REGIME-AWARE META-CONTROL FOR DEEP REINFORCEMENT LEARNING IN FINANCIAL TRADING

Cole Krudwig
Heider College of Business
Creighton University
Omaha, Nebraska
`ColeKrudwig@creighton.edu`

November 29, 2025

ABSTRACT

This project investigates regime-aware reinforcement learning for equity trading by combining classical ideas from stochastic portfolio theory with modern deep reinforcement learning. Market regimes such as trending, mean-reverting, and high-volatility phases are represented implicitly through a panel of diffusion-inspired diagnostics, including rolling drift, realized volatility, volatility trend, mean-reversion residuals, and price velocity. On top of these features, we construct a small set of baseline trading strategies (momentum, mean reversion, defensive), each mapping the current state to a risky-asset weight and an implied cash allocation in a risk-free asset. A deep reinforcement learning meta-controller is then trained to select among these strategies rather than directly outputing raw portfolio weights. We display that the meta-controller can achieve competitive or superior risk-adjusted performance relative to these baselines, while preserving a clear interpretation in terms of regime-aware strategy selection.

1 Introduction

Reinforcement learning (RL) has become an increasingly popular framework for sequential decision making in quantitative finance, particularly for portfolio allocation and trading strategies under uncertainty [1]. Classical stochastic control formulations, such as those based on the Hamilton–Jacobi–Bellman (HJB) equation, provide principled optimality conditions for continuous-time portfolio problems under utility maximization, but are often analytically intractable in realistic market settings.

While the classical Merton portfolio problem leads to a continuous-time HJB partial differential equation for the value function, our implementation does not solve this PDE explicitly. Instead, we adopt the same CRRA utility and wealth dynamics, and approximate the optimal control policy in discrete time via deep reinforcement learning, using a reward proportional to the change in utility.

In this work, we develop a hybrid approach that combines an HJB-motivated objective with deep RL in a discrete-time, data-driven setting [2]. We construct a custom trading environment, `TradingEnv`, where the agent's goal is to maximize changes in the constant relative risk aversion (CRRA) utility of wealth while allocating between a risky equity and a risk-free asset. Rather than directly choosing continuous portfolio weights, the agent selects among a discrete set of interpretable regime-specific strategies. The strategies include a momentum/trend-following strategy, a mean-reversion strategy, and a defensive strategy with a moderate short exposure. These policies map the current feature vector and wealth to a risky-asset weight π_t and an implied cash weight $1 - \pi_t$.

We train off-policy (DQN) and on-policy (PPO, A2C) deep RL algorithms to act as a meta-controller over these expert policies and evaluate performance against classical portfolio benchmarks. The codebase is structured as a reproducible pipeline, with command-line tools for training and evaluation, and supports multiple symbols via automatic data fetching from `yfinance`.

The main contributions of this work are:

- A custom Gymnasium environment [3] implementing an HJB-inspired CRRA utility objective and a discrete action space over regime-specific trading strategies.
- A modular Python framework for training DQN/PPO/A2C agents and evaluating them against interpretable baselines (buy-and-hold, 60/40, and each base strategy alone).
- An empirical study on daily equity index data, demonstrating that a DQN meta-controller can improve risk-adjusted performance relative to simple benchmarks, with detailed analysis of Sharpe ratios, wealth paths, and regime usage.

2 Related Work

There is now a substantial literature on reinforcement learning in finance. Recent surveys, such as Fischer [4] and Yu [5], review applications ranging from portfolio management and execution to option pricing and risk control, and highlight open challenges in robustness, evaluation, and market realism. These works situate RL methods within a broader ecosystem of quantitative strategies and emphasize the need for careful reward design and realistic market assumptions.

Deep reinforcement learning has been applied directly to trading and portfolio problems using both discrete and continuous action spaces. Zhang et al. [6] propose DQN- and policy-gradient-based strategies for futures contracts and compare their performance against classical time-series momentum benchmarks, while Millea [7] provides a critical survey of deep RL trading systems, documenting common design patterns (reward shaping, feature engineering, action parameterizations) and recurring pitfalls such as overfitting and lack of robust baselines. Our work follows this line by adopting standard deep RL algorithms (DQN, PPO, A2C) and implementing them with Stable-Baselines3 [8], but focuses specifically on regime selection rather than direct position sizing.

