistic crypto conditions: high intraday volatility, volatility clustering, and rapidly changing liquidity. In such an environment, a purely mechanical schedule can be systematically exposed to tail events occurring within the liquidation window.

1.2 Problem Statement: Impact–Time Risk Trade-off

The core objective is to sell a large fixed initial inventory Q_0 over a finite horizon of T discrete time steps (here Q_0= 1000 BTC & $T = 240$, i.e., 4 hours at 1-minute granularity). The trader faces a classical trade-off:

The Cost of Urgency (Market Impact): Selling too fast minimizes the time the agent holds the asset, but forces it to accept worse prices. By dumping large amounts at once, the agent overwhelms available buyers, effectively driving the price down against itself.

The Cost of Waiting (Time Risk): Selling too slowly minimizes the self-inflicted price drop, but keeps the agent exposed to the market for longer. If the market naturally crashes during this waiting period, the agent loses value regardless of how carefully it sells.

A standard non-learning baseline for this task is TWAP (Time-Weighted Average Price). TWAP acts as a naive "autopilot" that simply slices the total order into N equal pieces and sells one piece at every time step, regardless of market conditions.

Because TWAP is "blind" to changing data distributions (regime shifts), the research question is:

> *Can a learning-based execution policy dynamically adapt its urgency to market regimes so as to minimize total execution cost while substantially reducing tail risk relative to TWAP?*

1.3 Approach and Contributions: From DQN to PPO

We approach this as a sequential decision-making problem and train Reinforcement Learning (RL) agents in a custom execution environment featuring a microstructure-consistent impact model and a stress-testing simulator. The development proceeded in two stages:

1. Stage 1 (Baseline RL): Double DQN. We first implemented a value-based agent (Double DQN). This stage was essential to validate the environment mechanics (impact computation, liquidation constraints, end-of-horizon “fire sale” logic) and to explore reward shaping under stable, clipped signals.
2. Stage 2 (Final RL): PPO Actor–Critic. We then transitioned to Proximal Policy Optimization (PPO) to address the limitations of value-based bootstrapping in a noisy, non-stationary financial setting and to obtain more stable learning dynamics. PPO also provides a direct stochastic policy representation, which proved more robust for long-horizon execution under heavy-tailed risk.

The final thesis of the report is that the best-performing policy behaves like a risk-insurance execution engine: it accepts a small opportunity cost in benign regimes (selling slightly faster than TWAP) in order to dramatically reduce crash exposure (tail risk / CVaR) when regimes deteriorate.

2 Theoretical Framework & Microstructure Validation

2.1 Optimal Execution and Implementation Shortfall

We consider the liquidation of a parent order of size Q_0 over a fixed horizon T. Let P_0 be the arrival price at the beginning of the execution window. At each time $t \in \{0, \dots, T-1\}$, the agent chooses an action that determines the executed quantity $q_t \geq 0$ (subject to remaining inventory constraints), producing revenue $q_t P_t^{exec}$.

The classical benchmark for execution quality is Implementation Shortfall (IS), defined as the difference between the hypothetical mark-to-market value at arrival and the realized revenue:

$$IS = Q_0 P_0 - \sum_{t=0}^{T-1} q_t P_t^{exec}. \tag{1}$$

Minimizing IS is equivalent to maximizing realized revenue, but crucially under execution prices that depend on the agent's own trading intensity.

2.2 Temporary Market Impact and Parameter Calibration

A critical component of the simulation is the modelling of temporary market impact—the cost of demanding immediate liquidity. We adopt the "Square-Root Law," a universally observed empirical regularity where instantaneous price impact scales sub-linearly with trade size. Specifically for Bitcoin, *Donier and Bonart (2015)* analyzed over one million metaorders and confirmed that the impact exponent converges to approximately 0.5, validating the square-root assumption for digital assets.

In our environment, the impact fraction I_t^{temp} is modeled as:

$$I_t^{temp} = \lambda_0 \cdot \underbrace{\left(1 + \frac{\sigma_t^{real}}{\sigma_t^{roll}}\right)}_{\text{Liquidity Stress Factor}} \cdot \underbrace{\left(\frac{q_t}{\overline{V}_t}\right)^{0.5}}_{\text{Square-Root Law}}, \qquad I_t^{temp} \leq I_{\max} \tag{2}$$

The execution price is then:

$$P_t^{exec} = P_t \left(1 - I_t^{temp}\right), \tag{3}$$

where q_t is the trade size, $\overline{V}_t$ is the rolling average volume, σ_t^{real} acts as a short-horizon realized volatility estimate, computed from the recent returns over a short period L :

$$\sigma_t^{real} = \sqrt{\sum_{i=1}^{L} r_{t-i}^2}\,.$$

and σ_t^{roll} as the rolling average volatility. Computing these "rolling" quantities requires a calibration step. This volatility term is crucial: purely volume-based models fail during "liquidity black holes" (Flash Crashes). By scaling impact with realized volatility, the model captures the empirical reality that liquidity resilience evaporates during stress events, as documented in recent institutional benchmarks by *Talos Trading (2025)*.

Calibration of the Liquidity Coefficient (λ_0)

The parameter λ_0 controls the baseline "stiffness" of the market. Rather than selecting an arbitrary value, we performed a structural calibration to align the model with 2024 institutional market microstructure data. We selected $\lambda_0 = 0.003$ based on the impact heatmap shown in Figure 1, which validates the parameter against two distinct execution regimes:

1. The Micro-Trade Regime (Spread Proxy): For a small trade (1 BTC), the model yields an impact cost of approximately 1.9 basis points (bps). This aligns precisely with the effective bid-ask spread on high-liquidity venues like Binance. Research by *Kaiko (2024)* indicates that BTC/USDT spreads compressed to the 1.0–3.0 bps range in the 2024 ETF era. A lower λ_0 would unrealistically imply execution inside the spread.
2. The Block-Trade Regime (Liquidity Depletion): For a "Whale" trade (100 BTC, approx. $9M), the model projects an impact of 19.0 bps. This is consistent with institutional Transaction Cost Analysis (TCA). Recent benchmarks by *Talos Trading (2025)* indicate that institutional parent orders with participation rates above 5% typically incur implementation shortfall in the 10–30 bps range due to order book depth consumption.

This calibration ensures the agent faces a realistic "physics engine": execution is cheap in quiet markets but becomes expensive during volatility spikes. This prevents the RL policy from exploiting an overly permissive simulator.

2.3 Risk, Tail Events, and Why "Tracking Error" Matters

In crypto markets, the probability of a large adverse price move over a multi-hour window is not negligible. Consequently, minimizing expected IS alone can lead to strategies that look optimal on average but exhibit severe tail risk. A practical institutional objective is often closer to wealth preservation than pure profit (alpha) generation.

This motivates a risk-aware formulation. Rather than rewarding the agent for directional gains (which are largely dominated by market drift), we design objectives that penalize deviation from the arrival reference. Conceptually, the

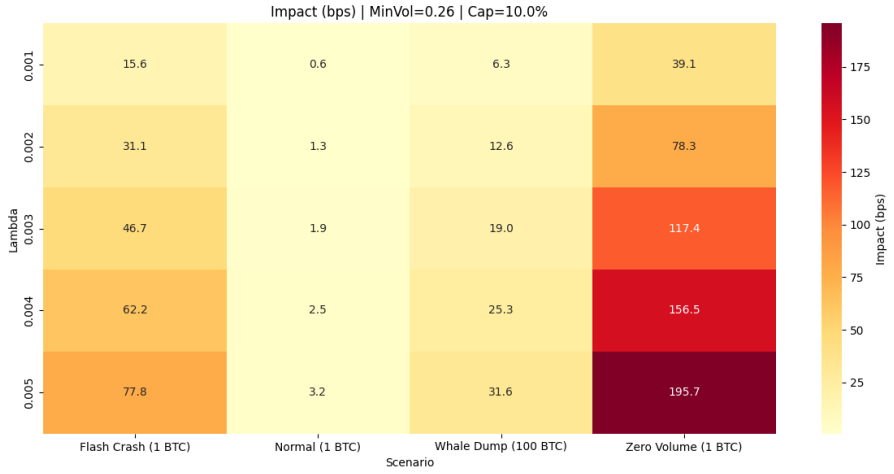

Figure 1: **Calibration of Market Impact Parameter** ($\lambda_0 = 0.003$). The heatmap illustrates the resulting impact cost (in basis points) across varying trade sizes and volatility regimes. The values illustrate a realistic progression from spread-like costs ($\sim$1.9 bps) for small trades to prohibitive slippage ($\sim$46.7 bps) during simulated Flash Crash events, forcing the agent to learn prudent liquidity timing.

policy should behave as a "smart insurer": sell sufficiently early to reduce exposure to crash risk, but not so aggressively that impact costs dominate.

2.4 Reinforcement Learning Formulation (MDP)

We model execution as a finite-horizon Markov Decision Process:

- State s_t: a feature vector encoding remaining inventory, time-to-go, liquidity ratio, price ratio, and recent volatility dynamics.
- Action a_t: a discrete choice among percentages of remaining inventory to sell (0%, ..., 100%).
- Transition: driven by either real market data or a calibrated GARCH simulator for stress training.
- Reward r_t: designed to reflect execution quality and risk, normalized by the initial portfolio value.

3 Data Analysis & Environment Design

3.1 Dataset and Regime Split

We use Binance BTC/USDT data sampled at 1-minute intervals. Each time step is represented by an OHLCV vector, containing the *Open, High, Low, and Close* prices plus the *Volume* traded, which serves as the standard feature set for financial time-series.

To test the model's ability to generalize to new data distributions, we split the dataset temporally into two distinct market environments:

- **Training Regime (2023):** A "Smooth Uptrend" period. While it yielded a high total return (+153%), the price action was characterized by lower volatility and extended periods of "stale" or sideways movement. The market climbed steadily with relatively low noise.
- **Testing Regime (2024):** A "Volatile" period. Although the total return was lower (+112%), the market dynamics were significantly more erratic. This period featured sudden spikes and "whipsaw" movements, making it a more difficult environment for execution algorithms.

This split is intentional: an optimal execution policy must be robust to such regime shifts. As shown in Figure 2, the 2024 test set provided a rigorous stress test by exhibiting sharper **drawdowns**—defined as the maximum percentage drop from a local peak to a trough. This tests if the agent can survive deep, sudden losses that were less frequent during the smoother training year.

