Master's Project: AI Agents in Perpetual Futures Markets

Lazaro N. Hofmann (19-731-306)
Lennart P. Eikens (21-933-106)
Ivan Isaenko (24-743-809)
Arber Fetahu (19-751-858)
Zhongjian Yu (23-742-216)

University of Zurich

Abstract. This study investigates the use of reinforcement learning (RL) for algorithmic trading and statistical arbitrage in cryptocurrency markets, with a focus on perpetual futures contracts. The framework combines reinforcement learning–based portfolio decision–making with a process-automated research and execution pipeline.
An RL agent trained to exploit relative price dynamics across crypto-assets was evaluated out-of-sample under multiple transaction fee regimes. The empirical results exhibit strongly conditional performance. The model was trained on Binance spot and perpetual futures data spanning May 2024 to April 2025, covering the 50 largest currencies by market capitalization. In a frictionless environment, the learned policy achieves a positive cumulative return. However, performance deteriorates substantially once transaction costs are introduced, with negative returns under both low and high fee structures. These results indicate that while the algorithm learns economically meaningful trading behavior, transaction costs constitute a binding constraint for RL-based statistical arbitrage strategies.
In addition to the empirical analysis, the study presents a proof of concept for automated trade execution. Leveraging a process automation layer based on large language model orchestration, the system enables end-to-end operation from signal generation to order placement. Overall, the findings highlight both the potential and the practical limitations of reinforcement learning in crypto-derivatives trading and underscore the importance of realistic cost modeling and robust system design.

Keywords: Artificial Intelligence · Decentralized Finance · Perpetual Futures Contracts

1 Introduction

Decentralized finance (DeFi) has evolved rapidly from simple token exchanges to complex on-chain and hybrid derivatives markets. A central development in this evolution is the rise of perpetual futures contracts (perpetuals), which have become a dominant trading instrument in cryptocurrency markets [1]. Perpetuals

allow traders to take leveraged long and short positions without an expiration date and rely on periodic funding payments to anchor prices to an underlying index [38].

Futures are derivative contracts that obligate parties to buy or sell an asset at a predetermined price on a specific future date [6]. While traditional fixed-term futures are standard in legacy finance, they are often less useful in the crypto ecosystem because liquidity is fragmented across multiple expiration dates (e.g., monthly or quarterly), making it difficult for traders to enter or exit large positions without significant slippage [39,10]. In contrast, perpetuals consolidate liquidity into a single, non-expiring instrument, fostering a more robust and efficient market environment [1].

Today, perpetuals account for the majority of cryptocurrency derivatives trading volume, often significantly exceeding spot market volume. As of 2024, the total monthly trading volume for crypto derivatives, dominated by perpetuals, has frequently surpassed $3 trillion, representing over 70% of the total crypto market share [3]. These instruments are actively traded on both centralized and decentralized platforms such as Binance, dYdX, and Hyperliquid.

Compared to spot markets, perpetuals offer several advantages for algorithmic trading. They enable cost-efficient short selling, provide deeper liquidity for major assets, and typically feature lower effective transaction costs, especially for high-frequency strategies. These properties make perpetuals a natural environment for market-neutral and relative-value strategies, including statistical arbitrage.

To maintain price parity with the underlying spot market in the absence of an expiration date, perpetuals utilize a funding rate mechanism—a periodic exchange of fees between long and short traders that incentivizes market participants to arbitrage any price divergence between the contract and the asset's index price [1]. However, the structure of perpetual markets also introduces additional frictions, such as funding payments, leverage constraints, and rapid regime shifts driven by speculative activity and liquidation dynamics [17].

Classical statistical arbitrage approaches, such as pair trading, are challenged in this setting. Statistical arbitrage is a quantitative trading strategy that exploits temporary price discrepancies between historically related assets, such as two correlated cryptocurrencies, by taking opposing long and short positions under the assumption that their prices will eventually revert to a mean relationship. While cross-asset relationships may exist, they are often unstable, and frequent rebalancing can lead to substantial transaction costs that erode theoretical profits [25]. As a result, static rule-based strategies may struggle to remain economically viable in DeFi perpetual markets where the basis and funding rates can fluctuate violently.

In this context, reinforcement learning (RL) has emerged as a promising alternative. Reinforcement learning agents optimize sequential decisions through interaction with an environment, allowing them to adapt position sizing, timing, and exposure dynamically in response to market conditions and trading frictions [12]. This project adopts a hybrid finance–machine learning perspective

and investigates whether RL-based AI agents can learn economically meaningful trading policies for statistical arbitrage in cryptocurrency perpetual markets under realistic cost structures. By focusing on both learning behavior and practical execution, the study aims to assess the extent to which reinforcement learning can address the limitations of traditional arbitrage strategies in modern DeFi derivatives markets [15].

Related Work

Statistical arbitrage encompasses a variety of quantitative investment strategies that exploit temporary price differences between similar assets, relying on the premise that these deviations will eventually revert to a historical equilibrium [46,33]. The strategy typically unfolds in two distinct phases: the formation phase, where assets with historical co-movements are identified, and the trading phase, where positions are executed based on spread deviations [33].

While originally developed for traditional equity markets, statistical arbitrage has found fertile ground in the cryptocurrency domain due to the market's high volatility, 24/7 operation, and fragmented liquidity [47,13]. Research indicates that the cryptomarket remains relatively inefficient compared to mature markets, potentially offering significant arbitrage opportunities [13]. However, the high-frequency nature of these markets introduces challenges regarding execution latency and transaction costs, necessitating automated, algorithmic approaches [47,37].

Traditional statistical arbitrage typically relies on econometric models and heuristic rules. The seminal distance method approach proposed by Gatev et al. [14] selects pairs by minimizing the sum of squared deviations between normalized price series. Trading signals are triggered when the spread diverges by a fixed multiple of its historical standard deviation (e.g., two standard deviations).

A more rigorous statistical approach involves cointegration, often utilizing the Engle-Granger or Johansen tests to ensure a long-term stationary relationship between assets [23]. Researchers [26,33] have also modeled spreads using the Ornstein–Uhlenbeck (OU) process, trading based on the mean-reversion speed and volatility parameters estimated via maximum likelihood.

Despite their historical success, these classic methods face significant limitations in modern cryptomarkets. Conventional methods often assume linear relationships between asset returns, failing to capture the non-linear dynamics inherent in volatile markets [36]. Also, rule-based approaches typically rely on fixed entry and exit thresholds determined during a formation period. These static parameters often fail to adapt to the rapid regime changes and volatility clustering characteristic of cryptocurrency markets [33,22]. Performance is highly sensitive to the choice of hyperparameters (e.g. window size, z-score thresholds), yet robust methods for optimal selection remain elusive in traditional frameworks [33].

To overcome the rigidity of rule-based systems, recent literature has shifted toward Machine Learning (ML) and Deep Reinforcement Learning (DRL). These

methods allow for the development of dynamic agents capable of end-to-end learning, mapping market states directly to optimal trading actions [42].

Initial shifts toward ML involved using supervised learning to forecast spread movements. Fischer et al. demonstrated that a Random Forest classifier could successfully exploit short-term mean reversion in cryptocurrency markets using 1-minute data, outperforming naive buy-and-hold strategies even after accounting for transaction costs [13]. Similarly, Convolutional Auto-Encoders (CAE) have been employed to extract latent non-linear risk factors for more robust pair selection before applying trading strategies [36].

Reinforcement Learning (RL) represents the frontier of statistical arbitrage, treating trading as a Markov Decision Process (MDP) where agents maximize cumulative rewards [33]. Unlike static rules, RL agents can learn dynamic policies. For example, Kim et al. [22] proposed "HDRL-Trader," a hybrid architecture combining a Twin Delayed Deep Deterministic Policy Gradient (TD3) for trading actions and a Double Deep Q-Network (DDQN) specifically for determining dynamic stop-loss boundaries, significantly reducing drawdown risk.

In the context of high-frequency crypto data, Yang and Malik [47] introduced a "Dynamic Scaling" approach. Their Advantage Actor-Critic (A2C) agent learned not only when to trade but how much to invest based on the quality of the opportunity, achieving an annualized profit of 31.53% on 1-minute Bitcoin pair data. Vergara and Kristjanpoller [46] conducted a comparative analysis of DRL agents (DQN, A2C, PPO) in the crypto market. They found that Deep Q-Networks (DQN) outperformed other agents and traditional cointegration strategies, achieving superior risk-adjusted returns.

Adapting RL for financial stakes may require specific risk-aversion mechanisms. Brim demonstrated that introducing a "Negative Rewards Multiplier" during training effectively teaches DDQN agents to adopt a conservative posture, taking "no position" actions when confidence is low. This is crucial for crypto perpetuals, where leverage and volatility can lead to rapid liquidation [4].

This work extends the literature by applying reinforcement learning to statistical arbitrage in cryptocurrency markets, with particular emphasis on high-frequency (1-min) perpetuals data across multiple asset pairs and on integrating reinforcement learning models with cointegration-driven pair selection within an agent-based system.

Contributions

This work makes the following contributions:

- It develops a reinforcement learning–based framework for statistical arbitrage in cryptocurrency perpetual futures markets, in which the algorithm learns portfolio allocation policies over cointegrated asset pairs using one-minute high-frequency data.
- It provides an out-of-sample empirical analysis of the sensitivity of reinforcement learning–based trading strategies to transaction costs using high-frequency cryptocurrency data from perpetual futures markets on Binance,

covering multiple cointegrated asset pairs denominated in USD stablecoins over the period from May 2024 to April 2025. The results show that, while the strategy achieves a total return of approximately 42% in a frictionless setting, performance deteriorates to a loss of about 16% under a low-fee regime (1.44 bps) and to approximately 32% under a high-fee regime (4.50 bps), quantifying the extent to which transaction costs impact realized profitability.
– It presents a proof of concept for an end-to-end, process-automated trading system, integrating signal generation, evaluation, and automated trade execution through a large language model–based orchestration layer.

All codes are available on GitHub and GitLab. Furthermore, raw and preprocessed data can be found on OneDrive to support scientific transparency and reproducibility.