On the theoretical side, our use of CRRA utility and wealth dynamics is motivated by the continuous-time portfolio optimization framework of Merton [2], where the optimal policy solves a Hamilton–Jacobi–Bellman (HJB) equation. In contrast to this analytic approach, we approximate the optimal control numerically via deep RL in a discrete-time setting. Methodologically, our environment design is aligned with the OpenAI Gym/Gymnasium interface [3], and our regime-based action space can be viewed as a structured alternative to the more common direct weight or trade-size parameterizations used in prior RL-for-trading work [6, 7].

3 Data and Feature Engineering

3.1 Data Source

Historical daily OHLCV data for a given symbol (e.g., SPY) are obtained using the `yfinance` Python package. For each symbol S, we download a time series of:

$$\{\text{open}_t, \text{high}_t, \text{low}_t, \text{close}_t, \text{volume}_t\}_{t=1}^{T}.$$

We compute log returns of the closing price as

$$r_t = \log\left(\frac{\text{close}_t}{\text{close}_{t-1}}\right), \quad t = 2, \ldots, T. \tag{1}$$

The final dataset is stored as a CSV file with columns

`open`, `high`, `low`, `close`, `volume`, `Log_Return`.

For each symbol, we download daily data starting on 1 January 1990 or from the earliest available listing date for that equity. Our study primarily evaluates models trained and tested on broad equity index funds, but we also include preliminary experiments on riskier asset classes such as cryptocurrencies and individual large-cap stocks.

3.2 Feature Engine

We construct a panel of hand-crafted features using a custom class `FeatureEngine`. These features are designed to capture short- and long-horizon drift, realized volatility, volatility trends, mean reversion, and price velocity. The main standardized features (z-scores) included in the environment are:

- `Drift_Short_Z`: short-horizon drift signal,
- `Drift_Long_Z`: long-horizon drift signal,
- `RV_Signal_Z`: realized volatility or variance signal,
- `Vol_Trend_Z`: trend in volatility,
- `MR_Residual_Z`: mean reversion residual,
- `Price_Velocity_Z`: scaled price velocity.

The feature engine takes the raw OHLCV data, computes these quantities over rolling windows, normalizes them, and merges them back into the main DataFrame. The final feature matrix is merged with the price data using an inner join on the date index to ensure alignment.

The rolling window length for all short-horizon indicators was set to 21 trading days, reflecting roughly one calendar month of activity. This choice was informed by prior work suggesting that 20–22 day horizons are effective for capturing short-term trends in equity markets [9] and by preliminary sensitivity analyses in which alternative window sizes yielded either noisier signals or slower reaction to regime changes.

4 Trading Environment

4.1 Wealth Dynamics and Reward Function

We consider a single risky asset with log return r_t and a constant risk-free rate r_f. Let W_t denote the portfolio wealth at time t, and let π_t denote the fraction of wealth allocated to the risky asset. The remaining $1 - \pi_t$ is allocated to the risk-free asset. The one-period wealth update is:

$$W_{t+1} = W_t \left(1 + \pi_t r_t + (1 - \pi_t) r_f\right). \tag{2}$$

In our implementation, we fix an annual risk-free rate R_f at 2%, and convert it to an effective per-step rate r_f under a trading-days-per-year assumption. We also initialize wealth at $W_0 = 10{,}000$.

The investor's preferences are modeled using CRRA (power) utility:

$$U(W) = \frac{W^{1-\gamma}}{1-\gamma}, \quad \gamma > 1, \tag{3}$$

with risk-aversion parameter $\gamma = 2$ in the current implementation. The base quantity that drives the reward is the change in utility

$$\Delta U_t = U(W_{t+1}) - U(W_t). \tag{4}$$

In code, once the agent selects an action $a_t \in \{0, 1, 2\}$, the environment queries the `TradingStrategies` module to obtain a scalar portfolio return

$$R_t^{\text{port}} = \texttt{calculate\_strategy\_returns}(t, a_t),$$

which already incorporates the risky-asset exposure π_t and the implied allocation to the risk-free asset. Wealth is then updated according to

$$W_{t+1} = W_t\big(1 + R_t^{\text{port}}\big), \tag{5}$$

and the corresponding utilities $U(W_t)$ and $U(W_{t+1})$ are computed.