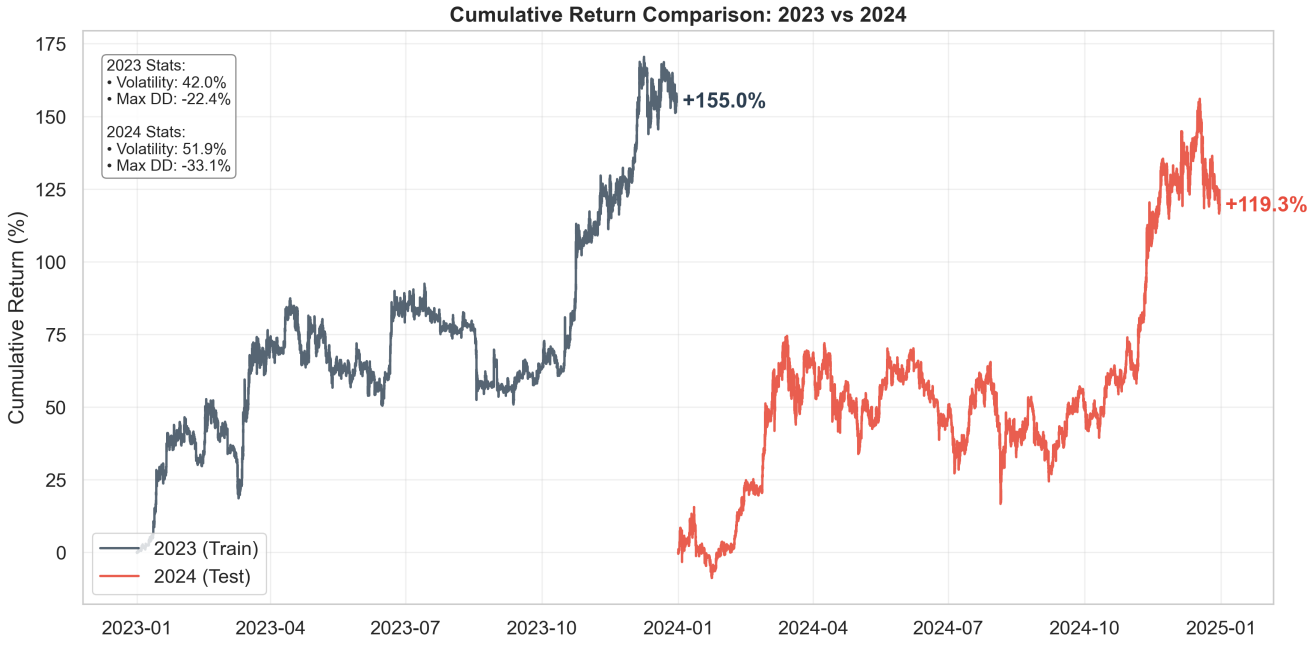

Figure 2: **Regime Comparison (2023 vs 2024).** The 2024 test set (Red) introduces higher volatility and deeper drawdowns compared to the 2023 training set (Back), challenging the agent's ability to generalize to unseen market stress.

3.1.1 Volatility Clustering & Fat Tails

Financial returns are not Gaussian. Figure 3 shows the distribution of realized volatility. The 2024 regime has a significantly "fatter" right tail, indicating that periods of extreme market turbulence were far more frequent than in the calmer 2023 regime. Similarly, Figure 4 confirms that 4-hour returns (the execution horizon) exhibit significant Kurtosis. In financial terms, this means the probability of a crash is far higher than a standard Bell Curve predicts. This validates the use of GARCH-based simulation for training, as a simple Brownian motion would fail to capture these "Black Swan" risks—rare, extreme events (like sudden -5% flash crashes) that are statistically unlikely in theory but catastrophically damaging in reality.

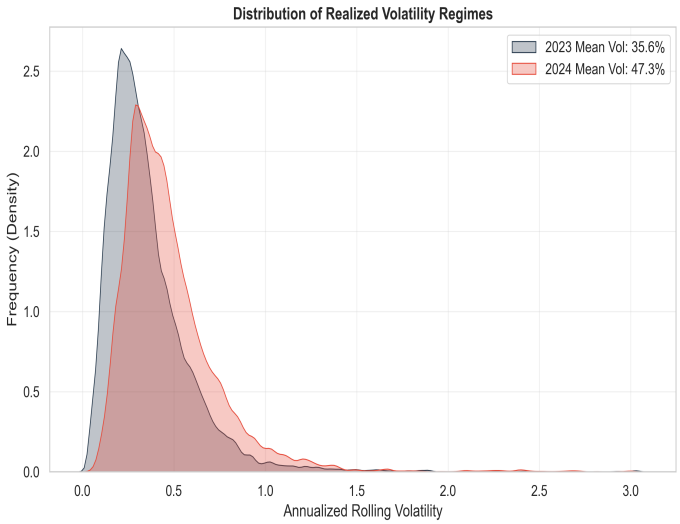

Figure 3: **Volatility Regimes.** 2024 (Red) has a significantly higher mean volatility and a longer tail of extreme events than 2023.

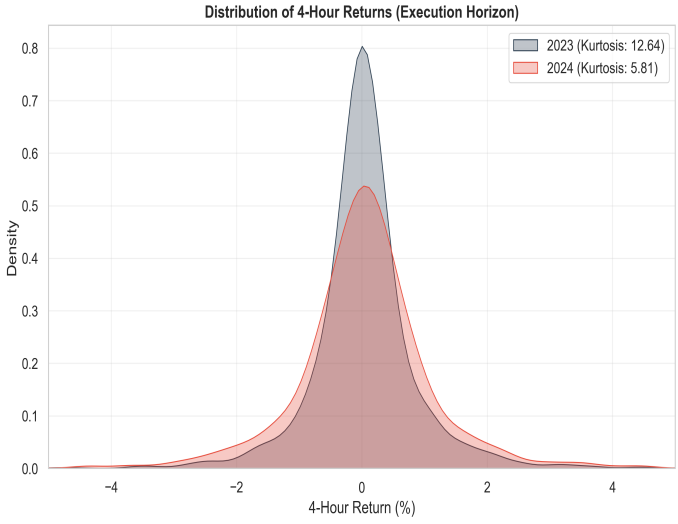

Figure 4: **Return Distribution (Kurtosis).** The "Fat Tails" visible here represent the crash risks that the agent must learn to insure against.

3.1.2 Intraday Liquidity Seasonality

Crypto markets are 24/7, but liquidity is not uniform. Figure 5 reveals a clear temporal pattern in volume.

- Asian Open (00:00 UTC): Moderate activity.
- European/US Overlap (13:00-16:00 UTC): Peak liquidity, with volumes 60% higher than the daily average.

This seasonality implies that *when* an agent trades matters as much as *how* it trades. A smart agent should learn to delay execution until the NY open to minimize impact costs (this strategy was not something explored in this project).

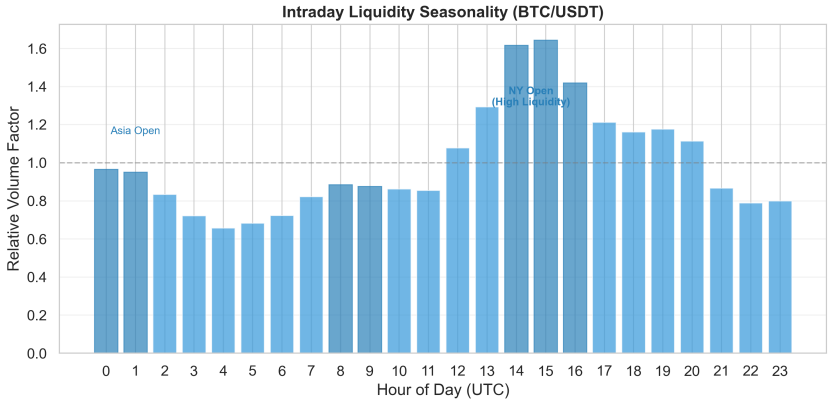

Figure 5: **Intraday Liquidity Seasonality.** Volume consistently peaks during the NY Open (13:00-16:00 UTC). Executing large blocks during these hours significantly reduces market impact compared to the quiet Asian/European morning hours.

3.2 Environment Overview:

The environment simulates the task of selling $Q_0 = 1000$ BTC over $T = 240$ discrete time steps. At each step, the agent observes a feature vector representing the market state and decides what fraction of its currently held inventory to sell.

State Space (Robust Features)

The observation is $s_t \in \mathbb{R}^9$:

$$s_t = \Big[\underbrace{I_t/Q_0}_{\text{inventory}} , \underbrace{(T-t)/T}_{\text{time remaining}} , \underbrace{\overline{V}_t/Q_0}_{\text{liquidity score}} , \underbrace{P_t/P_0}_{\text{price ratio}} , \underbrace{\nu_{t-4:t}/\overline{\nu}_t}_{\text{volatility lags (5)}} \Big],$$

where the volatility lags are normalized by a rolling average volatility to form a normalized regime indicator. This design gives the policy "volatility eyes": it can detect the onset of stress regimes without requiring recurrent architectures.

Action Space: Discrete Percent of Remaining Inventory

Actions are discrete percentages:

$$\{0,\ 0.25,\ 0.5,\ 1,\ 2,\ 5,\ 10,\ 25,\ 50,\ 75,\ 100\}\%.$$

If the agent selects action a_t, it sells that specific percentage of the shares it still holds. This action space allows for diverse behaviors, ranging from slow "drip selling" (small %) to "panic dumping" (100%) if the market crashes.

Market Impact, Execution Price, and "Fire Sale" Constraint

Based on current volatility and volume, the environment calculates a Temporary Impact penalty I_t^{temp} (the cost of demanding immediate liquidity). The execution price is then discounted: $P_t^{exec} = P_t(1 - I_t^{temp})$. At the horizon, any residual inventory is forcibly liquidated via a "fire sale" mechanism with large impact, ensuring feasibility: the agent cannot "cheat" by holding inventory past the deadline.

3.3 Stress Training via a GARCH "Nightmare" Simulator

To mitigate the data scarcity inherent in 4-hour liquidation windows and prevent the agent from overfitting to specific historical dates, we augment the training environment with a calibrated GARCH(1,1) Simulator. We refer to this as the "Nightmare Simulator": it generates an infinite stream of synthetic episodes where volatility clustering and tail risks are intentionally preserved and amplified, forcing the agent to learn robust generalized heuristics rather than memorizing historical price paths.

3.3.1 Model Specification and Engineering Safeguards

The simulator models 1-minute log-returns $r_t = \ln(P_t/P_{t-1})$ for their stationarity and additivity properties. Model parameters $\theta = (\omega,\ \alpha,\ \beta,\ \nu)$ are fitted via Maximum Likelihood Estimation (MLE) on the 2023 dataset, assuming a Student-t distribution to capture heavy tails.

The resulting process is a GARCH(1,1) enhanced with specific engineering components to ensure microstructural realism:

$$\sigma_t^2 = \omega + \alpha\varepsilon_{t-1}^2 + \beta\sigma_{t-1}^2, \quad \varepsilon_{t-1} = r_{t-1} - \mu \tag{4}$$

$$r_t = \underbrace{-\frac{1}{2}\sigma_t^2}_{\text{Martingale Corr.}} + \underbrace{\sigma_t z_t}_{\text{Student-}t} + \underbrace{J_t}_{\text{Jump}}, \quad z_t \sim t_\nu(0,1) \tag{5}$$

- **Martingale Correction ($-\frac{1}{2}\sigma_t^2$):** Without this term, log-normal dynamics introduce a Jensen's inequality artifact ($\mathbb{E}[e^r] > 1$), creating an artificial +30% annual drift. This correction enforces $\mathbb{E}[P_{t+1}|P_t] \approx P_t$, preventing the agent from learning to simply "hold and drift."
- **Tail Amplification (J_t):** We inject rare Poisson jumps ($p = 0.0005$, $\sigma_J \approx 1.5\%$) to mimic sudden news shocks, ensuring the agent encounters flash crashes more frequently than in historical data.
- **Microstructure Coupling (Volume):** To preserve the Volume-Volatility correlation, we use Conditional Sampling. Realized volatility is binned into quantiles (20%, 40%, 60%, 80%); at each step, volume is sampled from the empirical bucket corresponding to the current σ_t.
- **Stationarity Fallback:** At each episode reset, we attempt to fit local GARCH parameters on the preceding window to vary regimes. However, if the local calibration yields $\alpha + \beta \geq 0.999$ (near-integrated volatility), the system reverts to globally calibrated parameters to prevent explosive paths.