Paper Organization Section 2 introduces the theoretical foundations underlying this study, including core concepts in reinforcement learning, decentralized finance, and cryptocurrency market microstructure. Section 3 reviews classical arbitrage and statistical arbitrage strategies, with a focus on cointegration-based trading and their applicability to perpetual futures markets. Section 4 describes the overall system architecture, including the trading venues, data acquisition and preprocessing pipeline, rolling cointegration framework, feature engineering, and the reinforcement learning formulation encompassing state, action, and reward design. Section 5 details the execution and orchestration layer, presenting the Hummingbot-based execution framework and the AI agent architecture built using LangGraph and LangSmith. Section 6 presents the evaluation of the reinforcement learning agent, reporting out-of-sample performance across different transaction fee regimes and benchmark comparisons. Section 7 discusses the main empirical findings and their implications, and Section 8 concludes the paper by summarizing the results and outlining directions for future research.

2 Background

2.1 Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning that focuses on sequential decision-making under uncertainty. In the context of machine learning, RL is distinct in that it involves training agents to make decisions based on feedback from the environment. Feedback which, crucially, may be incomplete or delayed. Unlike supervised learning, which relies on labeled datasets, or unsupervised learning, which seeks patterns in unlabeled data, RL involves an *agent* that learns to interact with an *environment* by trial and error. The agent receives *feedback signals (rewards or punishments)* and adapts its behavior to maximize long-term outcomes [44]. This paradigm has gained increasing importance, particularly with the advent of deep reinforcement learning (DRL), which has enabled agents to solve complex, high-dimensional tasks such as playing games, controlling robots, and optimizing resource allocation [30,40].

Reinforcement Learning in Finance

Many financial decision are inherently sequential, meaning that decisions (e.g., order placement, position sizing, rebalancing) affect future states through inventory, cash, and market impact. This makes them expressible as Markov decision processes, where the state combines market information (prices, spreads, limit order book features) with agent variables (inventory, remaining time, risk limits) [16]. In this setting, reinforcement learning is interesting because it optimizes a policy directly against a user-defined objective computed from realized outcomes, rather than first fitting a forecasting model and then deriving decisions indirectly. Existing work emphasizes objectives beyond raw return, such as optimizing risk-adjusted performance measures (e.g., Sharpe-type criteria) within a direct RL framework [31].

For *execution optimization*, RL has already been trained on high-frequency limit order book data to learn execution policies that trade off price improvement, impact, and non-execution risk [32]. For *market making*, RL agents explicitly manage inventory risk while quoting in simulated limit order book environments [41]. For *portfolio allocation*, RL has been used to map market states to portfolio weights or trading actions, typically incorporating transaction costs and constraints in the environment and/or reward [20,48].

Concept and Model

The formal framework of reinforcement learning is often described using *Markov Decision Processes (MDPs)* [34]. An MDP is defined by the following components:

- **Environment:** The external system that responds to the agent's actions and evolves accordingly.
- **Agent:** The learner or decision-maker that interacts with the environment to achieve a goal.
- **States (S):** Representations of the environment at a given time.
- **Actions (A):** Possible decisions the agent can take.
- **Reward Function (R):** A scalar signal that indicates the immediate utility of an action in a given state.
- **Policy (π):** A strategy that maps states to actions, either deterministically or probabilistically.
- **Value Functions:** Expected long-term return for states or state-action pairs, often estimated through algorithms such as temporal-difference learning [43].

The agent's objective is to learn an optimal policy, π^*, that maximizes the expected cumulative discounted reward:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where $\gamma \in [0, 1)$ is the discount factor controlling the weight of future rewards [44].

Exploration vs. Exploitation

A fundamental challenge in reinforcement learning (RL) is the *exploration–exploitation trade-off*. Exploitation involves choosing actions that the agent already believes to yield high rewards, while exploration requires testing less-known actions to discover potentially better strategies [45]. Balancing these two aspects is critical, as excessive exploitation risks convergence to suboptimal policies, while excessive exploration can delay learning efficiency. Approaches to handle this trade-off include ϵ-greedy policies, *Upper Confidence Bound (UCB)* methods, and more sophisticated strategies such as *Thompson sampling* [2].

Python Libraries and Applications

The growth of open-source software has made reinforcement learning widely accessible for research and practice. Prominent Python libraries include:

- **OpenAI Gym:** A toolkit for developing and comparing RL algorithms in standardized environments [5].
- **Stable-Baselines3:** A popular implementation of RL algorithms with a focus on usability and reproducibility [35].
- **RLlib:** A scalable RL library integrated with Ray, supporting distributed training [27].
- **TF-Agents and PyTorch RL:** Frameworks that support DRL algorithms with deep learning backends.

Applications of RL span across multiple fields:

- **Game playing:** Atari benchmarks [30], AlphaGo [40].
- **Robotics:** Autonomous navigation and manipulation tasks [24].
- **Finance:** Portfolio optimization and algorithmic trading [31].
- **Healthcare:** Adaptive treatment strategies and drug dosing [49].

Challenges and Future Directions

Despite recent successes, reinforcement learning faces several challenges. Sample inefficiency, stability of training, and generalization to unseen environments remain open research problems [9]. Furthermore, ethical concerns arise when applying RL to sensitive domains such as healthcare or autonomous driving, where exploration may have real-world risks. Future directions include *safe RL*, *multi-agent reinforcement learning (MARL)*, and integration with *causal inference* to improve robustness and interpretability.

2.2 Perpetual Futures Contracts

Perpetual futures contracts, or perps, are derivatives that allow traders to speculate on the future price of an asset without ownership of said asset. In contrast to traditional futures, these contracts possess no expiration date. In 2016, the

cryptocurrency exchange BitMEX introduced these products, which have since become the leading derivative instruments in crypto markets. In 2023, they accounted for over 75% of Bitcoin futures trading volume.

Perps mimic traditional futures in their structure but differ crucially in their lack of maturity. In traditional futures, convergence to the spot price occurs naturally at expiration. Perps, however, require a mechanism to keep their price f_t aligned with the spot price x_t. In this context, the spot price represents the immediate current market value of the underlying asset (e.g., Bitcoin) for instant delivery. To reduce microstructure noise, exchanges typically compute funding and liquidation using an index-based mark price, so the funding mechanism and margining are tied to a reference price rather than transient prints.

Funding payments represent a recurring fee exchanged between holders of long and short positions. When the funding rate f_t exceeds the spot price x_t, those in long positions compensate short position holders. Conversely, when the funding rate falls below the spot price, short position holders pay long position holders. The fundamental objective of this mechanism is to enforce the Law of One Price. By creating a direct financial cost for holding a position that deviates from the spot market, the funding rate incentivizes arbitrageurs to close the gap. This effectively forces the perpetual price to mean-revert to the spot price, acting as a continuous, algorithmic substitute for the expiration date found in traditional contracts.

Most exchanges refresh the funding rate every eight hours, calculated as follows:

$$\text{Funding Rate}_t = \lambda(f_t - x_t) + \delta$$

where λ reflects the sensitivity of the funding rate to the basis (the difference between perp and spot), and δ accounts for interest rate differentials between the base and quote currencies [3].

Under no-arbitrage assumptions in a discrete-time model, the theoretical price of a linear perp contract can be expressed as:

$$f_t = \mathbb{E}_t^{\mathbb{Q}}\left[\sum_{n=0}^{\infty}\left(\frac{1}{1+\kappa}\right)^n (\kappa - \iota)\, x_{t+n}\right]$$

where κ is the anchoring premium rate and ι is the interest component reflecting funding pressure. This expression shows that the perp price equals the discounted expected spot price at a random future time, effectively turning the funding rate into a probabilistic weighting mechanism.

In specific configurations, notably when

$$\iota = r_a - r_b$$

where r_a and r_b are the risk-free rates of the quote and base currencies, respectively, the perp price converges exactly to the spot price. This enables replication through dynamic portfolios of the spot asset and risk-free bonds, highlighting the perp's synthetic nature.

Due to their continuous nature, low capital requirements (through leverage), and high liquidity, perps have become a cornerstone of decentralized and centralized crypto trading platforms alike, including dYdX, GMX, Binance, and ByBit [7,8].

3 Arbitrage Strategies in Perpetual Futures Markets

This chapter provides an overview of different arbitrage strategies and assesses their feasibility to determine the most appropriate option for our project. Arbitrage in the context of cryptocurrencies often involves exploiting market inefficiencies across spot, futures, and perpetual swap markets. The following subsections expand on the main arbitrage strategies relevant to our project, including their mechanisms, return potential, associated risks, and mathematical formulations.

3.1 Cross-exchange arbitrage

Cross-exchange arbitrage is the most classical form of arbitrage. The strategy consists of buying an asset on one exchange where the price is lower and simultaneously selling it on another exchange where the price is higher. These price discrepancies are often driven by capital controls and market segmentation rather than simple inefficiency [29].

Let P_A be the asset price on Exchange A and P_B on Exchange B. The gross arbitrage profit before transaction costs is given by:

$$\Pi = (P_B - P_A) \cdot Q$$

where Q is the traded quantity.

The net profit after including trading fees (f_A, f_B) and transfer cost (c_T) is:

$$\Pi_{net} = (P_B(1 - f_B) - P_A(1 + f_A) - c_T) \cdot Q$$

Risks include transfer delays (blockchain confirmation times), slippage if order books are thin, and sudden price movements that erase the spread. In the case of perpetual swaps, cross-exchange arbitrage may also involve capturing differences in funding rates between exchanges.

3.2 Cash-and-Carry arbitrage

Cash-and-carry arbitrage exploits the difference between the spot price S_t of an asset and its futures price $F_t(T)$ for maturity T. In contango ($F_t(T) > S_t$), one can buy the spot asset and simultaneously short the futures contract, thereby locking in a nearly risk-free return [28].

The theoretical no-arbitrage relationship between spot and futures is given by the cost-of-carry model:

$$F_t(T) = S_t \cdot e^{(r+u-y)(T-t)}$$

where: r = risk-free rate, u = storage/holding costs (including financing costs), y = convenience yield.