To discourage excessively volatile actions, we augment the pure utility-based signal with a simple quadratic penalty on large portfolio returns. Specifically, we define a risk-adjusted return

$$\tilde{R}_t = R_t^{\text{port}} - \lambda\big(R_t^{\text{port}}\big)^2, \tag{6}$$

where $\lambda = \texttt{LAMBDA} = 5$ controls the strength of the penalty on large single-period gains or losses. The final per-step reward is a weighted combination of the scaled utility change and this risk-adjusted return:

$$r_t = w_U \, c \, \Delta U_t + w_S \, \tilde{R}_t, \tag{7}$$

where $c = \texttt{REWARD\_SCALE} = 10^7$ is a numerical scaling factor, $w_U = \texttt{U\_WEIGHT} = 1$ controls the contribution of the HJB-motivated utility term, and $w_S = \texttt{S\_WEIGHT} = 10^2$ controls the contribution of the risk-adjusted return term.

Finally, episodes are terminated either when the end of the price series is reached or when the wealth falls below a ruin threshold (e.g., $W_t < 0.1 W_0$), in which case an additional negative reward is applied. This design encourages the agent to seek long-run growth in utility while avoiding extreme drawdowns and overly aggressive single-period bets.

4.2 State Representation

The environment TradingEnv is implemented as a Gymnasium Env with a continuous observation space and discrete action space. At each time step t, we construct a base observation vector

$$x_t = \left[z_t^\top, \tilde{W}_t\right]^\top \in \mathbb{R}^{d+1}, \tag{8}$$

where $z_t \in \mathbb{R}^d$ is the vector of standardized features

$$z_t = (\texttt{Drift\_Short\_Z}, \texttt{Drift\_Long\_Z}, \texttt{RV\_Signal\_Z}, \texttt{Vol\_Trend\_Z}, \texttt{MR\_Residual\_Z}, \texttt{Price\_Velocity\_Z}),$$

and $\tilde{W}_t = W_t/W_0$ is the normalized wealth relative to initial wealth W_0.

To provide temporal context, the observation presented to the agent consists of the last L base observations stacked into a single vector:

$$o_t = \left[x_{t-L+1}^\top, \ldots, x_t^\top\right]^\top \in \mathbb{R}^{L(d+1)}, \tag{9}$$

with a lookback length (e.g. LOOKBACK_STEPS $L = 21$). The observation space is implemented as a Box with appropriate bounds.

4.3 Action Space and Base Strategies

The action space is discrete with three actions:

$$\mathcal{A} = \{0, 1, 2\}. \tag{10}$$

Each action corresponds to invoking a base trading strategy from the TradingStrategies class:

- **Action 0: Momentum Strategy.** Allocates fully to the risky asset:

$$\pi_t^{\text{mom}} = 1.0.$$

- **Action 1: Mean Reversion Strategy.** Uses the standardized residual MR_Residual_Z to scale exposure against deviations from the mean:

$$\pi_t^{\text{mr}} = \text{clip}\left(-0.5 \cdot \texttt{MR\_Residual\_Z}_{t-1},\ -1.0,\ 1.0\right),$$

  where values are capped in $[-1, 1]$.

- **Action 2: Defensive Strategy.** Holds a fixed small short position:

$$\pi_t^{\text{def}} = -0.5.$$

The corresponding cash weight is $1 - \pi_t$ in each case.

Given the chosen action a_t, the environment computes the risky allocation π_t via the appropriate strategy and updates wealth accordingly.

4.4 Step Function

The step method of TradingEnv implements the core environment transition given an action $a_t \in \{0, 1, 2\}$. Conceptually, one call to step corresponds to moving forward by one trading day, updating wealth according to the chosen regime-specific strategy, computing a reward based on the change in utility and the realized portfolio return, and then returning a new observation together with termination flags.