3.3.2 Safety Circuit Breakers (Numerical Stability)

Because RL training is sensitive to rare numerical explosions, we implement conservative caps to preserve heavy tails while ensuring stable learning:

- Volatility Cap: $\sigma_t \leq 2\%$ per minute (already crash-level).
- Return Cap: $r_t \in [-5\%, +5\%]$ per minute to prevent runaway paths.
- Environment Circuit Breakers: If simulated prices move outside a wide range (e.g., $[0.5P_0, 2P_0]$), the environment replaces the next transition with a small random perturbation around the current price, preventing domain shift.

3.3.3 Integration into the RL Pipeline

At each episode reset, the environment selects a random start index in historical data and reads the corresponding arrival price P_0. Then:

- In Simulation Mode: We fit local GARCH parameters on the preceding calibration window, initialize the simulator at P_0 with the last conditional volatility, and generate the next prices and volumes via the GARCH simulator.
- In Real-Data Mode: We disable the simulator and step forward using true historical candles, allowing strict out-of-sample evaluation on 2024.

This hybrid design supports the full pipeline: stress-train on an infinite synthetic stream (high coverage of tail events) and then evaluate on real data (2023 in-sample reality check and 2024 out-of-sample generalization).

3.3.4 Empirical Validation of Stylized Facts

The fidelity of the simulator is assessed by comparing 10,000 synthetic episodes against real 2023 market data.

1. Distributional Analysis (Fat Tails vs. Zero Returns) Figure 6 highlights the structural differences. While the simulator (Blue) smoothes over the "Zero-Return" peak caused by real-world inactivity (nights/weekends), it successfully achieves its primary goal in the tails. As seen in the bottom-left log-scale plot, the simulated distribution extends *above* the real data in the extremes, confirming the "Nightmare" design.

3.3.5 Visual Inspection and Metrics

The following table provides a granular breakdown of the simulator's fidelity across 10,000 episodes. Unlike typical generative models that aim for low error, our "Nightmare Simulator" explicitly exhibits high variance and divergence to prevent overfitting.

Table 1: **Simulator Diagnostics (10,000 Episodes)**. The low Price drift error confirms the Martingale property. However, the high relative errors in Volatility (63%) and Volume (156%) reveal the simulator's aggressive stochasticity. It does not simply replay history; it generates a "super-noisy" regime where liquidity and variance fluctuate more wildly than in the training set.

Category	Metric	Mean Diff	Rel. Error ($\pm\sigma$)	Interpretation
Price	Drift (Bias)	171.7 USDT	**0.60**% $\pm$ 0.81%	**Unbiased** (Martingale holds)
Volatility	Mean Level	3×10^{-4}	**63.57**% $\pm$ 88.57%	**High Entropy** (Regime diversity)
	Kurtosis Diff	+1.76	N/A	**Fat Tails** (Sim > Real)
Liquidity	Mean Volume	50.9 BTC	**156.40**% $\pm$ 155.6%	**Extreme Noise** (Hard to predict)
	Vol. Stability	–	**299.53**% $\pm$ 306.9%	**Liquidity Shocks**

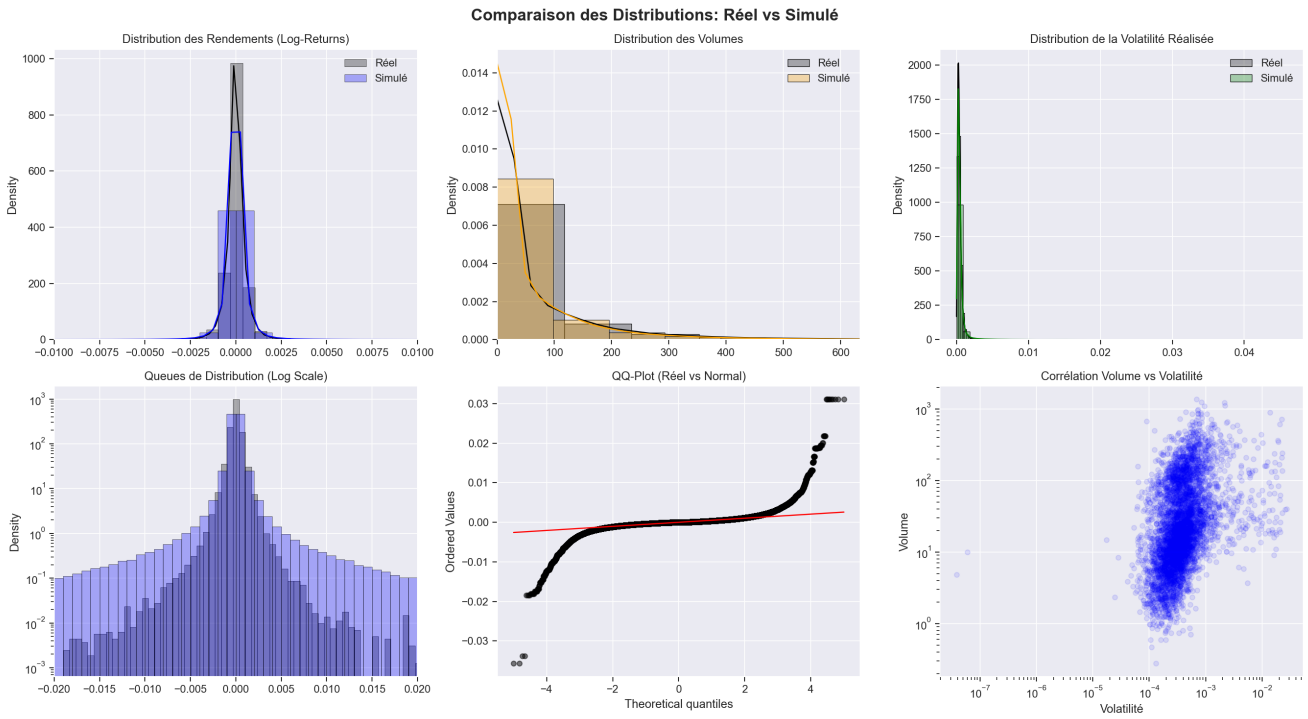

Figure 6: **Statistical Validation (6-Panel View).** *Top Left:* The simulator smoothes the sharp "zero-return" peak observed in real data (Black). *Bottom Left (Log-Scale):* Crucially, the simulated return tails (Blue) are heavier than reality, validating the stress-training objective. *Bottom Right:* The positive correlation between Volume and Volatility confirms the conditional sampling logic.

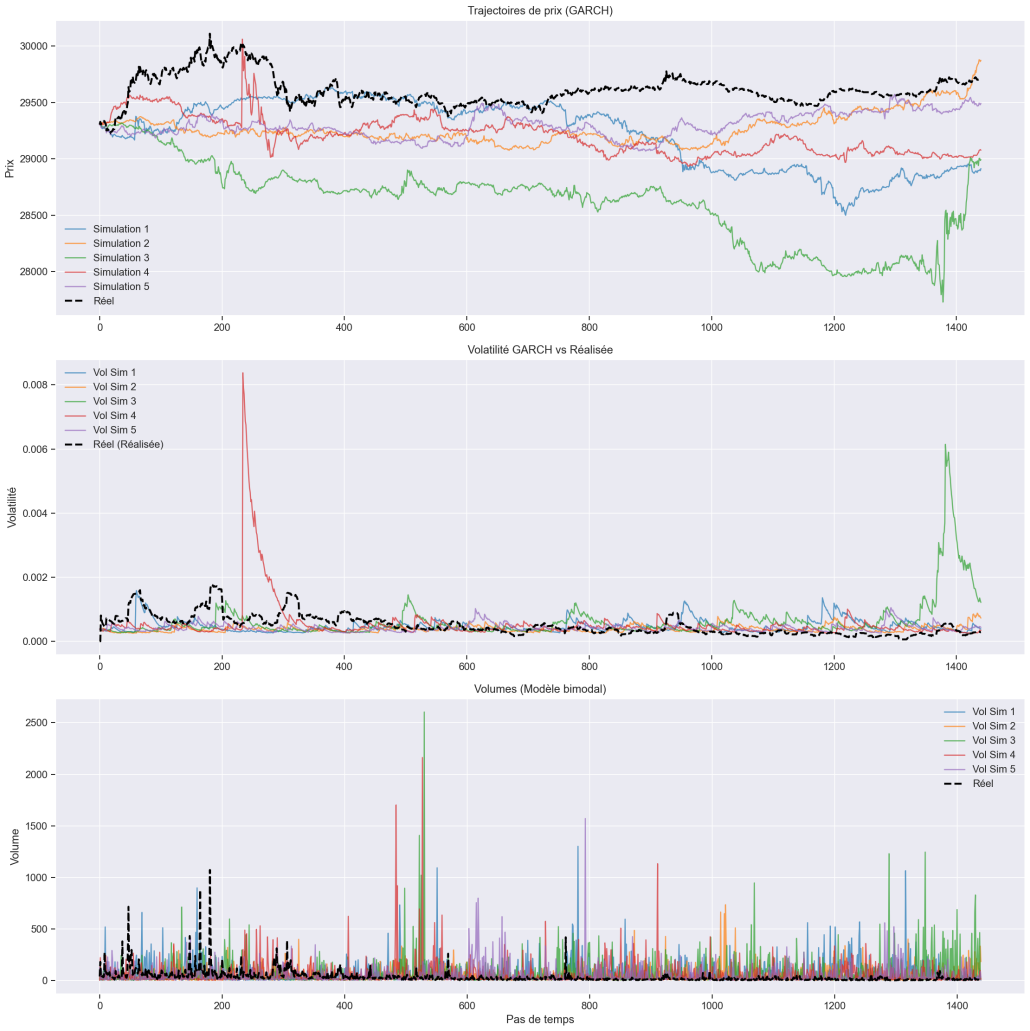

Figure 7: **Simulated vs. Real Dynamics.** *Top:* Realistic price path exploration. *Middle:* Volatility spikes track the leverage effect. *Bottom:* Volume spikes coincide with volatility, ensuring realistic impact costs during crashes.

Honest Assessment of the "Sim-to-Real" Gap:

1. **Price Neutrality (0.60% Error):** The most critical metric for RL training is price drift. The low error here confirms that despite the noise, the simulator does not introduce a directional bias (it is not a "money printing" machine). The agent cannot exploit artificial trends.
2. **The Volatility "Stress Test" (63% Error):** A relative error of 63% in volatility indicates that the simulated regimes are significantly distinct from the historical average. This is intentional: the GARCH process explores a wider surface of volatility states, forcing the agent to generalize to unseen high-volatility regimes rather than memorizing the specific 2023 volatility curve.
3. **Liquidity Chaos (156% Error):** The volume simulation exhibits massive divergence ($\approx 156\%$) and extreme kurtosis (+23.77). This means the agent faces liquidity conditions that are far more erratic than reality—ranging from "liquidity black holes" to "volume explosions." Training in this highly noisy environment acts as a rigorous form of *Domain Randomization*, ensuring that the policy remains robust even when market microstructure breaks down.