The arbitrage profit (ignoring transaction costs) is:

$$\Pi = (F_t(T) - S_t) \cdot Q$$

The annualized return can be approximated as:

$$R \approx \frac{F_t(T) - S_t}{S_t} \cdot \frac{365}{T-t}$$

Risks include capital being locked until futures expiry, changes in the futures premium due to liquidity shocks, and margin requirements for the short futures position.

3.3 Funding rate arbitrage

Perpetual futures (perps) introduce the concept of a funding rate, a periodic payment exchanged between long and short traders to keep futures prices aligned with spot prices [17].

A simple arbitrage setup is: Long Q units in the spot market (buy asset). Short Q units in the perpetual futures market.

The position is delta-neutral, so exposure to price movements is hedged. The profit per funding interval is:

$$\Pi = FR \cdot P \cdot Q \cdot \Delta t$$

where: FR = funding rate (per interval), P = mark price of the asset, Q = quantity, Δt = funding interval fraction of a day (e.g., $\frac{1}{3}$ if funding occurs every 8 hours).

Over a holding horizon H, the cumulative arbitrage return is:

$$R_{funding} = \sum_{i=1}^{H/\Delta t} FR_i \cdot P_i \cdot Q \cdot \Delta t$$

Risks include imperfect hedging if the spot and perp prices diverge (basis risk), sudden negative funding flips, and high borrowing costs when obtaining capital for the spot position.

3.4 Statistical Arbitrage

Statistical arbitrage relies on identifying pairs of assets whose prices are historically correlated or cointegrated. The idea is to buy the undervalued asset and short the overvalued one, betting that the spread will revert to its mean [25].

Define log-prices of two assets X_t and Y_t. A common model is:

$$Y_t = \alpha + \beta X_t + \epsilon_t$$

where ϵ_t is the stationary spread (residual). The spread is:

$$z_t = Y_t - (\alpha + \beta X_t) \tag{1}$$

If the spread deviates significantly from its mean μ_z, we construct a position: Long Y_t and short βX_t when $z_t < \mu_z - k\sigma_z$, Short Y_t and long βX_t when $z_t > \mu_z + k\sigma_z$,

where σ_z is the standard deviation of z_t and k is a threshold parameter.

Profit from a trade cycle (spread reverting from z_t to μ_z) is approximately:

$$\Pi \approx (z_t - \mu_z) \cdot Q$$

Risks include breakdown of cointegration (spreads that diverge indefinitely), volatility shocks, and transaction costs from frequent rebalancing.

3.5 Comparison of Arbitrage Strategies

Table 1: Comparison of Arbitrage Strategies

Strategy	**Returns**	**Risks**	**Potential Use of AI**	**Platform Requirements**
Cross-Exchange	Moderate returns depending on market inefficiencies and available spreads. Profits can be frequent but usually small per trade.	Risks include transfer delays, withdrawal fees, exchange downtime, and price slippage during transfers.	AI can optimize trade execution, identify arbitrage windows in real-time, and reduce latency through predictive modeling.	Requires accounts on multiple centralized or decentralized exchanges with automated trade execution infrastructure (bots or scripts).
Cash-and-Carry	Low to moderate but stable returns, especially during periods of high futures premiums (contango). Returns are fixed after position opening.	Risks include futures premium compression, opportunity cost of locked capital, and liquidity risk in futures markets.	AI can assist in market monitoring, automate entry/exit timing, and optimize capital allocation based on predictive analytics.	Requires access to both spot and futures markets; integration with derivatives exchanges and possibly collateral management systems.
Funding Rate	Moderate returns, especially in bullish markets with high funding rates. Returns accumulate passively while positions are open.	Risks include rapid market movements, funding rate reversals, and imperfect hedging leading to net losses.	AI can predict funding rate trends, identify optimal times to enter trades, and dynamically rebalance positions.	Requires access to perpetual futures markets and the ability to maintain market-neutral positions through spot and derivatives trading.
Statistical	Moderate to high returns depending on market conditions and correlation stability. Returns can be significant in volatile markets.	Risks include breakdown of historical correlations, execution slippage, and model overfitting leading to false signals.	AI plays a central role in statistical arbitrage through machine learning models for correlation detection, anomaly spotting, and trade signal generation.	Requires historical data access, statistical modeling capabilities, and automated trading infrastructure; typically computationally intensive.

After assessing the four strategies, Statistical Arbitrage is selected as the primary focus for this project. While strategies like Cross-Exchange or Cash-and-Carry offer more deterministic returns, they often face "limits to arbitrage" such as high execution costs, transfer latency, or capital inefficiency. In contrast, Statistical Arbitrage provides a high-dimensional, non-linear environment that is ideal for Reinforcement Learning. An RL agent can move beyond static cointegration models by learning to navigate complex market regimes, optimize entries and exits based on shifting correlations, and dynamically manage risk in the presence of transaction costs. This approach leverages the core strengths of AI—pattern recognition and sequential decision-making—to capture alpha that traditional rule-based systems might miss in the volatile DeFi perpetual markets.

4 Methodology

This section presents an overview of the proposed AI agent and the structure of its operational processes. The system is designed as a modular pipeline that separates *model training*, *live trade execution*, and *periodic fine-tuning*, while maintaining a shared representation of market structure and learned trading behavior.

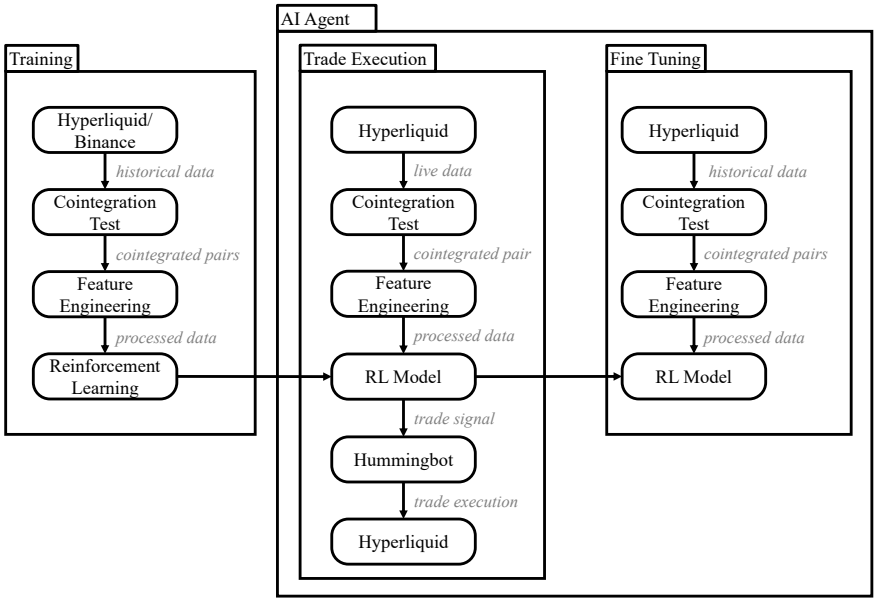

Fig. 1. High-level architecture of the AI agent and RL model.

Figure 1 provides a high-level overview of the agent architecture. The design follows a clear lifecycle: historical data are used to identify tradable relationships

and train reinforcement learning policies, which are then deployed in a live execution environment. As market conditions evolve, the agent can be periodically retrained or fine-tuned using updated historical data, closing the feedback loop between learning and execution.

The **training stage** operates entirely offline and relies on historical data obtained from centralized and decentralized exchanges. In this stage, candidate asset pairs are identified through cointegration tests, after which market and relational features are constructed. These processed features form the state space of the environment in which a reinforcement learning model is trained. The output of this stage is a set of learned policy parameters that encode the agent's trading behavior under the assumed market frictions.

The **trade execution stage** represents the online deployment of the trained agent. Live market data are streamed from the exchange and processed using the same cointegration and feature engineering pipeline as in training to ensure consistency between offline and online representations. The reinforcement learning model generates continuous trading signals, which are translated into executable orders by an external execution engine. In this work, order placement and exchange interaction are handled by Hummingbot, allowing the agent to operate in real time while abstracting away low-level exchange-specific details.

The **fine-tuning stage** provides a mechanism for adapting the agent to changing market dynamics. Rather than continuously updating the policy online, which may introduce instability, the system periodically retrains or fine-tunes the reinforcement learning model using recent historical data. This process re-estimates cointegration relationships, updates feature distributions, and refines the learned policy before redeployment. The separation between execution and fine-tuning ensures stability during live trading while allowing the agent to evolve over time.

Overall, this architecture emphasizes modularity, reproducibility, and economic realism. Each component—data collection, statistical modeling, reinforcement learning, execution, and monitoring—can be independently modified or extended, enabling systematic experimentation while preserving a coherent end-to-end trading workflow.

In the following subsections, each component of the architecture is described in detail.

4.1 Data Sources and Trading Venue Positioning

Our research framework maintains a clear division between **offline training/backtesting** and **online execution/validation**. **Hyperliquid** was selected as the primary execution venue (see Table 2), primarily due to its on-chain Central Limit Order Book (CLOB), which ensures transparency in market data and order execution within a decentralized perpetual environment. For the offline phase—including agent training and performance evaluation—we utilized historical data from **Binance USDⓈ-M Perpetual Futures**. Binance was chosen for its extensive historical coverage and stable API, enabling the construction of a robust and reproducible experimental pipeline.

While we initially processed approximately 58.2GB of raw Hyperliquid archives from AWS S3, the available granular data only dates back to March 22, 2025. This span is insufficient for the one-year "train-validate-test" split required for this study. Consequently, we utilized Binance as the primary training source while maintaining consistent data field definitions and alignment policies via the **CCXT** framework to bridge both venues. Details regarding the AWS S3 data pipeline and its implementation are provided in Appendix A.2.

Table 2. Comparative analysis of selected platforms.