Operationally, the environment maintains an integer index current_step that points to the current row in the feature and return DataFrame, as well as the current wealth W_t and the last utility value $U(W_t)$. When step(action) is called, the environment first increments the time index and checks whether it has moved beyond the last valid time step. If current_step is greater than or equal to max_steps, the episode is considered finished: no further trades are executed, the reward is set to zero, and the environment records the final wealth in the info dictionary under the key "terminal_wealth". This allows downstream evaluation code to recover the terminal wealth even though no additional portfolio update occurs at the final step.

If the episode has not yet terminated, the environment proceeds to compute the one-period portfolio return induced by the selected action. It calls the TradingStrategies object with the new time index and the action:

```
portfolio_return = strategies.calculate_strategy_returns(current_step,
action).
```

This method encapsulates the logic of the momentum, mean-reversion, or defensive strategy, returning a scalar portfolio return R_t^{port} that already reflects both the risky position and the implied risk-free allocation. The environment then produces the reward for the current time step and enforces the ruin constraint. Finally, the environment updates its internal observation history by appending the new base observation and discarding the oldest one. Algorithm 1 summarizes the environment transition for a single step.

Algorithm 1 `TradingEnv.step(action)`

```
Require: current time index $t$, current wealth $W_t$, action $a_t \in \{0, 1, 2\}$
 1: Increment time: $t \leftarrow t + 1$
 2: if $t$ exceeds last valid index then
 3:     return terminal observation, reward 0, terminated=True
 4: Compute risky allocation $\pi_t$ from TradingStrategies given $a_t$
 5: Observe risky log return $r_t = \text{Log\_Return}_t$
 6: Compute portfolio return:
        $R_t^{\text{port}} = \pi_t r_t + (1 - \pi_t) r_f$
 7: Update wealth: $W_{t+1} \leftarrow W_t(1 + R_t^{\text{port}})$
 8: Compute utility: $U_{t+1} \leftarrow U(W_{t+1})$, reward $r_t^{\text{env}} = c(U_{t+1} - U_t)$
 9: Update utility state: $U_t \leftarrow U_{t+1}$
10: Build new observation $o_{t+1}$ from updated history and wealth
11: Check ruin condition (e.g., $W_{t+1} < 0.1 W_0$) and set terminated if triggered
12: return $o_{t+1}$, $r_t^{\text{env}}$, terminated, truncated=False, info
```

5 Reinforcement Learning Algorithms

5.1 Algorithms

We train three standard deep reinforcement learning algorithms from Stable-Baselines3:

- **DQN** (Deep Q-Network), an off-policy value-based method suitable for discrete action spaces.
- **PPO** (Proximal Policy Optimization), an on-policy policy gradient method.
- **A2C** (Advantage Actor-Critic), a synchronous actor–critic method.

All models use an MLP policy with a shared two hidden layer network architecture consisting of 128 units each. Hyperparameters such as learning rate, discount factor, exploration schedule (for DQN), and batch size are selected based on preliminary experiments and standard defaults. Table 1 displays the selected hyperparameters for the three models.

Table 1: Hyperparameters for DQN, PPO, and A2C.

	DQN	PPO	A2C
Policy	MlpPolicy	MlpPolicy	MlpPolicy
Network	[128,128]	[128,128]	[128,128]
Learning rate	e-2	3e-4	7e-4
Discount γ	0.99	0.99	0.99
Batch size	64	54	5
Total timesteps	10,000	10,000	10,000

5.2 Training and Evaluation Pipeline

The codebase exposes a command-line interface (`cli.py`) with two main commands:

- `train`: trains DQN, PPO, and A2C models on a specified symbol. If the corresponding CSV does not exist in the `data/` directory, the script invokes the `Fetch` class to download and preprocess the data.

- `evaluate`: evaluates all trained models on the final $(1 - s)$ fraction of the data, where s is a user-specified train split. Evaluation includes both RL models and baseline benchmarks.

We employ an 80/20 time-based split by default: the first 80% of observations are used for training, and the final 20% for out-of-sample evaluation. All experiments are conducted on the test slice only once to avoid lookahead bias.