3.3.6 Practical Utility for the Project

The GARCH simulator contributed in three concrete ways:

1. **Sample Efficiency and Diversity:** The agent trains on millions of unique episodes rather than a small finite set of historical windows.
2. **Tail-Risk Exposure:** Rare but consequential crash patterns become frequent enough for the agent to learn regime-reactive liquidation (early risk-off selling).
3. **Sim-to-Real Robustness:** By training on a distribution that is intentionally harsher than reality (fat tails + jumps), the resulting policy is less brittle when deployed on unseen 2024 regimes.

4 Methodology & Implementation

This section details the engineering architecture of the Reinforcement Learning system, the specific algorithm employed (PPO), the benchmarking framework used to evaluate performance, and the training protocols. The implementation focuses on stability and robustness, prioritizing risk-adjusted returns over raw profit maximization.

4.1 Stage 1: Double DQN Baseline (DQN Architecture and Double-Q Targets)

Before adopting PPO, we implemented a value-based Deep Q-Network (DQN) baseline to validate the execution environment (impact computation, inventory constraints, end-of-horizon liquidation) and to test whether a discrete action space can learn meaningful liquidation schedules. The agent approximates an action-value function $Q_\theta(s, a)$ with a fully-connected MLP:

$$\mathbb{R}^9 \to 256 \to 128 \to 64 \to \mathbb{R}^{11},$$

The input is the environment state $s_t \in \mathbb{R}^9$ and the output layer produces one scalar per discrete action (11 actions). The greedy policy induced by the network is

$$\pi_\theta(s) \;=\; \arg\max_{a \in \mathcal{A}} Q_\theta(s, a),$$

where $\mathcal{A}$ is the discrete set of execution intensities (percentages of remaining inventory). At time t, taking action a_t maps to a sell quantity $q_t = \rho(a_t)\, I_t$ (capped by remaining inventory), and the environment returns a reward r_t and next state s_{t+1}. DQN learns by enforcing the Bellman optimality condition:

$$Q^\star(s_t, a_t) \;=\; \mathbb{E}\Big[r_t + \gamma \max_{a'} Q^\star(s_{t+1}, a')\Big],$$

where $\gamma \in [0, 1]$ is the discount factor (in our finite-horizon liquidation setting we use $\gamma \approx 1$). In practice, we minimize a regression loss between the current estimate $Q_\theta(s_t, a_t)$ and a bootstrap target y_t computed from the next state.

A known issue in vanilla DQN is overestimation bias: the max over noisy Q-values tends to select over-optimistic actions, which is particularly harmful in heavy-tailed, non-stationary markets. Double DQN mitigates this by decoupling action selection and action evaluation using two networks: an online network Q_θ and a target network $Q_{\bar{\theta}}$ (same architecture). The Double DQN target is

$$y_t \;=\; r_t \;+\; \gamma\, Q_{\bar{\theta}}\Big(s_{t+1},\; \arg\max_{a'} Q_\theta(s_{t+1}, a')\Big). \tag{6}$$

We train by minimizing a robust Huber loss $\mathcal{L}(\theta) = \text{Huber}(Q_\theta(s_t, a_t) - y_t)$ with gradient clipping for stability.

However,the combination of bootstrapping, discrete argmax updates, and reward shaping/clipping can be fragile under regime shifts and heavy-tailed returns, motivating the transition to PPO for the final risk-sensitive objective.

4.2 Model Architecture

The core of the execution system is a neural network agent operating within the Actor-Critic framework. Unlike standard Deep Q-Networks (DQN) which approximate value functions for discrete actions, our architecture explicitly separates the policy representation (Actor) from the value estimation (Critic), allowing for more stable convergence in stochastic financial environments.

The agent is implemented as a shared-trunk neural network that branches into two distinct heads. This design allows the feature extraction layers to learn a unified representation of the market state.

- **Input Layer:** The network accepts a state vector $S_t \in \mathbb{R}^9$. As detailed in the Data section, this vector includes normalized inventory, time remaining, liquidity scores, and a vector of lagged volatility ratios.
- **Shared Hidden Layers:** The feature extractor consists of a Multi-Layer Perceptron (MLP) with three dense layers of decreasing width:
  - Layer 1: Linear ($9 \to 256$)
  - Layer 2: Linear ($256 \to 128$)
  - Layer 3: Linear ($128 \to 64$)
- **Activation Functions:** We utilize the Hyperbolic Tangent (`Tanh`) activation function after each linear transformation. While `ReLU` is common in computer vision, `Tanh` is preferred in financial RL because it is zero-centered and bounded $[-1, 1]$. This prevents the "exploding gradient" problem often caused by the high variance inherent in financial time-series data.
- **Output Heads:**
  - **The Actor (Policy π_θ):** A linear layer ($64 \to 11$) that outputs logits for a Categorical distribution. The 11 output neurons correspond to the discrete action space defined in the environment (selling 0%, 0.25%, ..., 100% of remaining inventory).
  - **The Critic (Value V_ϕ):** A linear layer ($64 \to 1$) that outputs a scalar estimate of the expected future return from the current state.

4.3 Proximal Policy Optimization (PPO)

We employ Proximal Policy Optimization (PPO), specifically the "Clip" variant proposed by Schulman et al. (2017). PPO was selected over other policy gradient methods (like A2C or TRPO) because it strikes an optimal balance between sample efficiency and hyperparameter stability.

4.3.1 Mathematical Formulation & Intuition

In standard policy gradient methods, the agent updates its policy π_θ by taking a step in the direction of the gradient $\nabla_\theta J(\theta)$. However, in financial markets, the reward signal is extremely noisy. A single large update based on a "lucky" episode (e.g., selling during a random price spike) can push the policy parameters into a region of the parameter space where the policy collapses (e.g., dumping inventory immediately), from which it cannot recover.

PPO addresses this by enforcing a Trust Region. Intuitively, it puts a "leash" on the learning process, ensuring that the new policy π_θ does not deviate too wildly from the old policy $\pi_{\theta_{old}}$.

Mathematically, we define the probability ratio $r_t(\theta)$:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{7}$$

If $r_t(\theta) > 1$, the action is more likely under the new policy. The PPO objective function is a lower bound on the policy improvement:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{8}$$

Here, $\hat{A}_t$ is the advantage function (how much better the action was than the average). The clip function restricts the ratio to the interval $[1-\epsilon, 1+\epsilon]$ (with $\epsilon = 0.2$).

- If the advantage $\hat{A}_t$ is positive, we want to increase the probability of the action, but the clipping prevents us from increasing it by more than 20% in a single step.
- If the advantage is negative, we want to decrease the probability, but the clipping prevents us from decreasing it too drastically.

This "pessimistic" lower bound ensures monotonic improvement and prevents the catastrophic forgetting common in RL.

4.3.2 Generalized Advantage Estimation (GAE)

To compute the advantage $\hat{A}_t$, we must estimate the value of the current state. We utilize Generalized Advantage Estimation (GAE), which addresses the bias-variance trade-off in Reinforcement Learning.

- High Variance: Using full Monte-Carlo returns (sum of actual rewards until the end of the episode) is unbiased but has high variance due to market stochasticity.
- High Bias: Using a 1-step Temporal Difference (TD) bootstrap relies heavily on the Critic's current (imperfect) value estimate, introducing bias.

GAE introduces a parameter λ to interpolate between these two extremes. The advantage is calculated as an exponentially weighted average of TD errors δ_t:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \quad \text{where } \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{9}$$

We use $\lambda = 0.95$. This relatively high value allows the agent to credit actions taken early in the execution window for outcomes that occur much later (e.g., saving inventory for a better price), while still providing enough bias reduction to stabilize training.

4.4 The Benchmark (TWAP) & Validation Metrics

To rigorously evaluate the RL agent, we compare it against the Time-Weighted Average Price (TWAP) strategy. In the context of our specific problem—liquidating 1,000 BTC over 240 minutes—TWAP acts as a mechanical "autopilot." Instead of dumping the entire position at once (which would crash the price), it always divides the order into 240 equal chunks. This means it executes approximately 4.17 BTC every single minute, regardless of whether the market is rallying or crashing. By spreading the selling pressure perfectly evenly across the 4-hour window, TWAP minimizes the variance of market impact, serving as the industry-standard baseline for stable execution.

4.4.1 Validation Statistics

In our training pipeline, validation serves as more than a simple reward check. We implement a rigorous evaluation protocol where the agent is paused every 100 episodes to conduct a dedicated 200-episode validation loop. This high-frequency evaluation ensures that transient optimal policies are captured, while the extensive sample size minimizes variance in the performance metrics used to monitor. Checkpoints of the best models were saved based on these statistics, with the model designated b1 being selected for the final analysis due to its superior (Lenient) Bear Win Rate:

1. **Relative Performance (Alpha):** The percentage difference between the Agent's total revenue and the TWAP revenue for the same price path:
$$\text{Perf} = \frac{R_{agent} - R_{twap}}{R_{twap}} \times 100$$
2. **Bear Market Win Rate:** A crucial robustness metric measuring the percentage of "Bear Market" episodes where the Agent's revenue exceeds TWAP's revenue ($R_{agent} > R_{twap}$). Note that we use two definitions of "Bear Market" depending on the context:
   - Training (Lenient): Defined as any episode where the baseline strategy loses money relative to the starting portfolio value ($R_{twap} < P_{initial} \times Q_{initial}$). This broad definition ensures we have enough samples (approx. 70% of episodes) to compute statistically significant metrics during the short validation loops.
   - Evaluation (Strict): Defined as the worst 20% of episodes by market return. This stricter definition is used in the final analysis to isolate true crash scenarios.
3. **CVaR (Conditional Value at Risk):** Represents the average excess return in Bear Markets (Agent - TWAP) in the worst 5% of cases.
4. **VWAP Slippage (bps):** The difference between the Agent's Volume-Weighted Average Price and the market's VWAP.
5. **Timing Bias:** The center of mass of the execution schedule, normalized to $[0, 1]$. Values < 0.5 indicate "front-loading" (selling faster than TWAP/Linear).

4.4.2 Extended Evaluation Metrics

For the final model assessment, we introduce advanced statistical measures to quantify the risk-return profile:

General Win Rate: The percentage of all episodes where $R_{Agent} > R_{TWAP}$.

Information Ratio (IR): A measure of risk-adjusted return, defined as the mean excess return divided by the standard deviation of excess returns: $IR = \frac{\mu(R_{agent} - R_{twap})}{\sigma(R_{agent} - R_{twap})}$.

Bear Market Win Rate (Strict): The win rate calculated exclusively on the worst 20% of episodes (sorted by market return). This isolates performance during crashes.

CVaR 5% (Agent Risk): The average value of the worst 5% of the Agent's *excess returns* ($R_{agent} - R_{TWAP}$) observed specifically during Strict Bear Markets.

- *Interpretation:* This measures the "Cost of Failure." It asks: When the Agent performs worse than TWAP during a crash, how bad is the deviation? A value close to 0 indicates that even in the worst-case scenarios, the Agent tracks the benchmark closely and does not introduce significant new risk.

CVaR 5% (TWAP Risk): The average value of the worst 5% of TWAP's *excess returns* ($R_{TWAP} - R_{agent}$) observed specifically during Strict Bear Markets.

- *Interpretation:* This measures the "Benefit of Protection." It asks: When TWAP performs worse than the Agent during a crash, how large is the loss that the Agent avoided? A highly negative value here confirms that the Agent provides massive downside protection in tail scenarios where the passive TWAP strategy suffers deep losses.