Platform	**Role**	**Matching System**	**Fee Model**	**Funding Mechanism**
Hyperliquid	Execution	On-chain CLOB (L1)	Tiered by volume	Protocol-defined
Binance	Training Data	Centralized CLOB	VIP Tiers	Perpetual funding
dYdX v4	Alternative	Off-chain matching	Protocol fee schedule	Hourly (Governance)
Paradex	Alternative	Off-chain sequencer (L3)	Protocol fee schedule	Premium-based model

4.2 Data Acquisition and Time Coverage

Offline data was retrieved from Binance Futures using Python and the **CCXT** library. We extracted 1-minute OHLCV sequences and corresponding funding rates for all candidate assets. The dataset spans from **May 2024 to April 2025**. To mitigate look-ahead bias and ensure robust out-of-sample evaluation, the data was partitioned into non-overlapping training, validation, and testing sets chronologically.

To ensure that policies learned offline are executable on Hyperliquid, we restricted our universe to the intersection of assets listed on both platforms. Starting from the top-50 tokens by market capitalization, we filtered for liquidity and tradeability, resulting in a final pool of **26 assets**. This selection provides a sufficient candidate space for the subsequent rolling cointegration and pair-trading phases.

4.3 Data Preparation and State Alignment

We adopted a "minimal necessary processing" strategy to ensure temporal consistency and feature reproducibility. All OHLCV data were aligned to a standardized **1-minute grid**. Sparse missing intervals were handled via forward-filling (`ffill`) or removal to maintain the state continuity required by the Markov Decision Process (MDP).

Since funding rates are updated at lower frequencies, they were aligned with the 1-minute price series using a "last available value" (backward merge) policy. This ensures that at any minute t, the agent observes the most recent funding information available, reflecting the actual settlement logic of perpetual contracts. The final state vector incorporates normalized spread signals, account status

(position exposure and cash), and technical indicators such as RSI, MACD, and Kalman-filtered trend deviations. Finally, these aligned log-price series serve as the input for the rolling window cointegration and ADF stationarity tests described in Section 4.4.

4.4 Cointegration

The statistical arbitrage strategy is based on the Engle–Granger cointegration framework [11,21]. Let X_t and Y_t denote the log-prices of two assets. The long-run equilibrium relationship is modeled as

$$Y_t = \alpha + \beta X_t + \epsilon_t, \tag{2}$$

where ϵ_t is the spread capturing short-term deviations from equilibrium.

This method is usually done in two steps: (1) Estimate joint dynamic of assets via simple OLS to get an estimate of hedge ratio (β) (2) Use residuals of this regression model as estimates of the theoretical noise (ϵ_t), which also represents the spreads, and test them for stationarity via Augmented Dickey-Fuller (ADF) test. If the p-value for ADF test is smaller than 0.05, we consider these pair of assets (A,B) as cointegrated.

The main challenge is that it is not sufficient to perform cointegration modelling once based on the full length of our historical data and then trade these pairs. There are several reasons why it would not work well. Firstly, cryptomarkets are known for their volatility and quick pace of change, therefore the joint dynamic of assets may vary significantly over time. Secondly, since our data has 1-min frequency, some long-term joint trends may not persist on these short-term timeframes. Thus, we perform cointegration tests on a rolling-window basis with additional filtering by correlation. For a day T, data from three previous days T-1, T-2, T-3 (4320 observations in total) is used to estimate OLS equations for cointegration tests (Figure 2). From all the cointegrated pairs for this window we choose five most correlated pairs. Afterwards, only these top-5 pairs (it could be less than five if there was less than five cointegrated pairs) are considered for the trading period. The trading period is one day (1440 observations). Thus, our approach is based on three days rolling-window cointegration with $2/3$ overlap and additional correlation filtering.

It is computationally infeasible to apply this procedure to all the possible asset pairs which are present on Hyperliquid. For this reason, we also applied pre-selection of pairs to narrow our scope. We consider only those assets which are traded on both chosen exchanges - Binance and Hyperliquid - are in top-50 by market cap and their correlation on our historical Binance data is at least 0.85. Our final sample consists of 25 different assets.

4.5 Feature Engineering

In this project we delegate decisions about entering and exiting trades to a reinforcement learning model. Only spreads themselves may not provide enough

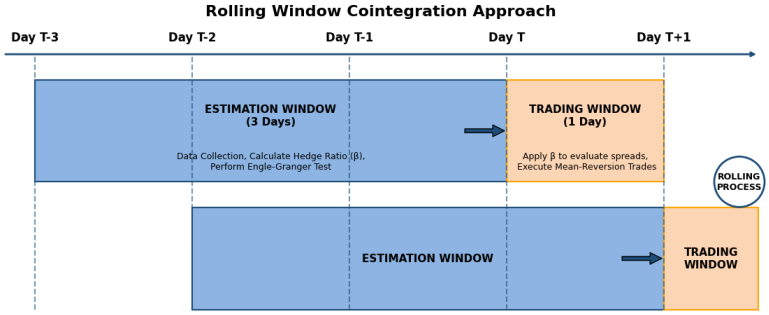

Fig. 2. Rolling Window Cointegration Strategy: 3-day Estimation Window (Selection) followed by a 1-day Trading Window.

information for a model to make a good decision. In order to fill this gap and extend state space of a model, we create some additional features. Overall, we split the data into the following three time intervals:

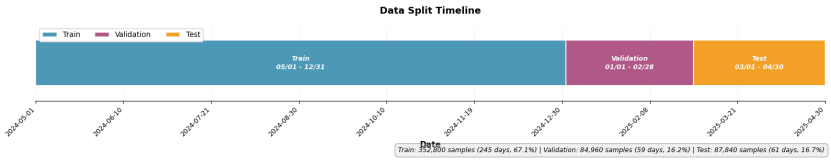

Fig. 3. Data Split for Training, Validation and Testing.

4.5.1 Spread-Based Features The primary signal for the agent is the normalized spread (z-score), which transforms the price discrepancy into a stationary signal. It is calculated by standardizing the current spread s_t using the mean μ_{window} and standard deviation σ_{window} derived from the 3-day estimation window:

$$z_t = \frac{(p_t^A - (\alpha + \beta p_t^B)) - \mu_{window}}{\sigma_{window}} \tag{3}$$

To get better understanding, we have plotted spreads dynamics for one of the trading days (Figure 4).

To further characterize the spread's behavior, we include the following features:

– **Spread Volatility (EWMA)**: Provides a measure of signal stability and risk by computing the exponentially weighted moving standard deviation of

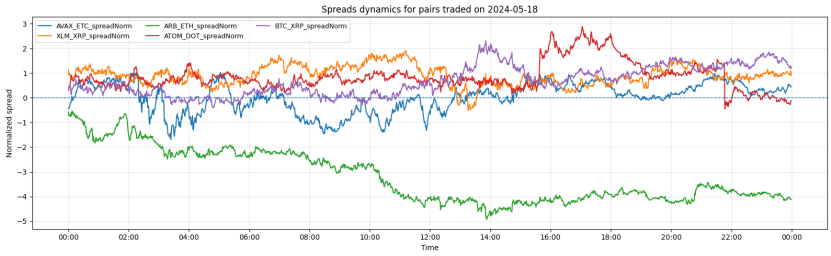

Fig. 4. Normalized spreads dynamics for 2024-05-18.

the spread:

$$\sigma_{s,t} = \sqrt{(1-\lambda)\sum_{i=0}^{\infty}\lambda^i(s_{t-i}-\bar{s})^2}, \quad \lambda = 0.94 \tag{4}$$

- **Spread Kalman MA**: Acts as a dynamic mean-reversion target for the spread, calculated as a 20-period rolling average to filter out high-frequency noise.
- **Spread Moving Average**: A standard 20-period simple moving average (SMA) that assists the agent in identifying the local trend of the normalized spread.
- **Relational Statistics**: The hedge ratio (β), intercept (α), Pearson correlation (ρ), and the Augmented Dickey-Fuller p-value (p-val) are propagated as static features for each window. These allow the agent to assess the current strength and reliability of the cointegration relationship.

4.5.2 Technical and Momentum Indicators For each individual asset in the pair, we compute a suite of features to capture momentum and trend dynamics:

- **Relative Strength Index (RSI)**: A 20-period momentum oscillator used to identify overbought or oversold conditions in the underlying assets[cite: 1]:

$$RSI_t = 100 - \left[\frac{100}{1 + \frac{AvgGain}{AvgLoss}}\right] \tag{5}$$

- **Moving Average Convergence Divergence (MACD)**: Captures changes in the strength, direction, and momentum of the price trend by calculating the difference between a 5-period fast and 14-period slow EMA:

$$MACD_t = EMA_5(p_t) - EMA_{14}(p_t) \tag{6}$$

- **Stochastic RSI**: Increases the sensitivity of the RSI to local price extremes, allowing the agent to pinpoint potential turning points:

$$StochRSI_t = \frac{RSI_t - \min(RSI_{t-20:t})}{\max(RSI_{t-20:t}) - \min(RSI_{t-20:t})} \tag{7}$$

- **Kalman Filter Proxy**: A 20-period rolling mean used as a dynamic price baseline (K_t) to provide a smoother representation of the asset's value.
- **Price Bias**: Measures the local deviation from the filtered price trend, calculated as $Bias_t = p_t - K_t$.
- **Directional Signs**: The ratio of positive to negative returns over a 10-minute rolling window, used to estimate short-term market sentiment.

4.5.3 Funding Rates To account for the specific frictions of the perpetual futures market, we integrate funding features:

- **Funding Rate (FR_t)**: The periodic payment exchanged between positions, representing the direct cost or profit of maintaining a delta-neutral hedge.
- **Funding Countdown (T_{left})**: The minutes remaining until the next funding settlement. The countdown automatically adjusts to asset-specific intervals (e.g., 4h for TON/ENA and 8h for others on Binance, or 1h on Hyperliquid).
- **Price Shadows**: Proxies for intraday price rejection and volatility. Upper shadow is defined as $p_t - \min(p_t, p_{t-1})$, and lower shadow is defined as $\max(p_t, p_{t-1}) - p_t$.