6 Results

6.1 Benchmark Performance

Table 2 reports the performance of the benchmark strategies on the held-out test period for a representative experiment on SPY. All strategies are initialized with $W_0 = \$10{,}000$ and evaluated over $1{,}275$ trading days in the final test segment.

Table 2: Benchmark performance on the final test segment.

Strategy	Steps	Final Wealth ($)	Total Return (%)	Sharpe
Buy-and-Hold 100%	1275	18,385.03	83.85	0.786
60/40	1275	15,271.66	52.72	0.862
Momentum-only	1275	18,385.03	83.85	0.786
Mean-reversion-only	1275	13,124.21	31.24	0.620
Defensive-only	1275	8,102.66	-18.97	-0.440

Several observations follow from Table 2. First, the buy-and-hold and momentum-only strategies are numerically identical in this setup: both end with a final wealth of approximately $18,385 and an annualized Sharpe ratio of 0.786. This is expected, since the momentum base strategy in our implementation simply maintains a fully long position in the risky asset (i.e., $\pi_t \equiv 1$), making it effectively equivalent to buy-and-hold when applied continuously over the test horizon.

Second, the constant 60/40 portfolio delivers a lower total return than buy-and-hold (about 53% vs. 84%), but achieves a higher Sharpe ratio of 0.862. This highlights the classical trade-off between absolute performance and risk-adjusted performance: the partial allocation to the risk-free asset reduces volatility sufficiently to improve the Sharpe ratio, even though the final wealth is lower than that of a fully invested strategy.

Third, the mean-reversion-only strategy underperforms both buy-and-hold and 60/40 in this particular sample, ending with a final wealth of roughly $13,124 (a 31% total return) and a Sharpe ratio of 0.620. This suggests that the mean-reversion signal, when followed mechanically at all times, is not strong enough to compensate for the noise and trend structure present in this SPY test period. Finally, the defensive-only strategy performs worst, losing capital on average and exhibiting a negative Sharpe ratio. A permanently defensive stance is costly in an overall rising market and serves mainly as a hedge during drawdowns.

Taken together, these benchmarks provide a useful backdrop for evaluating the RL agents: buy-and-hold represents a naive high baseline, 60/40 a simple risk-managed allocation, and the three base strategies illustrate the behavior of each regime in isolation.

6.2 RL Models vs. Benchmarks

Table 3 displays the three RL agents (DQN, PPO, and A2C). All agents are trained on the first 80% of the data and evaluated once on the final 20% segment, using identical price and feature trajectories.

Table 3: Comparison of RL models on the final test segment.

Model / Strategy	Steps	Final Wealth ($)	Total Return (%)	Sharpe
DQN	1275	19,646.65	96.47	0.941
PPO	1275	13,518.35	35.18	0.597
A2C	1275	16,746.36	67.46	0.681

The DQN meta-controller achieves the strongest overall performance in this configuration. Starting from $10,000, it ends the test period with a final wealth of approximately $19,646, corresponding to a total return of about 96% and an annualized Sharpe ratio of 0.941. Both the absolute return and the Sharpe ratio exceed those of the buy-and-hold and momentum-only benchmarks (final wealth \$18,385, Sharpe 0.786), and the Sharpe ratio also exceeds that of the more conservative 60/40 portfolio (Sharpe 0.862). This indicates that the DQN agent is able, at least in this sample, to use regime switches between the base strategies to achieve both higher growth and better risk-adjusted performance than simple static allocations.

The other two RL algorithms, PPO and A2C, perform more modestly. PPO attains a final wealth of roughly \$13,518 (a 35% total return) with a Sharpe ratio of 0.597, underperforming both buy-and-hold and 60/40 on a risk-adjusted basis. A2C represents a middle ground, achieving a final wealth of about \$16,746 (a 67% total return) and a Sharpe ratio of 0.681, better than the mean-reversion-only and defensive benchmarks but still below the DQN and 60/40 strategies in Sharpe. These results suggest that, for this environment and hyperparameter configuration, the off-policy DQN agent is better suited to learning a stable mapping from the lookback feature history and wealth to regime choices than the on-policy PPO and A2C methods.