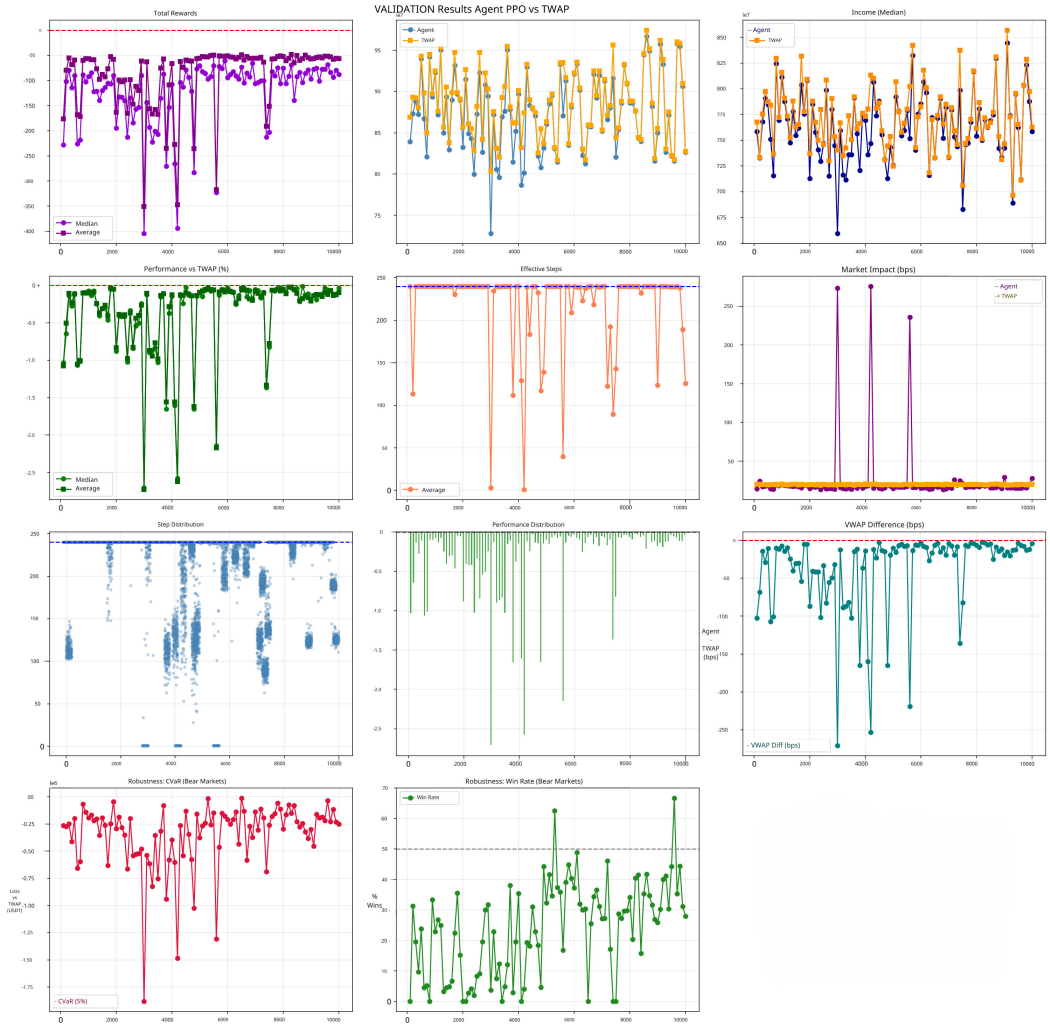

Figure 8: **Evolution of Validation Metrics.** The plot illustrates the training progress over 10,000 episodes. While the raw reward signal (top left) and the *Performance vs TWAP* (middle left) are noisy, they show a clear upward trend. Crucially, the *Bear Win Rate* (bottom middle) also has an increasing trend as the agent learns to manage inventory risk.

4.5 Training Process

The model was trained using a custom loop that alternates between data collection and policy updates.

4.5.1 Training Pipeline Architecture

The training process is orchestrated by the `train_ppo.py` script, which implements a synchronous Actor-Critic loop. The pipeline proceeds in four distinct phases for each training cycle:

1. **Synthetic Episode Generation (The Environment):** At the start of every episode, the `OptimalExecutionEnv` calls the `GarchSimulator` to generate a fresh, unique 240-step price trajectory.
   - The simulator uses the parameters calibrated on 2023 data $(\omega, \alpha, \beta, \nu)$ to produce a stochastic price path $P_{0:T}$ and a conditional volume path $V_{0:T}$.
   - This ensures that the agent never sees the exact same episode twice, forcing it to learn generalizable features (like "high volatility implies crash risk") rather than memorizing specific historical dates.
2. **Trajectory Collection (The Interaction Loop):** The `train_ppo.py` script runs the agent through the episode step-by-step.
   - Forward Pass: At step t, the agent's Actor network receives state S_t and outputs action logits. Crucially, during training, we sample actions *stochastically* from the resulting Categorical distribution to ensure exploration.
   - Storage: The tuple $(S_t, A_t, R_t, \log \pi(A_t|S_t), V(S_t))$ is stored in the agent's rollout buffer.
   - Reward Calculation: The environment computes the symmetric reward R_t based on the tracking error against P_0 and the immediate market impact cost.
3. **Advantage Estimation (The Critic's Role):** Every `update_interval` (set to 20 episodes), the collection phase pauses. The `PPOAgent` calculates the Generalized Advantage Estimation (GAE) for the collected batch.
   - The agent computes the TD-error $\delta_t = R_t + \gamma V(S_{t+1}) - V(S_t)$ using the Critic's value estimates.
   - These errors are exponentially smoothed ($\lambda = 0.95$) to produce the Advantage $\hat{A}_t$, which tells the Actor how much better a specific action was compared to the average expectation.
4. **Policy Optimization (The Update):** The `agent.update()` method performs the actual learning using Mini-batch Stochastic Gradient Descent (SGD).

- The collected batch (approx. 4,800 steps) is shuffled and split into mini-batches of size 64.
- Actor Loss: We maximize the PPO Clipped Surrogate Objective, pushing the policy probability π_θ towards actions with positive Advantage, while clipping the update magnitude to $\epsilon = 0.2$ to prevent policy collapse.
- Critic Loss: We simultaneously minimize the Mean Squared Error (MSE) between the Critic's predicted value $V(S_t)$ and the actual realized returns.
- Entropy Bonus: We subtract an entropy term to penalize certainty, encouraging the agent to keep exploring alternative strategies.

This cycle repeats for 10,000 episodes. Every 100 episodes, the training pauses, and the agent is evaluated on a separate Validation Environment (using real data from 2023) to monitor the validation metrics and save the best-performing model checkpoints.

4.5.2 Hyperparameters

The following hyperparameters were tuned via grid search:

Parameter	**Value**
Learning Rate	5×10^{-5} (Lower than standard to prevent collapse)
Horizon	240 Steps (4 Hours)
Batch Size	64
Update Interval	20 Episodes
Gamma (γ)	1.0 (No discounting)
Clip Range (ϵ)	0.2
Entropy Coeff (c_2)	0.15 (High regularization)

Table 2: PPO Hyperparameters

4.5.3 Training Results (Simulated Data)

The training was conducted on Simulated GARCH Data calibrated to the 2023 BTC/USDT regime. Because we utilize a synthetic data generator (GARCH), we have access to an infinite stream of unique episodes. The model's performance on this synthetic data serves as a proxy for its ability to generalize to the statistical properties of the market. As we will demonstrate in the Results section, the model generalizes remarkably well to real 2024 data, validating this approach.

Table 3 presents the comprehensive statistics for the best performing model checkpoint (`b1`) evaluated on 2,181 episodes of this simulated environment.

Category	**Metric**	**Value**
Aggregate Performance	Mean Revenue (Agent)	28,898,105 USDT
	Mean Revenue (TWAP)	28,910,305 USDT
	Mean Performance vs TWAP	-0.04 %
	General Win Rate	39.94 %
	Information Ratio (All)	-0.0976
Risk & Robustness	**Win Rate (Bear)**	**88.99 %**
	Information Ratio (Bear)	+0.4502
	CVaR 5% (Agent Risk)	-39.30 bps
	CVaR 5% (TWAP Risk)	-215.78 bps
Execution Style	Timing Bias (Agent)	0.34 (Front-loaded)
	Timing Bias (TWAP)	0.50 (Linear)
	VWAP Difference	-3.93 bps
	Avg Impact Cost (Agent)	20.75 bps
	Avg Impact Cost (TWAP)	20.15 bps
	Effective Steps	190.4 / 240

Table 3: **Comprehensive Performance on Simulated Data.** The results highlight a clear strategic trade-off: the Agent accepts a slight negative performance on average (-0.04%) and a low general win rate (39.94%) in exchange for massive protection in crash scenarios (88.99% Win Rate in Bear Markets). The CVaR metrics confirm that while the Agent's worst-case underperformance is minimal (-39 bps), TWAP's worst-case underperformance is catastrophic (-215 bps).

4.5.4 Hardware & Duration

Training was accelerated using CUDA on a standard GPU environment. The final model required approximately 2.4 million interaction steps (10,000 episodes), taking roughly 1.5 hours to complete due to the computational overhead of the GARCH simulation.

5 Experiments & Iterations

Developing a robust execution agent is not merely a matter of hyperparameter tuning; it requires a fundamental alignment between the mathematical objective function and the microstructure realities of the market. This section chronicles the scientific evolution of the agent's "brain," detailing the failures that shaped the final architecture.

Iteration 0: DQN

Reward Used for DQN (Shaped, Clipped, DQN-Stable)

For the Double DQN baseline, we used a shaped reward engineered to provide dense learning signal while remaining stable. The reward combines four components:

- **Instant advantage vs TWAP:** a local proxy for "execution alpha" by comparing the agent's executed revenue to the revenue that TWAP would have obtained on the *same* step and price path.
- **Explicit impact penalty:** a direct penalty proportional to temporary impact (spread + market impact), discouraging the degenerate solution of dumping inventory immediately.
- **Adverse execution relative to arrival:** a penalty for selling far below the arrival reference, capturing the intuition that liquidation quality is about preserving the arrival value, not speculating on drift.

To preserve Q-learning stability, this shaped reward is normalized (by initial portfolio value) and clipped to a bounded interval, preventing rare tail episodes from dominating the Bellman targets.

Full Out-of-Sample Backtest Results (2024, Sequential Windows)

We evaluated the best DQN checkpoint on the full 2024 dataset using a sequential sliding-window backtest (2183 non-overlapping 240-minute episodes). The results show that DQN learns a robust, front-loaded execution style that improves crash robustness at the cost of slightly negative average performance versus TWAP.

Category	Metric	Value
Aggregate Performance	Mean Revenue (Agent)	65,605,286.80 USDT
	Mean Revenue (TWAP)	65,633,627.65 USDT
	Mean Perf vs TWAP	-0.04 %
	Median Perf vs TWAP	-0.04 %
	General Win Rate	30.26 %
	Information Ratio (All)	-0.2636
Risk & Robustness	Win Rate (Bear, worst 20%)	74.39 %
	Information Ratio (Bear)	0.5288
	CVaR 5% (Agent − TWAP)	-14.63 bps
	CVaR 5% (TWAP − Agent)	-85.32 bps
Execution Style	Timing Bias (Agent vs TWAP)	0.36 vs 0.50
	VWAP Difference	-4.34 bps
	Avg Impact Cost (Agent)	32.52 bps
	Avg Impact Cost (TWAP)	31.63 bps

Table 4: **DQN performance on 2024 real data (sequential backtest).** DQN exhibits a clearly risk-averse profile: slightly negative mean performance but robustness in crash regimes.