The final state vector S_t is a concatenation of asset-specific technicals, pairwise relational statistics, funding dynamics, and the agent's current portfolio composition (weights w_1, w_2 and cash balance).

4.6 Reinforcement Learning Model

4.6.1 Action Space The trading agent operates with a single continuous action variable

$$a_t \in [-1, 1], \tag{8}$$

where a_t represents the portfolio's exposure to the spread 1 between the two assets:

- $a_t < 0$: long asset 1 and short asset 2 (positive spread exposure),
- $a_t = 0$: no position (full cash),
- $a_t > 0$: short asset 1 and long asset 2 (negative spread exposure).

This formulation ensures smooth control over the portfolio's net exposure.

4.6.2 State Space The state space S_t at time t consists of three components: (i) single-asset features, (ii) pairwise relational features, and (iii) the current portfolio composition. Together, these elements describe both market conditions and the agent's internal position, forming a complete Markov state representation.

$$S_t = \big(S_t^{(\text{asset})},\ S_t^{(\text{pair})},\ S_t^{(\text{portfolio})}\big). \tag{9}$$

Single-Asset Features These features summarize price dynamics, technical structure, volatility, and funding conditions for each individual asset. Examples include:

- price-based signals: close, close bias, Kalman-filtered price,
- volatility measures: EWMA volatility, upper and lower shadow lengths,
- momentum indicators: MACD, RSI, StochRSI, directional signs,
- funding information: funding rate and time to next funding event.

Pairwise Relationship Features These features describe the statistical interaction between the two assets and their deviation from long-run equilibrium. Typical elements include:

- cointegration statistics: α, β, correlation, p-value,
- normalized spread 1 signals: normalized spread, Kalman-normalized spread, moving-average normalized spread, volatility-normalized spread.

These variables allow the agent to recognize mean-reversion opportunities and assess the strength of the pair relationship at each timestep.

Portfolio Composition The agent's current allocation is represented by

$$S_t^{(\text{portfolio})} = (w_t^{(1)}, w_t^{(2)}, w_t^{(c)}), \tag{10}$$

where $w_t^{(1)}$, $w_t^{(2)}$, and $w_t^{(c)}$ denote the portfolio weights in asset 1, asset 2, and cash, respectively.

Including portfolio composition is essential in reinforcement learning: the action selected at time t directly modifies the next state S_{t+1} through the updated weights, establishing the Markov property that is necessary for stable RL training. This feedback loop ensures that the agent learns not only how market signals evolve, but also how its own trading decisions affect future reward opportunities and transaction costs.

At each training step, a random eligible asset pair may be sampled. This exposure to heterogeneous pair dynamics improves generalization and reduces overfitting to any specific pair or market condition.

4.6.3 Reward Function The reward function governs the agent's risk-taking behavior and economic interpretation of its decisions. We therefore adopt a quadratic utility formulation, which provides an economically grounded trade-off between expected returns and risk, closely related to mean–variance optimization.

Let the net portfolio return after transaction costs be defined as:

$$x_t = r_t^p - c_t, \tag{11}$$

where r_t^p is the instantaneous portfolio return and c_t represents transaction costs.

The instantaneous utility is:

$$U(x_t) = x_t - \frac{\lambda}{2} x_t^2. \tag{12}$$

Thus, the reward function is given by:

$$R_t = (r_t^p - c_t) - \frac{\lambda}{2}(r_t^p - c_t)^2, \tag{13}$$

where λ is the risk-aversion coefficient that scales the quadratic penalty to balance the trade-off between expected returns and volatility.

This formulation balances profitability with risk control: the linear term rewards positive returns, while the quadratic penalty discourages large fluctuations, effectively imposing risk aversion. Dividing λ by 2 is a mathematical convention that simplifies the first derivative (marginal utility) and aligns the function with classical Mean-Variance Optimization frameworks.

4.6.4 Model Configuration The model operates on high-frequency cryptocurrency market **data sampled** at a **one-minute** resolution. Raw data are preprocessed using forward-filling to handle missing observations, followed by the removal of any remaining missing values.

The **dataset** spans a one-year period from **May 2024 to April 2025**, providing sufficient market variation for training, validation, and testing.

The trading environment is formulated as a sequential decision-making problem with a cash position included in the state. Actions are modeled in continuous space, allowing the agent to take fractional long or short positions between -1 and 1. A discrete action space with fixed position increments is also supported for alternative experiments.

The environment operates on a **daily trading window** with a sliding step of one day. Each decision is conditioned on a rolling **lookback window of 20 observations**, providing short-term historical context.

Data are divided into non-overlapping training, validation, and test sets using a chronological split to avoid look-ahead bias. The training period spans eight months, followed by a two-month validation window for model selection and hyperparameter monitoring. The final two months are reserved exclusively for out-of-sample testing. An alternative split configuration is used for early experiments to stress-test model generalization under shorter horizons.

The primary algorithm used in this study is **Proximal Policy Optimization (PPO)**, an on-policy reinforcement learning method known for stable training in continuous action spaces. The agent employs a **multilayer perceptron policy network (MlpPolicy)** to map observed states to trading actions.

Training is performed over four million timesteps using a discount factor of $\gamma = 0.99$. Generalized Advantage Estimation (GAE) is applied with $\lambda = 0.95$ to reduce variance in policy gradient updates.

Model performance is evaluated periodically during training using out-of-sample episodes. Evaluation frequency is aligned with trading horizons to ensure

economically meaningful comparisons. Performance metrics and diagnostic plots are generated automatically and stored in a structured reports directory.

5 Hummingbot & AI Agent Implementation

5.1 Hummingbot

The execution layer is built upon Hummingbot [18], an open-source client primarily used for automated market making in cryptocurrency markets. It allows users to deploy high-frequency trading logic on both centralized and decentralized venues by exchange interactions through a unified connector framework. The system relies on a local, event-driven architecture to manage inventory and order lifecycles, enabling the execution of investment strategies.

In this project, Hummingbot is used as the execution gateway. Its utilization provides several advantages over custom API implementations, including:

- Hummingbot maintains a local, continuously updated view of order books via WebSocket market data feeds and tracks balances/order events via exchange user streams when available (with REST-based refresh or fallback where needed). This reduces reliance on per-trade REST polling for inventory checks and lowers decision latency.
- Order Lifecycle Tracking: Hummingbot creates internal objects for every submitted order, tracking their state transitions (created $\rightarrow$ submitted $\rightarrow$ filled or failed). This robustness prevents state desynchronization, such as zombie orders, where the system assumes an order failed when it actually filled.
- Exchange Abstraction: The strategy logic is written against a generalized interface. This allows the underlying exchange connector (in our case, Hyperliquid) to be swapped without changing the execution logic.

In the following, the execution workflow is explained.

5.1.1 Data Acquisition and Alignment The execution cycle begins with data ingestion. At every decision interval, the system queries the Hyperliquid API via the `candleSnapshot` endpoint to fetch OHLCV (Open, High, Low, Close, Volume) data for the active asset pair. A critical aspect in this initial step is temporal alignment. The system ensures that the retrieved time-series data for both assets (e.g., ETH and BTC) are synchronized to the same Unix timestamps. Incomplete or mismatched frames caused e.g. by API latency are detected and handled via a retry mechanism to prevent the propagation of corrupted states to the inference engine.

5.1.2 Real Time Feature Engineering Once valid data is retrieved, the system must reconstruct the state space $\mathcal{S}$.

The transformation process involves three distinct steps:

1. **Cointegration Parameter Application:** The system utilizes the cointegration parameters (α and β) determined during the pre-market selection phase. These parameters are applied to the live price streams to calculate the raw spread:
$$s_t = P_{A,t} - (\alpha + \beta P_{B,t})$$
2. **Kalman Filtering:** A linear Gaussian state-space model is instantiated and applied to the price series and the calculated spread. This recursively estimates the true latent state of the asset prices, filtering out high-frequency market microstructure noise.
3. **Z-Score Normalization:** The inputs are normalized to a standard normal distribution to ensure the neural network receives gradients within a stable range. The spread is normalized using a rolling window $w = 30$:
$$Z_t = \frac{Spread_t - \mu_{t-w:t}}{\sigma_{t-w:t}}$$

5.1.3 Tensor Construction and Inference The final step in the logic layer is the construction of the observation tensor. The system concatenates the processed features into a single, flattened vector of dimension 963. The vector structure is ordered to match the input layer of the policy network:

1. Pair Features (240 dimensions): 8 statistical indicators (including Spread Z-Score, Correlation, Volatility) over a 30-step lookback.
2. Asset A Features (360 dimensions): 12 technical indicators (including RSI, MACD, Normalized Close) over a 30-step lookback.
3. Asset B Features (360 dimensions): Identical structure to Asset A.
4. Portfolio Allocation State (3 dimensions): The current weights for Asset A, Asset B, and Cash.

This tensor is passed to the pre-trained RL agent. The agent outputs a continuous action $a_t \in [-1, 1]$, which dictates the target portfolio allocation at time t. This decision is serialized into a JSON object (`signal.json`) containing the timestamp, asset identifiers, and target weights, serving as the asynchronous message to the execution layer. An example JSON object is provided in Appendix A.4.

5.1.4 The Execution Bridge The interface between the asynchronous signal generator and the synchronous execution engine is handled by a custom script running within Hummingbot's event loop. The script utilizes the framework's `on_tick()` method to poll the `signal.json` file at 1-second intervals. To ensure execution integrity, the script implements validation logic.

The execution bridge is responsible for aligning the current inventory with the target weights calculated in the previous step with and saved as the aforementioned JSON-object. For each asset i in the active pair, the script first queries the exchange connector for the current position size $P_{current}$. It then calculates the

target position size based on the total account equity E and the RL-generated weight W_i:

$$P_{target} = \frac{E \times W_i}{Price_i} \tag{14}$$

The entire execution system is containerized using Docker to ensure environment consistency and portability across deployment infrastructure. The Hummingbot instance runs within a dedicated Docker container, which manages the complex dependencies required for secure exchange connectivity and WebSocket handling. To facilitate the asynchronous communication between the layers, a shared volume is mounted between the host system and the Docker container (a diagram of the signal-to-execution workflow can be seen in the Appendix A.5).