6.3 Strategy Selection and Behavior

Beyond aggregate performance metrics, an advantage of the regime-switching formulation is that the agent's behavior can be interpreted in terms of how frequently and under what conditions it selects each base strategy. To analyze this, we log the sequence of actions chosen by the meta-controller over the test period. Figure 1 displays the behavior of our trained models through the count of actions selected.

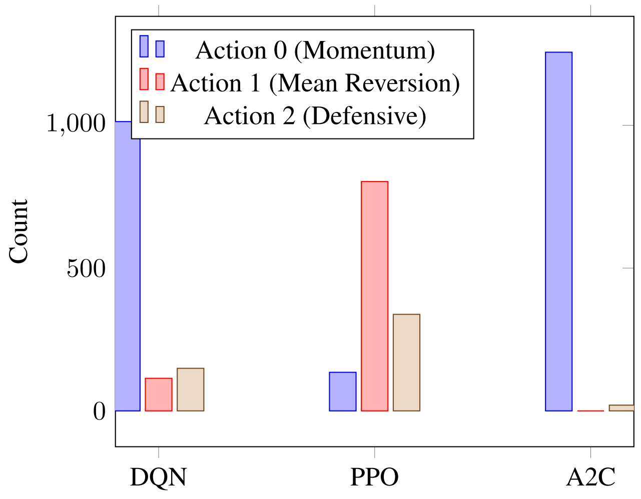

Figure 1: Action selection bars for DQN, PPO, and A2C on the test segment.

Figure 2 shows the equity curves of the three agents over the test horizon. Comparing these trajectories with the corresponding action distributions, we see how regime selection translates into performance over time. Although the A2C agent exhibits an action profile broadly similar to that of DQN, its relatively sparse use of the mean-reversion and defensive regimes coincides with a lower terminal wealth. By contrast, the PPO agent shifts into mean-reversion and defensive actions more frequently than the other two models, and this more conservative regime mix is reflected in its substantially weaker overall performance.

6.4 Generalization to Other Assets

To assess whether the proposed regime-switching framework and trained agents are specific to SPY or exhibit broader applicability, we repeat the same training and evaluation pipeline on a small panel of additional assets. In particular, we consider:

- **QQQ:** the NASDAQ-100 ETF, representing a growth- and technology-heavy equity index with higher volatility than SPY;
- **NVDA:** a large-cap single stock with pronounced trends and drawdowns, illustrating performance on idiosyncratic equity risk; and
- **BTC-USD:** spot Bitcoin, providing an example of a highly volatile crypto asset with distinct regime structure and non-equity return dynamics.

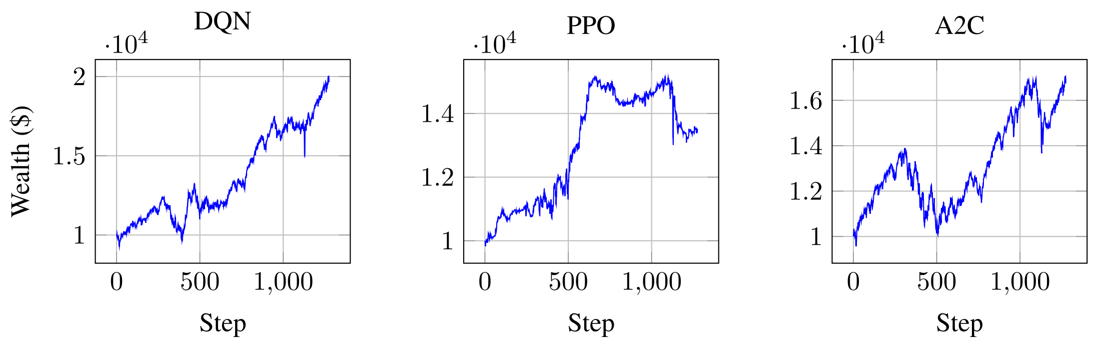

Figure 2: Equity curves for DQN, PPO, and A2C on the test segment.

For each symbol, we use daily OHLCV data fetched via `yfinance`, construct the same set of features, and employ the same train/test split protocol as in the SPY experiments (e.g., the first $s = 0.8$ fraction of observations for training and the remaining $1 - s$ for out-of-sample evaluation). Hyperparameters for DQN, PPO, and A2C are kept fixed across assets to isolate the effect of changing the underlying return distribution rather than re-tuning each model per symbol.