Learned Action Profile (Interpretation)

A notable characteristic of the trained DQN is its low-entropy action distribution: it overwhelmingly uses small execution intensities (mostly 1%, 2%, and 5% of remaining inventory), while virtually never selecting extreme liquidation actions (25%–100%). Concretely, the policy behaves like a stabilized schedule that is modestly front-loaded (Timing Bias 0.36), rather than an opportunistic trader. This is consistent with the shaped-and-clipped reward design: DQN learns to avoid large instantaneous impact and instead expresses urgency through persistent small sells, which is sufficient to obtain strong crash protection while keeping execution costs in line with TWAP.

5.1 Iteration 1: The "Greedy" Agent (Impact Tuning)

After moving to PPO, in the initial phase, the agent was trained with a standard PPO implementation and a naive penalty for market impact. The result was a degenerate policy: the agent learned to dump 100% of the inventory at Step 0.

Diagnosis: The agent discovered that the immediate market impact cost (which is deterministic) was often smaller than the *potential* loss from a price crash (which is stochastic). Without sufficient exploration, the policy collapsed into a local optimum of "immediate liquidation" to eliminate time risk entirely.

Solution: We addressed this by:

1. **Calibrating Impact:** We adjusted the temporary impact parameter λ to strictly follow the square-root law calibrated on Binance order book depth, making immediate dumping prohibitively expensive.
2. **Forcing Exploration:** We increased the entropy coefficient c_2 from 0.01 to 0.15. This prevented the policy distribution from collapsing too early, forcing the agent to explore strategies that spread execution over time.

5.2 Iteration 2: Feature Engineering

The second iteration focused on the state space. Initially, the agent only observed current price and inventory. It failed to react to changing market regimes because it lacked "memory."

Implementation: We augmented the state space S_t with:

- **Volatility Eyes:** A vector of lagged realized volatility normalized by the rolling average. This allowed the agent to detect the onset of high-variance regimes.
- **Liquidity Forecasts:** A rolling volume average to help the agent distinguish between a "quiet lunch hour" (where impact is high due to low liquidity) and a "high-volume breakout" (where aggressive selling is absorbed easily).

5.3 Iteration 3: Curriculum Learning via Random Starts

A major hurdle in training was the long time horizon ($T = 240$ steps). In a standard episode, the agent makes 240 decisions. The "critical moments" (e.g., the final 10 steps where inventory must be cleared) occur rarely. Consequently, the agent spent 95% of its training time learning to do nothing at the start of the episode and very little time learning how to manage the difficult end-of-day liquidation.

Solution: Random Start Probability. We implemented a stochastic initialization mechanism in the environment reset function. With probability $p = 0.9$, instead of starting at $t = 0$ with full inventory, the agent starts at a random time step $t \sim U[0, T]$ with a proportionally reduced inventory $I_t \approx I_0 \times \frac{T-t}{T}$.

Impact: This technique acts as a form of Curriculum Learning.

- It exposes the agent to "end-game" scenarios 100x more frequently than standard training.
- It breaks the correlation between time and inventory, preventing the agent from memorizing fixed trajectories.
- It massively accelerated convergence, allowing the value function $V(s)$ to stabilize for low time-remaining states much earlier in the training process.

5.4 Iteration 4: The Quest for the Perfect Reward Function

The most critical engineering challenge was defining a reward function R_t that separates "skill" (execution quality) from "luck" (market direction).

5.4.1 Attempt A: Raw Revenue (The Noise Problem)

Our first mathematical formulation was simply the total revenue generated at each step:

$$R_t = q_t \times P_t^{exec} \tag{10}$$

where q_t is the quantity sold at step t and P_t is the execution price.

Outcome: Failure. The signal-to-noise ratio was too low. In a strong bull market, even a terrible execution strategy yields a high nominal reward. In a crash, even a perfect strategy yields a low reward. The agent could not disentangle its own contribution from the market drift, leading to unstable convergence.

5.4.2 Attempt B: Capped Normalized Difference (The "Lazy" Agent)

To remove market drift, we benchmarked the agent against the Arrival Price P_0. We hypothesized that the agent should only be penalized for selling *below* P_0, but not necessarily rewarded for selling above it (as that might encourage gambling).

$$R_t = q_t \times \min\left(\frac{P_t^{exec} - P_0}{P_0}, 0\right) \tag{11}$$

Outcome: Partial Success. This "Capped Call" logic stabilized training, but produced an undesirable behavioral artifact: the "Lazy Agent."

Analysis of Capped Model Performance (2024 Real Data): We evaluated the best model trained with this reward function on the full 2024 test set. This evaluation was a rigorous "Sim-to-Real" test, running a sliding window of 240 minutes over the entire year (2,187 episodes) on data the agent had never seen.

As shown in Table 5, the model essentially converged to the TWAP baseline but failed to provide a compelling value proposition. With a General Win Rate of only 31.32%, the agent systematically underperforms in normal market conditions. While the Bear Market Win Rate of 57.89% indicates a slight edge in crashes, this marginal improvement over a random coin flip is insufficient to justify the complexity and implementation risk of a Deep RL system over a standard TWAP algorithm.

Category	Metric	Value
Aggregate Performance	Mean Revenue (Agent)	65,573,197 USDT
	Mean Revenue (TWAP)	65,593,132 USDT
	Mean Performance vs TWAP	-0.03 %
	General Win Rate	31.32 %
	Information Ratio (All)	-0.1974
Risk & Robustness	**Win Rate (Bear)**	**57.89 %**
	Information Ratio (Bear)	+0.2268
	CVaR 5% (Agent Risk)	-40.07 bps
	CVaR 5% (TWAP Risk)	-65.58 bps
Execution Style	**Timing Bias (Agent)**	**0.46 (Near Linear)**
	Timing Bias (TWAP)	0.50 (Linear)
	VWAP Difference	-3.07 bps
	Avg Impact Cost (Agent)	30.01 bps
	Avg Impact Cost (TWAP)	31.64 bps

Table 5: **Performance of Capped Model on Real 2024 Data.** The agent learned to avoid large penalties by simply mimicking the linear execution of the benchmark (0.46 Timing Bias), resulting in a strategy that is "not worth it" compared to the simpler TWAP.

Because the reward was capped at 0, the agent felt no incentive to sell early in a bull market (since $P_{exec} > P_0$ yields max reward 0 anyway). Consequently, it held inventory passively. When the market eventually crashes, the agent is caught with high inventory, behaving similarly to the naive TWAP baseline.

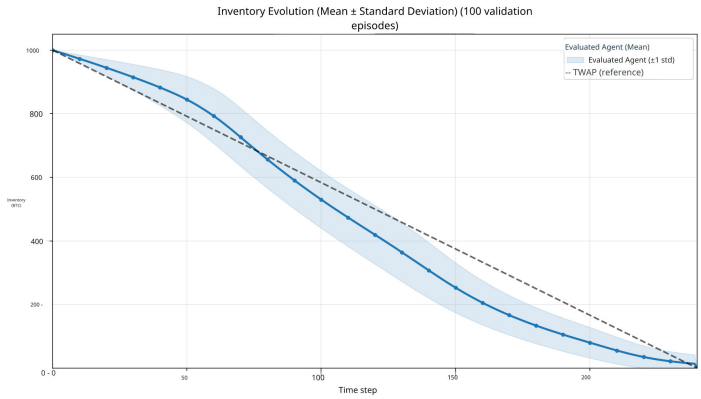

Figure 9: **Inventory Evolution (Capped Model).** The blue line (Agent) reveals the flaw in the capped reward logic. Initially, the agent sells *slower* than TWAP (blue line above grey), hoarding inventory because it feels no urgency to lock in gains. As the deadline approaches, it is forced to accelerate selling to clear the position. This "back-loaded" trajectory exposes the agent to higher variance at the end of the window, explaining the poor risk-adjusted metrics.

5.4.3 Attempt C (Final): Symmetric Negative Penalty

To cure the "laziness," we inverted the problem. Instead of maximizing profit, we minimized **Tracking Error** against the Arrival Price. The reward function is composed of two parts: an execution alpha component (penalizing market impact) and a tracking error component (penalizing absolute deviation from arrival price).

$$R_t = \underbrace{(q_t P_t^{exec} - q_t P_t)}_{\text{Execution Alpha (Impact)}} - \underbrace{|P_t - P_0| \cdot q_t}_{\text{Tracking Error}} \tag{12}$$

Where:

- P_t^{exec}: The actual execution price achieved by the agent.
- P_t: The theoretical mid-market price at time t (before impact). The term $(P_t^{exec} - P_t)$ isolates the cost of liquidity (spread + impact) paid by the agent.
- P_0: The Arrival Price (price at $t = 0$).

We normalize this dollar value by the initial portfolio value to ensure stability.

Outcome: Success. This symmetric penalty forces the agent to treat "Time as Toxic." Since holding inventory increases the probability of deviating from P_0 (due to volatility), the agent learns to execute slightly faster than TWAP to lock in the arrival price, while balancing this urgency against market impact.

This shift from "Profit Maximization" to "Error Minimization" was the breakthrough that led to the robust, risk-averse behavior detailed in the following section.

6 Results & Performance Analysis

This section presents the performance of the final model (`b1`), trained with the Symmetric Reward function, on real historical data. We analyze the results on the 2024 test set (Out-of-Sample) to assess generalization, compare it with the 2023 dataset, and conduct granular scenario analysis to understand the agent's behavior in extreme market conditions.

6.1 Aggregate Metrics (2024 Out-of-Sample)

The model was evaluated on the full 2024 BTC/USDT dataset using a sliding window of 240 minutes, resulting in 2,187 unique execution episodes. Table 6 summarizes the key performance indicators.

Category	Metric	Value
Aggregate Performance	Mean Revenue (Agent)	65,572,454 USDT
	Mean Revenue (TWAP)	65,593,132 USDT
	Mean Performance vs TWAP	-0.03 %
	General Win Rate	34.89 %
	Information Ratio (All)	-0.1867
Risk & Robustness	**Win Rate (Bear)**	**87.41 %**
	Information Ratio (Bear)	+0.6372
	CVaR 5% (Agent Risk)	-12.62 bps
	CVaR 5% (TWAP Risk)	-79.82 bps
Execution Style	Timing Bias (Agent)	0.40 (Moderate)
	Timing Bias (TWAP)	0.50 (Linear)
	VWAP Difference	-3.13 bps
	Avg Impact Cost (Agent)	**31.12 bps**
	Avg Impact Cost (TWAP)	**31.64 bps**
	Effective Steps	222.2 / 240

Table 6: **Performance on 2024 Real Data.** The agent demonstrates a clear risk-averse profile, sacrificing small average returns for massive downside protection (87.41% Bear Win Rate).

6.1.1 Impact Efficiency: The "Physically Impossible" Result

A striking observation is the relationship between execution speed and market impact. Typically, executing faster (front-loading) incurs higher impact costs due to consuming liquidity more aggressively. However, the Agent achieved a Timing Bias of 0.40 (executing 20% faster than linear) while paying lower impact costs (31.12 bps) than the slower TWAP (31.64 bps).