The external instance generator process writes the signal.json file to this shared directory on the host, while the Hummingbot container reads from it in real-time. This configuration isolates the machine learning dependencies in the host environment from the lightweight, latency-sensitive execution environment inside the container, preventing resource contention.

5.2 AI Agent

This section describes the integration and orchestration of previously developed trading system components into a unified agent architecture using LangGraph and LangSmith. The statistical arbitrage methodology, reinforcement learning formulation, and execution infrastructure described throughout Sections 3, 4, and 5 are not reintroduced here. Instead, this chapter focuses on how these components are coordinated, traced, and evaluated within a structured agent workflow.

LangGraph is employed as a workflow orchestration layer, while LangSmith provides execution tracing and observability. Neither framework participates in latency-critical execution or market interaction. All trading logic and execution remain external to these tools.

5.2.1 Architectural Overview The implemented agent architecture consists of two interlaced but decoupled workflows operating at different temporal resolutions:

- A *slow-tick workflow* responsible for structural decisions, specifically the selection of tradable asset pairs.
- A *fast-tick workflow* responsible for inference and signal generation on the currently active pairs.

The workflows communicate exclusively through explicit artifacts persisted to disk. No implicit state is retained in memory or within the market. This design ensures that all system state is observable, reproducible, and externally inspectable. LangGraph formalizes both workflows as directed acyclic graphs (DAGs), while LangSmith records execution traces and metadata for post-hoc analysis.

In addition to the slow-tick and fast-tick execution nodes, the agent architecture includes a small set of explicit control-plane nodes that operate on realized performance rather than market inputs. These nodes are implemented as LangGraph components and are executed independently of the per-tick trading loop.

An overview of the implemented agent architecture, including the performance monitoring and governance components, is shown in Figure 5.

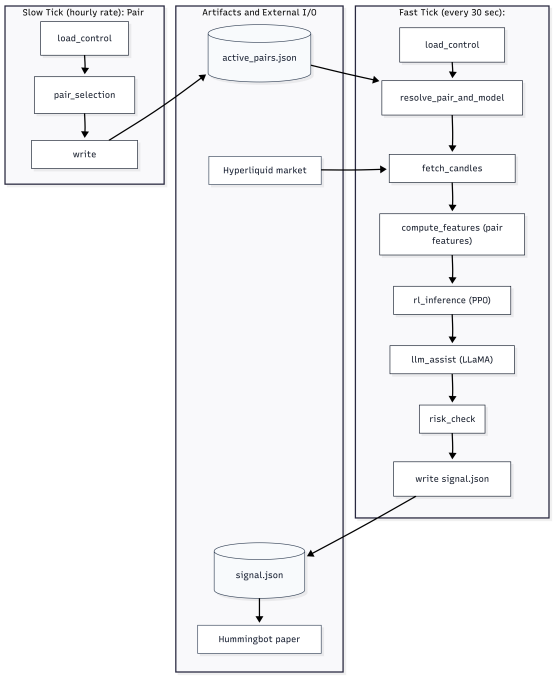

Fig. 5. Agent architecture implemented using LangGraph and LangSmith. The trading engine encapsulates both fast-tick inference and slow-tick pair selection, while a separate performance evaluation loop assesses realized outcomes and triggers structural actions such as pair reselection, model retraining requests, or system pausing. LangSmith provides execution tracing and observability but does not participate in live trading decisions.

5.2.2 Slow Tick Workflow: Pair Selection The slow-tick workflow executes on a coarse cadence (e.g., hourly) and determines the active trading universe.

The statistical logic underlying pair selection is implemented as described in Section 5, which builds upon the cointegration framework detailed in Sections 3 and 4. This includes correlation filtering and Engle–Granger cointegration testing [11,21].

Within the agent architecture, this logic is encapsulated as a LangGraph workflow consisting of:

- **Load Control:** Loads configuration parameters governing the asset universe, selection cadence, and ranking thresholds.
- **Pair Selection:** Executes the statistical ranking and selection procedure to identify the most suitable tradable pairs.
- **Artifact Write:** Persists the selected pairs and associated metadata to `active_pairs.json`.

The resulting artifact serves as the authoritative trading universe for the fast-tick workflow. Structural market assumptions are thus updated infrequently and remain isolated from per-tick inference logic.

5.2.3 Artifact Layer and External I/O System coordination relies on a minimal set of explicit artifacts:

- `active_pairs.json`, defining the current trading universe.
- `signal.json`, containing the most recent execution signals.

In addition, realized trading performance is summarized externally into a lightweight artifact (`perf_snapshot.json`), which captures aggregated metrics such as profit and loss, win rate, drawdown, and execution quality over a rolling window. This artifact is produced by the execution layer and consumed exclusively by the performance evaluation workflow.

Market data are retrieved on demand from the exchange via the data acquisition pipeline described in Section 4. No persistent internal market state is maintained beyond these artifacts.

5.2.4 Fast Tick Workflow: Inference and Signal Generation The fast-tick workflow executes at a fixed short interval (e.g., every 30 seconds) and generates trading signals for the active pairs.

The reinforcement learning formulation, feature engineering pipeline, and PPO policy architecture are described in detail in Section 4. Within the agent architecture, these components are orchestrated as a LangGraph workflow with the following responsibilities:

- **Load Control:** Loads runtime configuration and operator-defined overrides.
- **Resolve Active Pairs:** Reads the current trading universe from `active_pairs.json`.
- **Fetch Market Data:** Retrieves recent candle data from the exchange.
- **Feature Computation:** Constructs model features consistent with the training environment.
- **RL Inference:** Performs inference using a pretrained PPO policy in inference-only mode.
- **LLM Advisory Layer:** Generates auxiliary annotations without modifying the RL decision.
- **Risk Checks:** Applies hard constraints to ensure signal validity.

– **Signal Emission:** Writes validated signals to `signal.json`.

The reinforcement learning policy remains the sole source of trading decisions. The optional LLaMA-based component is included strictly as an interpretive and diagnostic aid and does not introduce additional decision authority.

5.2.5 Performance Evaluation: Strategic Decision Making The performance evaluation workflow consists of the following nodes:

– **Metrics Rollup:** Reads aggregated execution metrics from `perf_snapshot.json` and constructs a compact performance summary over a rolling window.
– **Watcher Evaluation:** Uses a local LLaMA-family model to assess trading health and recommend whether trading should continue unchanged or whether a structural action should be considered.
– **Governor:** Applies timing constraints, confidence thresholds, and hysteresis to the watcher recommendation to prevent oscillatory or premature structural changes.
– **Apply Action:** Executes the authorized structural action by triggering pair reselection, emitting a retraining request artifact, or pausing the trading system.

These control-plane nodes do not modify per-tick trading decisions and do not override reinforcement learning outputs. Instead, they operate at a supervisory level, initiating structural changes only when sustained performance degradation or regime shifts are detected, as seen in Figure 6..

5.2.6 Execution Layer Order execution is handled externally by Hummingbot, as described in Section 5. Hummingbot monitors `signal.json` and translates allocation signals into executable orders, managing exchange connectivity, order placement, and execution-level risk controls.

The agent itself does not place orders and does not maintain execution state, ensuring a strict separation between decision-making and execution.

5.2.7 Observability and Tracing LangSmith is used as an observability layer to record structured execution traces for both workflows. Inputs, outputs, and intermediate artifacts are logged consistently, enabling detailed inspection of agent behavior across runs.

In addition to tracing the execution workflows, LangSmith records the outputs of the performance evaluation and governance components. This allows post-hoc analysis of when and why structural actions—such as pair reselection or retraining requests—were initiated.

At the current stage, LangSmith is used exclusively for tracing and evaluation. Automated server-side control rules and webhook-triggered actions are intentionally excluded from the live trading loop to preserve determinism and auditability.

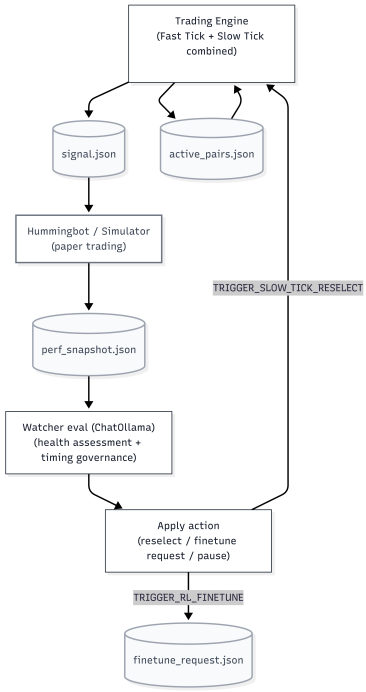

Fig. 6. Closed-loop trading agent with artifact-based governance. Trading decisions are generated by a unified engine, while an external performance watcher evaluates outcomes and initiates structural adjustments via explicit artifacts.

5.2.8 Implementation Details All workflows and experiments described in this chapter were executed using fixed, versioned software dependencies to ensure reproducibility. This subsection reports the versions of tooling relevant to the orchestration, observability, and integration layers implemented as part of this work.

LangGraph was used to formalize the slow-tick, fast-tick, and performance evaluation workflows as directed acyclic graphs, while LangSmith provided execution tracing and observability. At the time of implementation, the following versions were used:

- LangChain: v1.1.0
- LangGraph: v1.0.3
- LangSmith: v0.6.3

The optional LLM-assisted components, including both advisory annotations and performance evaluation, were implemented using the Ollama runtime (v0.6.1) with a local LLaMA-family model. Integration with the LLM was handled via `langchain-ollama` (v1.0.1).

The orchestration layer and supporting infrastructure were implemented in Python v3.x, with asynchronous execution supported by `aiohttp` (v3.13.2) and scheduling handled via `APScheduler` (v3.11.0). Data handling and analysis relied on NumPy (v2.2.6), Pandas (v2.3.2), and SciPy (v1.15.3).