Table 4 summarizes the out-of-sample performance of the three RL agents on QQQ, NVDA, and BTC-USD, reporting final wealth, total return, and annualized Sharpe ratio. For each asset, we also include the buy-and-hold and 60/40 benchmarks to provide context on how challenging the underlying series is.

Table 4: Out-of-sample performance on additional assets.

Asset	Model / Strategy	Steps	Final Wealth ($)	Total Return (%)	Sharpe
QQQ	DQN	1311	17,774.64	77.75	0.653
	PPO	1311	15,910.25	59.10	0.695
	A2C	1311	19,948.68	99.49	0.697
	Buy-and-Hold	1311	19,866.75	98.67	0.693
	60/40	1311	16,249.35	62.49	0.751
NVDA	DQN	1317	25,988.24	159.88	0.626
	PPO	1317	10,233.33	2.33	0.163
	A2C	1317	63,773.99	537.74	0.944
	Buy-and-Hold	1317	63,773.99	537.74	0.944
	60/40	1317	37,436.77	274.37	0.969
BTC-USD	DQN	785	26,861.02	168.61	1.063
	PPO	785	12,480.18	24.80	0.413
	A2C	785	25,460.56	154.61	0.952
	Buy-and-Hold	785	25,460.56	154.61	0.952
	60/40	785	19,043.44	90.43	0.985

Across the three additional assets, the regime-switching agents do not uniformly dominate the simple benchmarks, but several consistent patterns emerge. On QQQ, all strategies perform reasonably well in absolute terms, with final wealth roughly doubling over the test horizon. The A2C agent tracks buy-and-hold very closely in both return and Sharpe ratio, while DQN and PPO achieve slightly lower final wealth. The 60/40 allocation attains the highest Sharpe ratio on QQQ, suggesting that for a growth-heavy but still equity-like index, simple risk-smoothing via partial allocation to the risk-free asset remains highly competitive with learned regime switching.

For NVDA and BTC-USD, which exhibit much stronger trends and higher volatility, the behavior changes. On NVDA, A2C essentially reproduces the buy-and-hold strategy, matching both its very high total return and Sharpe ratio, while 60/40 sacrifices some upside in exchange for a modest improvement in Sharpe. DQN delivers solid but clearly inferior performance, and PPO barely outperforms cash. On BTC-USD, DQN is the standout model, achieving both the highest final wealth and the highest Sharpe ratio among all strategies, including buy-and-hold, while A2C again closely tracks the long-only benchmark. PPO remains the weakest of the three agents across all assets. Overall, these results suggest that the proposed meta-controller can adapt to very different return environments, but its advantage is most pronounced in highly volatile, regime-rich series like BTC-USD, while on strongly trending single names it tends to converge toward buy-and-hold–like behavior.

7 Conclusion and Future Work

We proposed a regime-aware reinforcement learning framework for single-asset trading, built around a custom Gymnasium environment with CRRA utility, a risk-free asset, and diffusion-inspired features. Instead of learning raw portfolio weights, a DQN/PPO/A2C meta-controller selects among three interpretable base strategies (momentum, mean reversion, defensive), which then determine the risky and risk-free allocations.

On SPY, the DQN meta-controller achieved higher final wealth and Sharpe ratio than buy-and-hold and a 60/40 portfolio, while PPO and A2C were weaker but still competitive. Cross-asset experiments on QQQ, NVDA, and BTC-USD showed that the approach is not uniformly superior: A2C often converges toward buy-and-hold behaviour on strongly trending assets, DQN shines on more volatile, regime-rich series like BTC-USD, and simple 60/40 allocations remain difficult to beat in some cases. Overall, the results suggest that HJB-guided regime switching can produce economically sensible policies, but its advantage is context-dependent.

Future work will focus on three directions: (i) refining the reward and risk controls (e.g., drawdown-aware or cost-aware objectives), (ii) extending the framework to multi-asset portfolios with transaction costs and more realistic execution, and (iii) improving robustness and generalization via richer features, more advanced RL algorithms, and systematic evaluation across assets, time periods, and random seeds.