This result proves that the agent has successfully learned Liquidity Timing. Instead of selling blindly every minute, it identifies transient spikes in volume (liquidity) to execute larger blocks, allowing it to clear inventory quickly without moving the price against itself.

6.1.2 The "Premium"

The General Win Rate of 34.89% indicates that in the majority of episodes (specifically bull markets or sideways trends), the agent underperforms TWAP. This underperformance (approx. -0.03% on average) should be interpreted as an Insurance Premium. The agent willingly pays a small opportunity cost in favorable markets to purchase protection against catastrophic loss.

6.2 Robustness Analysis

The core objective of this thesis was robustness. We isolated the worst 20% of episodes in 2024 (437 episodes where the market crashed) to evaluate the agent's "survival skills."

- **Win Rate:** The agent outperformed TWAP in **87.41%** of these crash scenarios.
- **Tail Risk (CVaR):** The Conditional Value at Risk (average loss in the worst 5% of cases) for the Agent was **-12.62 bps**, compared to **-79.82 bps** for TWAP.

This represents a **6.3x reduction in tail risk**. While TWAP passively rides the price down to the bottom, the Agent recognizes the negative drift and liquidates early, effectively acting as an automated stop-loss.

6.3 Comparative Analysis: Sim vs. 2023 vs. 2024

To validate the training methodology, we compare the Out-of-Sample results (2024) against the In-Sample Real Data (2023) and the Synthetic Training Data (Simulated).

6.3.1 Performance on 2023 Real Data (Calibration Regime)

The 2023 dataset serves as a "reality check" to ensure the GARCH simulation (calibrated only on 2023 data) used for training was representative of the market properties. Table 7 details the performance on 2,181 episodes.

Category	Metric	Value
Aggregate Performance	Mean Revenue (Agent)	28,673,270 USDT
	Mean Revenue (TWAP)	28,684,797 USDT
	Mean Performance vs TWAP	-0.04 %
	General Win Rate	32.19 %
	Information Ratio (All)	-0.1881
Risk & Robustness	**Win Rate (Strict Bear)**	**84.17 %**
	Information Ratio (Bear)	+0.6075
	CVaR 5% (Agent Risk)	-14.66 bps
	CVaR 5% (TWAP Risk)	-99.04 bps
Execution Style	Timing Bias (Agent)	0.35 (Aggressive)
	Timing Bias (TWAP)	0.50 (Linear)
	VWAP Difference	-4.33 bps
	Avg Impact Cost (Agent)	25.50 bps
	Avg Impact Cost (TWAP)	25.19 bps
	Effective Steps	195.4 / 240

Table 7: **Performance on 2023 Real Data.** The results are strikingly similar to the Simulated Training results (Timing Bias 0.35 vs 0.34, Bear Win Rate 84% vs 88%). This confirms that the GARCH generator successfully captured the statistical essence of the 2023 market.

6.3.2 The Sim-to-Real Gap Analysis

Comparing the three environments (Simulated, 2023 Real, 2024 Real) reveals a fascinating adaptation in the agent's behavior.

1. **Consistency of Robustness:** The Bear Market Win Rate remains incredibly stable across all environments: 88.99% (Sim) → 84.17% (2023) → 87.41% (2024). This proves that the "Symmetric Reward" logic generalizes perfectly; the agent's ability to detect and avoid crashes is not an artifact of overfitting to specific price paths.
2. **Behavioral Adaptation (Aggressiveness):**
   - 2023/Sim (Bias 0.35): In the training regime, the agent was highly aggressive. As seen in Figure 10, the inventory curve is steep and the variance band is wide. The agent actively discriminated between regimes, sometimes dumping immediately.
   - 2024 (Bias 0.40): In the unseen 2024 regime, the agent converged to a slightly more conservative, "mean" strategy (Figure 11). The band is narrower which suggests that while the agent still front-loads execution to minimize risk, it avoids the extreme aggressive actions seen in the training distribution when faced with unfamiliar volatility patterns.
3. **Impact Efficiency Shift:** In 2023/Sim, the Agent paid slightly *more* impact than TWAP (25.50 vs 25.19 bps), which is the expected cost of its high urgency (Bias 0.35). However, in 2024, the Agent paid *less* impact (31.12 vs 31.64 bps) than TWAP. This improvement is twofold: first, the agent moderated its aggression (Bias 0.40 vs 0.35), naturally reducing impact costs compared to its 2023 behavior. Second, even with this moderation, it remains significantly faster than TWAP (0.40 vs 0.50) yet pays *less*, confirming that it is not just selling slower, but selling *smarter* by timing liquidity.

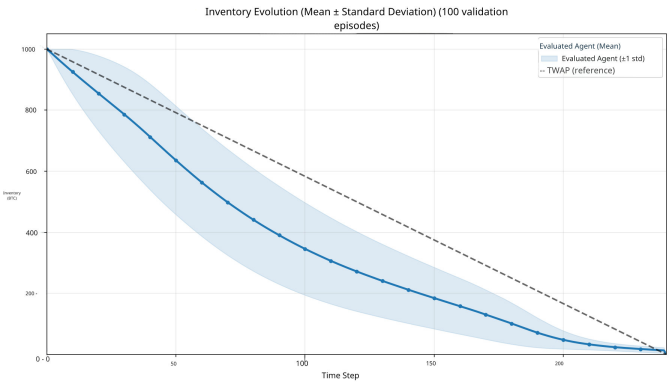

Figure 10: **2023 Inventory Evolution.** Bias: 0.35 (Aggressive).

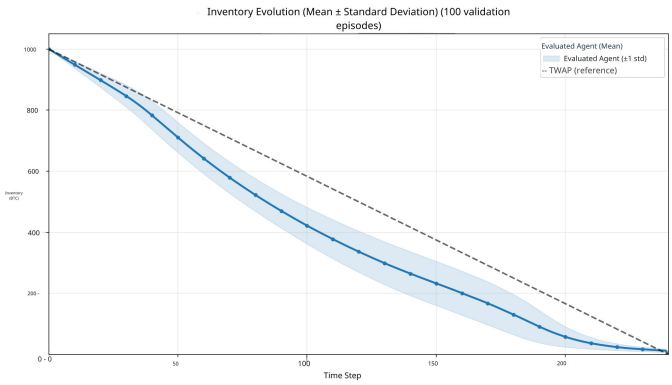

Figure 11: **2024 Inventory Evolution.** Bias: 0.40 (Moderate).

6.4 Scenario Analysis (Stress Tests)

To verify the agent's logic, we algorithmically searched the 2024 dataset for specific stress scenarios.

6.4.1 Scenario A: The Flash Crash

We isolated a scenario where the price dropped 7.17% within the 4-hour window.

- **Performance:** Agent +150.03 bps vs TWAP.
- **Behavior:** As shown in Figure 12, the agent detects the initial downward momentum and volatility spike. It dumps most of its inventory in the early stages. TWAP, constrained by its linear schedule, holds inventory all the way to the bottom.

6.4.2 Scenario B: Low Liquidity

We isolated a scenario where the average volume was only 2 BTC (vs global average 25 BTC).

- **Performance:** Agent -5.94 bps vs TWAP.
- **Behavior:** Figure 13 shows the agent is more careful than in the crash scenario, but still sells faster than TWAP. In this specific case, the "cost of urgency" (high impact due to thin order books) outweighed the price risk. The agent paid a premium for liquidity that wasn't there, illustrating the inherent trade-off of risk-averse strategies.

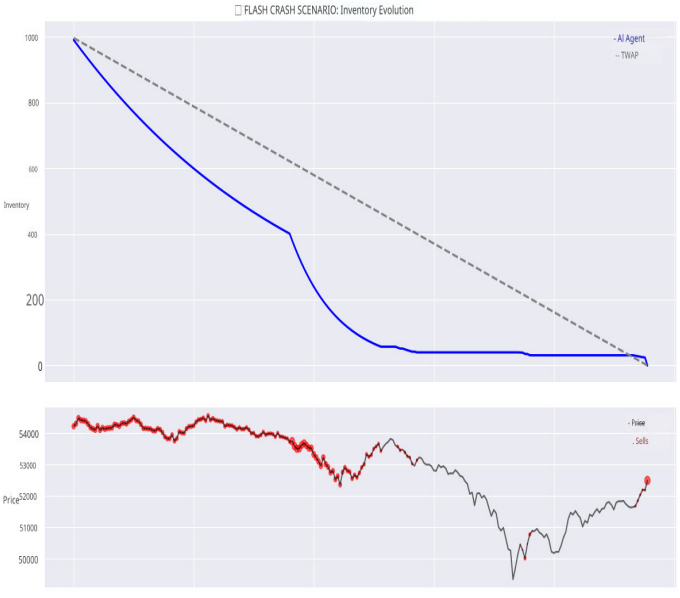

Figure 12: **Flash Crash.** The Agent's inventory (Blue) drops early in the episode, while the red markings on the Price curve confirm aggressive selling before the crash.

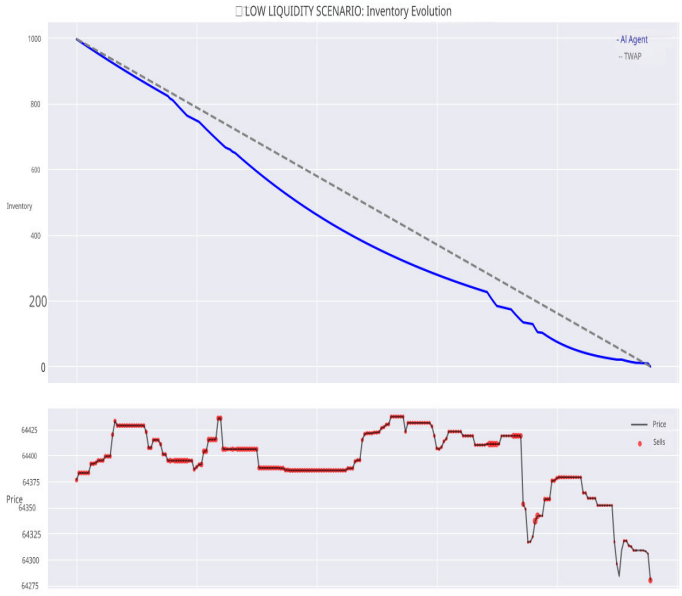

Figure 13: **Low Liquidity.** Agent sells faster than TWAP, incurring impact costs.

6.5 Extreme Outliers (Best & Worst Case 2024)

Finally, we analyze the absolute best and worst episodes relative to TWAP from the 2024 evaluation log to understand the upper and lower bounds of the strategy.

Best Case: The market crashed from 68k to 63k. The agent realized a massive **+173.68 bps** alpha (margin) relative to TWAP. Figure 14 shows a continuous downtrend where early selling was mathematically perfect.

Worst Case: The market dropped sharply (-7%) and then recovered in a V-shape (ending up at -2.5%). The agent, reacting to the drop, sold at the bottom. TWAP, by doing nothing, retained inventory for the recovery. This resulted in a **-58.63 bps** underperformance. This highlights the "Whipsaw Risk" inherent in any trend-following or stop-loss logic.