Order execution was handled externally by Hummingbot, as described in Section 5. Reinforcement learning model architecture, training procedures, and associated dependencies are detailed separately in Section 4 and are not part of the implementation contribution described in this chapter.

All configuration files, workflow artifacts, performance summaries, and LangSmith execution traces were versioned and retained to ensure consistency and enable post-hoc analysis.

6 Results

6.1 Reinforcement Learning Experiments

This section presents the empirical results of the reinforcement learning (RL) experiments conducted under different transaction fee assumptions. The objective is to evaluate how varying fee regimes affect the learned trading policy and its performance. All experiments use identical environments, state representations, reward definitions, and training horizons, differing only in the transaction fee structure applied.

Experimental Setup The RL agent is trained to trade perpetual futures contracts using market orders. As a result, only *taker fees* are considered throughout the experiments. Maker fees are excluded, as the agent does not place limit orders and therefore does not provide liquidity to the order book. This assumption

reflects a realistic execution setting for latency-sensitive strategies that require immediate position adjustments.

Transaction fees are modeled based on the official Hyperliquid perpetual futures fee schedule, which depends on (i) the trader's 14-day rolling weighted trading volume (volume tier) and (ii) the account category determined by HYPE staking (staking tier) [19].

Three fee regimes are considered:

- **Experiment 1 (No Fees):** A frictionless benchmark scenario with zero transaction costs (0 bps).
- **Experiment 2 (Lowest Fees):** Diamond account with Tier 6 volume level, corresponding to a taker fee of 1.44 basis points.
- **Experiment 3 (Highest Fees):** Base rate account with Tier 0 volume level, corresponding to a taker fee of 4.50 basis points.

All backtests are conducted on a held-out test set consisting of a concatenation of multiple asset pairs. The evaluation period spans approximately 270 trading days. Portfolio values and actions are updated at a one-minute sampling frequency. This setup allows the agent to be evaluated across heterogeneous market conditions and asset dynamics within a single continuous backtesting framework.

Hyperliquid Fee Structure Hyperliquid's trading fees are determined by a combination of a volume-based tier system and a staking-based discount schedule (see Table 3). Since the RL agent exclusively submits market orders, only taker fees are relevant for the experiments.

Table 3. Hyperliquid perpetual futures taker fee schedule (bps) by 14-day weighted trading volume and staking tier, including staking requirements and discounts [19].

Tier	**Base**	**Wood**	**Bronze**	**Silver**	**Gold**	**Platinum**	**Diamond**
HYPE staked	–	>10	>100	>1k	>10k	>100k	>500k
Fee discount	–	5%	10%	15%	20%	30%	40%
14d volume ($)							
–	**4.50**	4.28	4.05	3.83	3.60	3.15	2.70
>5M	4.00	3.80	3.60	3.40	3.20	2.80	2.40
>25M	3.50	3.33	3.15	2.98	2.80	2.45	2.10
>100M	3.00	2.85	2.70	2.55	2.40	2.10	1.80
>500M	2.80	2.66	2.52	2.38	2.24	1.96	1.68
>2B	2.60	2.47	2.34	2.21	2.08	1.82	1.56
>7B	2.40	2.28	2.16	2.04	1.92	1.68	**1.44**

Staking tiers (HYPE staked). Hyperliquid applies trading fee discounts based on the amount of HYPE tokens staked by the trader. These staking tiers determine the account category and apply a proportional discount to trading fees.

Volume tiers (14-day weighted volume). Hyperliquid applies perpetual futures taker fee schedule (in basis points) by volume tier and staking-based account category.

The RL agent executes trades exclusively via market orders in order to ensure immediate execution and avoid queue priority effects. Consequently, only taker fees are considered in all experiments, while maker fees are excluded. The fee levels used in the experiments correspond to the Diamond Tier 6 taker fee (1.44 bps) for the low-fee scenario and the Base rate Tier 0 taker fee (4.50 bps) for the high-fee scenario. Staking tiers are treated as exogenously given to enable a clean fee sensitivity comparison.

Performance The performance of the reinforcement learning agent is evaluated out-of-sample under different transaction fee regimes. For all experiments, the portfolio value is normalized to 1.0 at the beginning of the evaluation period, including the equal-weight baseline. The out-of-sample evaluation consists of a sequence of concatenated one-day trading windows, each corresponding to an asset pair identified as cointegrated, covering the period from 1st of March 2025 to 30th of April 2025.

Figure 7 shows the evolution of portfolio value for the reinforcement learning agent across fee regimes, together with the equal-weight benchmark.

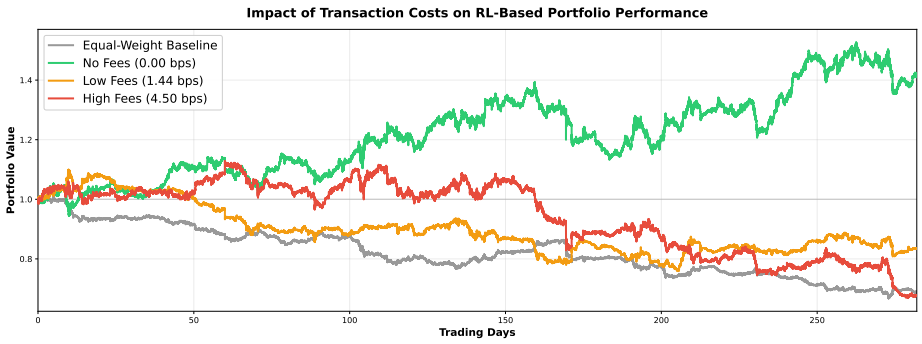

Fig. 7. Out-of-sample portfolio performance of the reinforcement learning agent under different transaction fee regimes. All portfolios are normalized to an initial value of 1.0.

Table 4 reports summary statistics for each configuration. In addition to total return and realized volatility, the table includes the mean and standard deviation of the agent's actions. The action variable lies in the interval $[-1, 1]$ and represents the signed exposure to the traded spread.

Table 4. Performance summary across transaction fee regimes.

Fee Regime	Total Return (%)	Volatility (%)	Action Mean	Action Std
No Fees	42.20	36.54	-0.376	0.926
Low Fees	-16.49	23.92	-0.277	0.835
High Fees	-32.37	34.37	-0.435	0.605
Equal-Weight Baseline	-31.29	18.29	-	-

In the no-fee setting, the reinforcement learning agent attains a final portfolio value of 1.4220. Under the low-fee regime, the final portfolio value is 0.8351, while under the high-fee regime it is 0.6763. The equal-weight baseline ends the evaluation period with a portfolio value of 0.6871.

Across fee regimes, the distribution of actions differs in terms of mean exposure and variability, as reflected in the reported action statistics.

6.2 AI-Agent Proof of Concept

The implemented system constitutes a proof of concept demonstrating that LangGraph and LangSmith can be effectively repurposed as coordination and observability layers for a quantitative trading agent. The primary results of this work are architectural and operational in nature, rather than economic.

The agent architecture successfully integrates independently developed components—including statistical pair selection, reinforcement learning inference, and external order execution—into a unified, reproducible workflow. Both slow-tick and fast-tick processes are formalized as explicit directed acyclic graphs, enabling deterministic execution and transparent state transitions.

A key result of the implementation is the feasibility of artifact-based state management for trading systems. All critical system state, including active trading pairs, execution signals, and performance summaries, is persisted to disk and explicitly consumed by downstream components. This design eliminates hidden in-memory dependencies and enables safe restarts, inspection, and replay of agent behavior.

The system further demonstrates end-to-end observability of decision workflows. Using LangSmith, all execution steps, inputs, and outputs are logged consistently, allowing post-hoc analysis of both trading decisions and higher-level control actions. This traceability provides a practical foundation for debugging, evaluation, and governance in learning-based trading systems.

While the present implementation does not introduce autonomous control rules, it establishes explicit control points for structural adaptation. The architecture supports monitored pair reselection, model retraining requests, and system pausing based on realized performance, although the triggering of these actions is deferred to future work.

Overall, the results indicate that agent-based orchestration frameworks can provide significant benefits in terms of modularity, transparency, and experimental rigor for quantitative trading systems.

7 Discussion

7.1 Reinforcement Learning

The reinforcement learning experiments highlight the critical role of transaction costs in determining the economic viability of learned trading policies. In the absence of transaction fees, the RL agent achieves strong positive returns and favorable risk-adjusted performance, demonstrating its ability to exploit short-term patterns in the data. However, even relatively low transaction costs lead to a substantial deterioration in performance, with returns turning negative under realistic fee levels and declining further as fees increase. This indicates that transaction costs constitute a first-order constraint for active, high-frequency RL-based trading strategies.

The agent's behavior adapts systematically to the presence of transaction costs. As fees increase, the learned policy reduces trading intensity, leverage, and turnover, with actions becoming more concentrated around the neutral position. This suggests that the agent internalizes execution costs through the reward function and responds by adopting a more conservative trading style. While this behavioral adaptation mitigates drawdowns, it is insufficient to fully offset the negative impact of transaction costs at higher fee levels.

From a broader perspective, the results underscore the challenges of applying reinforcement learning to high-frequency trading environments. Strategies that rely on frequent position adjustments are particularly vulnerable to execution costs, and apparent profitability in frictionless settings does not necessarily translate into economic gains once realistic trading frictions are introduced. This emphasizes the importance of explicitly modeling transaction costs when evaluating RL-based trading systems.

Across all fee regimes, the RL agent consistently outperforms the equal-weight baseline, which maintains a static 50/50 long allocation across asset pairs. While the baseline performs poorly in absolute terms, its inclusion provides a useful reference point and confirms that the learned policy captures meaningful cross-asset dynamics beyond naive buy-and-hold exposure. Nonetheless, outperforming such a baseline alone is insufficient to guarantee practical profitability under realistic market conditions.

7.2 AI Agents

The integration of AI agents in this project demonstrates that end-to-end automation of data ingestion, learning, decision-making, and execution is feasible in decentralized derivatives markets. By combining reinforcement learning models with an agent-based orchestration layer, the system coordinates multiple components—including market data retrieval, model inference, performance evaluation, and order execution—within a unified and auditable pipeline.