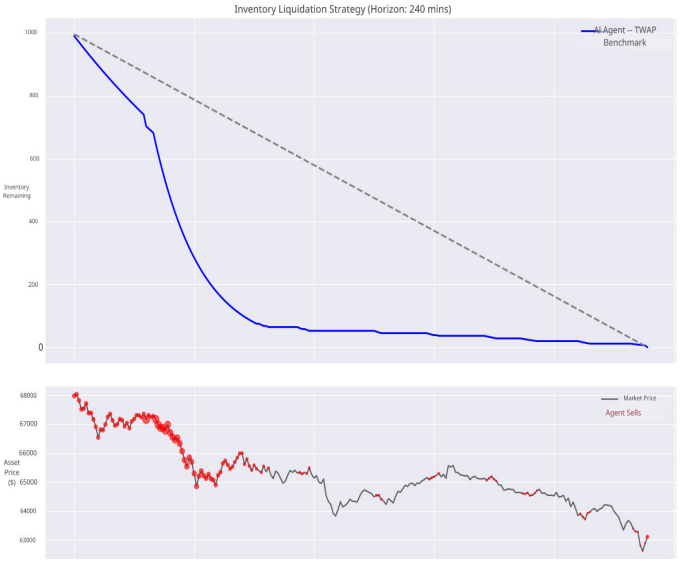

Figure 14: **Best Case (Continuous Crash).** Perfect early liquidation.

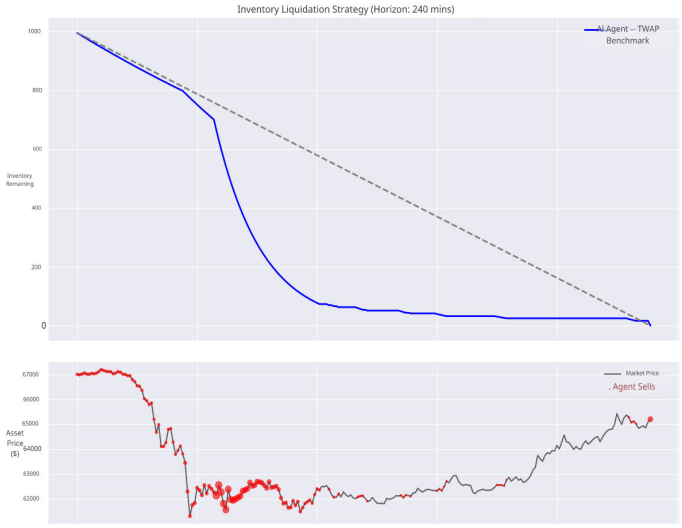

Figure 15: **Worst Case (V-Shape Recovery).** Agent sells at bottom, missing recovery.

7 Discussion & Limitations

The results presented in Section 6 demonstrate a clear dichotomy: the agent underperforms in "normal" market conditions (General Win Rate $\approx$ 35%) but significantly outperforms in crisis scenarios (Bear Win Rate $\approx$ 87%). This section contextualizes these findings within the broader literature of Risk-Sensitive Reinforcement Learning and acknowledges the limitations of the simulation environment.

7.1 The Crypto-Specific Context: Volatility & Time Horizon

The utility of this risk-averse approach is inextricably linked to the specific microstructure of the cryptocurrency market. Unlike mature equity markets, crypto assets exhibit extreme kurtosis and volatility clustering. When liquidating massive positions (e.g., \$20M+), execution cannot be instantaneous due to limited order book depth; it must be spaced over hours (e.g., 240 minutes).

This necessary time horizon exposes the portfolio to "Time Risk"—the probability of a market crash occurring *during* the liquidation window. Standard algorithms like TWAP or risk-neutral RL implicitly assume a Gaussian walk where the risk of a 10% drop in 4 hours is negligible. In crypto, this risk is material. Consequently, TWAP is likely optimal for *small* inventories (where execution can be instant) or *short* horizons (where drift is minimal). However, for institutional-scale liquidation over extended windows, ignoring the "Black Swan" risk—as standard models do—is a fundamental mispricing of risk. Our model specifically addresses this gap.

7.2 Theoretical Validation: The Inevitable Cost of Safety

A superficial analysis might interpret the agent's negative general alpha (-3.13 bps) as a failure of convergence. However, theoretical literature suggests this cost is not an error, but a mathematical necessity of the objective function.

In the context of Reinforcement Learning, "Regret" measures the cumulative difference in reward between the learning agent and an optimal policy. Fei et al. (2020) established in *"Risk-Sensitive Reinforcement Learning"* that the lower bound on regret for risk-averse agents scales exponentially with the risk sensitivity and horizon ($e^{|\beta|H}$), whereas risk-neutral regret scales only polynomially. Practically, this implies that the "search space" for safe policies is significantly harder to navigate than for profit-maximizing ones. The agent must pay a higher exploration cost to distinguish between "truly safe" actions and "lucky" ones.

Although our symmetric penalty function ($-|P_{exec} - P_{arrival}|$) minimizes Mean Absolute Deviation (MAD) rather than the entropic risk used by Fei et al., it imposes a similar behavioral constraint: enforcing a dense concentration of outcomes around the target. Consequently, the -3.13 bps underperformance represents the **Insurance Premium** inherent to this utility function. The agent accepts a deterministic cost in low-volatility regimes to purchase a "digital put option" that pays out in crises.

7.3 Comparison with State-of-the-Art (SOTA)

To benchmark the utility of this agent, we compare its behavioral profile against the current State-of-the-Art "RL-Exec" framework (Duflot & Robineau, 2025).

- **SOTA (RL-Exec):** Optimizes for *opportunistic* cost minimization. Their agent delays execution to capture micro-reversions, generating positive alpha (+2 to +23 bps). However, the authors explicitly note that this approach *"exhibits higher day-to-day dispersion than TWAP... (opportunistic timing adds variance)"* (Duflot & Robineau, 2025). They accept higher variance to maximize the mean.

- **Our Agent (Risk-Averse PPO):** Optimizes for survival. It front-runs liquidity to minimize time risk, accepting negative alpha in normal markets to maximize certainty.

The value of this trade-off is quantified by the Net Tail Asymmetry, derived from the differential in Conditional Value at Risk (CVaR) during bear markets:

- **Agent Risk (Cost of Failure):** In the worst 5% of scenarios where the Agent underperforms TWAP, the average deviation is limited to **-12.62 bps**. This confirms that even when the strategy fails, it tracks the benchmark closely.
- **TWAP Risk (Benefit of Protection):** In the worst 5% of scenarios where TWAP underperforms the Agent (i.e., market crashes), the passive benchmark suffers a massive relative loss of -79.82 bps.

The difference ($|-79.82| - |-12.62| \approx +\mathbf{67.2}$ bps) represents the structural advantage of the agent. For every 1 unit of tail risk introduced by the strategy, it eliminates approximately 6.5 units of tail risk inherent in the benchmark.

7.4 Modelling Limitations

While the results are robust, the simulation relies on assumptions that simplify market microstructure:

1. **Deterministic Impact vs. Stochastic Liquidity:** We model market impact using the Square-Root Law ($I \propto \sigma\sqrt{Q/V}$). This assumes liquidity is always available at a price. In reality, order books exhibit *stochastic gaps*. During a flash crash, liquidity does not just become expensive; it often evaporates entirely (liquidity black holes). Our model likely underestimates the cost of "panic dumping" in these extreme scenarios.
2. **Absence of Limit Order Book (LOB) Simulation:** Our simulation calculates impact analytically and executes exclusively via Market Orders (Taker flow). We do not simulate the queue position or fill probability of Limit Orders. A production-grade agent would utilize a full LOB simulator (e.g., ABIDES) to place Limit Orders (Maker flow), potentially capturing the spread to neutralize the 3 bps insurance premium.

8 Conclusion

8.1 Summary

This project successfully applied Deep Reinforcement Learning (PPO) to the problem of optimal trade execution under extreme volatility constraints. By engineering a symmetric reward function that penalizes tracking error against the Arrival Price, we trained an agent that solves the "Impact-Time Risk" trade-off distinctly from standard benchmarks.

The agent generalizes robustly to unseen 2024 data, validating the "Sim-to-Real" transfer of our GARCH-based training methodology. Crucially, the model demonstrates a statistically significant Asymmetric Payoff Profile: it accepts a marginal cost of ~ 3 bps in normal regimes to secure a $+67$ bps relative advantage (tail risk reduction) during market crashes.

8.2 Final Verdict: The "Crystalized Exit"

The final model is not an "Alpha Generator" for profit maximization; it is a tool for Wealth Preservation.

Institutional investors often face a specific psychological challenge: "The Crystalized Exit." Consider a fund manager who has generated a 50% return on a position and wishes to liquidate.

- **TWAP is "Greedy by Negligence":** By passively spreading execution over 4 hours, TWAP exposes the realized profits to unnecessary variance.
- **SOTA RL is "Greedy by Design":** By attempting to squeeze an extra 5 bps of execution alpha, standard RL agents take on time risk that is unacceptable for a manager whose primary goal is locking in gains.

Our agent is the only solution aligned with the objective of the risk-averse liquidator. It functions as an automated execution trader that effectively says: *"I will pay the spread and a small premium to get you out safely. If the market turns against us, I will not wait for a better price; I will liquidate immediately to protect your PnL."*

8.3 Future Work

To further bridge the gap between simulation and institutional production, future research should expand along the following dimensions:

- **Regime-Adaptive Risk Preferences:** By integrating momentum signals, the agent could learn to toggle between "Insurance Mode" (high risk aversion) during volatility and "Alpha Mode" (low risk aversion) during stable trends.
- **Cross-Venue Smart Order Routing (SOR):** Cryptocurrency liquidity is fragmented. Executing a whale-sized order on a single venue artificially inflates impact. Future iterations should expand the action space to include venue selection, splitting flow across centralized and decentralized pools to minimize aggregate slippage.
- **Transformer Architectures:** Replacing the current MLP state encoder with a Temporal Fusion Transformer (TFT). This would improve the agent's ability to capture long-range volatility dependencies and regime shifts that simple lag features may miss.
- **Limit Order Integration:** Expanding the action space to include Limit Orders. This would allow the agent to capture the bid-ask spread in quiet markets, potentially neutralizing the 3 bps insurance premium observed in our results.

References

[1] Almgren, R., & Chriss, N. (2000). *Optimal execution of portfolio transactions*. Journal of Risk, 3(2), 5-39.

[2] Donier, J., & Bonart, J. (2015). *A million metaorder analysis of market impact on Bitcoin*. Market Microstructure and Liquidity, 1(02), 1550008.

[3] Maitrier, M., et al. (2025). *The Double Square-Root Law: Market Impact in Crypto vs. Equities*. Quantitative Finance Letters.

[4] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms*. arXiv preprint arXiv:1707.06347.

[5] Fei, Y., Yang, Z., Chen, Y., Wang, Z., & Xie, Q. (2020). *Risk-Sensitive Reinforcement Learning: Near-Optimal Risk-Sample Tradeoff in Regret*. NeurIPS 2020 / arXiv:2006.13827.

[6] Duflot, E., & Robineau, S. (2025). *RL-Exec: Impact-Aware Reinforcement Learning for Opportunistic Optimal Liquidation*. arXiv preprint arXiv:2511.07434.

[7] Schnaubelt, M. (2022). *Deep reinforcement learning for the optimal placement of cryptocurrency limit orders*. European Journal of Operational Research.

[8] Talos Trading. (2025). *Institutional Market Impact Benchmarks: Understanding Liquidity in the ETF Era*. Talos Research Whitepaper, Dec 2025.