Crucially, the role of the AI agent in this architecture is not to replace core trading logic, but to structure, supervise, and govern it. Reinforcement learning policies and statistical selection mechanisms operate as modular components,

while the agent layer formalizes their interaction, execution order, and state transitions. This separation enables reproducibility, explicit state management, and post-hoc analysis without introducing additional decision authority into latency-sensitive trading paths.

The agent architecture further enables closed-loop evaluation by incorporating realized execution outcomes into a supervisory control process. By monitoring performance metrics and triggering structural actions such as pair reselection, retraining requests, or system pausing, the agent facilitates adaptive experimentation while preserving strict separation between inference, execution, and governance. Importantly, these control mechanisms operate on coarse time scales and do not override per-tick trading decisions.

Despite these advantages, the use of AI agents does not fundamentally alter the economic constraints faced by trading strategies. Market microstructure frictions—including transaction costs, liquidity conditions, and execution latency—remain binding regardless of the sophistication of the agent architecture. In this sense, AI agents function as an enabling systems layer that improves automation, modularity, and experimental rigor, rather than as a mechanism for overcoming structural limits of financial markets.

Overall, the findings suggest that the primary contribution of AI agents in decentralized trading systems lies in orchestration, observability, and governance. These capabilities support more reliable deployment and evaluation of learning-based strategies, while leaving core economic performance determined by market structure and model quality.

8 Conclusion

8.1 Summary of Results

This study examined the application of reinforcement learning–based AI agents to statistical arbitrage in cryptocurrency perpetual futures markets, with an emphasis on economic realism and practical deployability. Using a reinforcement learning framework trained on high-frequency market data, the agent was evaluated under multiple transaction cost regimes representative of real-world exchange conditions. The empirical results show that while the agent is capable of learning economically meaningful trading behavior and exploiting relative price dynamics in frictionless environments, its performance deteriorates substantially once realistic transaction costs are introduced. Even comparatively low fees lead to reduced profitability and increased drawdowns, highlighting transaction costs as a first-order constraint for active, high-frequency RL-based trading strategies in perpetual futures markets. Importantly, the agent adapts its behavior in response to higher costs by reducing exposure and trading intensity, indicating that the learned policy internalizes execution frictions through the reward structure. However, this behavioral adjustment is insufficient to fully offset the negative impact of fees at realistic levels, underscoring the limitations of frequent rebalancing strategies in this setting.

8.2 Future Research

8.2.1 Reinforcement Learning Beyond the empirical findings, this work demonstrates the feasibility of an end-to-end trading architecture that integrates data processing, reinforcement learning, and automated execution. The implemented AI agent serves as a proof of concept for system orchestration and live deployment; however, it does not currently provide a dedicated pipeline for online fine-tuning or continual learning. Extending the agent with systematic retraining or adaptive fine-tuning mechanisms represents a natural direction for future research.

PPO's performance is highly sensitive to the choice of hyperparameters. Consequently, one of potential rooms for improvement lies in a more systematic and careful hyperparameter tuning process. This was not undertaken in the present work due to resource constraints and because the primary objective of the study was the development and validation of a proof-of-concept agent rather than exhaustive performance optimization. Also, alternative reinforcement learning algorithms, such as Soft Actor-Critic (SAC) or Deep Q-Networks (DQN), may offer improved exploration–exploitation trade-offs or greater robustness under noisy reward signals. From a modeling perspective, incorporating beta estimation directly into portfolio weight calculations could strengthen the link between statistical relationships and execution decisions.

Future research could explore alternative reward function designs to better align reinforcement learning objectives with economically meaningful performance measures. In particular, reward formulations based on risk-adjusted metrics such as drawdown-sensitive utilities, downside-risk penalties, or multi-objective objectives that jointly optimize returns and trading costs may improve robustness under realistic market frictions.

Reducing the sampling frequency from one minute to coarser intervals, such as 30 minutes, may mitigate transaction cost sensitivity and improve signal stability.

Finally, increased computational resources, particularly additional memory, would allow for larger lookback windows and richer state representations, potentially enabling the agent to capture longer-term dependencies and regime dynamics. Together, these directions point toward more scalable and economically viable applications of reinforcement learning–based AI agents in decentralized derivatives markets.

8.2.2 AI Agent The agent architecture presented in this work establishes a structured and observable foundation for quantitative trading systems; however, several extensions are required to transition from a proof of concept toward a more autonomous and production-ready framework. The following directions represent natural and well-scoped avenues for future work.

Although the current implementation records detailed execution traces and performance summaries, structural decisions such as pair reselection, model retraining, or system pausing are not yet triggered autonomously. A natural extension is the introduction of automated control rules that evaluate logged metrics

and initiate such actions based on predefined criteria. LangSmith automation mechanisms provide a suitable substrate for implementing these rules in a transparent and auditable manner.

The present architecture relies on in-process workflows and artifact-based coordination. Introducing server-side webhooks would enable controlled external interaction with the agent in response to automation events, such as triggering slow-tick workflows or initiating retraining pipelines. This would further decouple orchestration logic from deployment-specific constraints and support more flexible operational setups, including distributed training and monitoring services.

Performance evaluation currently relies on aggregated metrics computed over fixed windows. Future extensions could integrate richer diagnostic signals, such as execution-quality measures, liquidity indicators, and regime classification features. These additions would improve monitoring sensitivity and support more nuanced control decisions, including regime-aware retraining or conditional policy selection.

Additionally, the decision making process of the AI agent could be supplemented by a social media node that explicitly scans social media outlets for news or trends on relevant crypto assets. This may allow the agent to be a step ahead and enable more profitable trades.

In the longer term, the architectural principles demonstrated in this work—explicit state management, artifact-based coordination, and end-to-end observability—could be extended toward more autonomous agent systems combining multiple learning components and hierarchical control structures. Future research should prioritize transparency, auditability, and controllability alongside increased automation, particularly in financial domains subject to strict operational and risk constraints.

References

1. Ackerer, D., Hugonnier, J., Jermann, U.: Perpetual futures pricing. NBER Working Paper Series **32936** (2024), https://www.nber.org/system/files/working_papers/w32936/w32936.pdf
2. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning **47**(2–3), 235–256 (2002)
3. Binance Research: Full-Year 2023 & Themes for 2024. Industry intelligence report, Binance (Jan 2024)
4. Brim, A.: Deep reinforcement learning pairs trading with a double deep Q-network. In: 2020 10th Annual Computing and Communication Workshop and Conference (CCWC). pp. 0222–0227. IEEE (2020)
5. Brockman, G., et al.: Openai gym (2016), arXiv preprint
6. Chamber, D.: How perpetual futures differs from traditional futures and why it matters for crypto (2024), https://digitalchamber.org/how-perpetual-futures-differs-from-traditional-futures-and-why-it-matters-for-crypto/
7. CoinGecko: 2023 Annual Crypto Industry Report (Jan 2024), https://www.coingecko.com/research/publications/2023-annual-crypto-report
8. DefiLlama: Derivatives (Perpetuals) Market Volume Dashboard (2024), https://defillama.com/derivatives
9. Dulac-Arnold, G., et al.: Challenges of real-world reinforcement learning (2021), arXiv preprint
10. Duron-Carielo, J., of Business, S.S.: Is there a future in perpetual futures? NYU Stern Research (2024), https://www.stern.nyu.edu/sites/default/files/assets/documents/Duron-Carielo_Is%20There%20A%20Future%20In%20Perpetual%20Futures.pdf
11. Engle, R.F., Granger, C.W.J.: Co-integration and error correction: representation, estimation, and testing. Econometrica **55**(2), 251–276 (1987)
12. Fischer, T.G.: Deep reinforcement learning in financial markets: A survey. Working Paper, University of Erlangen-Nuremberg (2018)
13. Fischer, T.G., Krauss, C., Deinert, A.: Statistical arbitrage in cryptocurrency markets. Journal of Risk and Financial Management **12**(1), 31 (2019)
14. Gatev, E., Goetzmann, W.N., Rouwenhorst, K.G.: Pairs trading: Performance of a relative-value arbitrage rule. The Review of Financial Studies **19**(3), 797–827 (2006)
15. Hambly, B., Lian, R., Siska, J.: Recent advances in reinforcement learning in finance. Mathematical Finance **33**(3), 437–503 (2023)
16. Hambly, B., Xu, R., Yang, H.: Recent advances in reinforcement learning in finance. Mathematical Finance **33**(3), 437–503 (2023)
17. He, Z., Manela, A., Ross, S.A., von Wachter, V.: Fundamentals of perpetual futures. NBER Working Paper Series w30613, National Bureau of Economic Research (2023)
18. Hummingbot Foundation: Public Hummingbot GitHub repository (2024), https://github.com/hummingbot/hummingbot, accessed: 2025-12-21
19. Hyperliquid: Hyperliquid Trading Fees (2024), https://hyperliquid.gitbook.io/hyperliquid-docs/trading/fees, accessed: 2025-03-12
20. Jiang, Z., Xu, D., Liang, J.: A deep reinforcement learning framework for the financial portfolio management problem. arXiv preprint arXiv:1706.10059 (2017). https://doi.org/10.48550/arXiv.1706.10059, https://arxiv.org/abs/1706.10059

21. Johansen, S.: Estimation and hypothesis testing of cointegration vectors in Gaussian vector autoregressive models. Econometrica **59**(6), 1551–1580 (1991)
22. Kim, S.H., Park, D.Y., Lee, K.H.: Hybrid deep reinforcement learning for pairs trading. Applied Sciences **12**(3), 944 (2022)
23. Ko, P.C., Lin, P.C., Do, H.T., Kuo, Y.H., Mai, L.M., Huang, Y.F.: Pairs trading in cryptocurrency markets: A comparative study of statistical methods. Investment Analysts Journal **53**(2), 102–119 (2024)
24. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. The International Journal of Robotics Research **32**(11), 1238–1274 (2013)
25. Krauss, C.: Statistical arbitrage pairs trading strategies: Review and outlook. Journal of Economic Surveys **31**(2), 513–545 (2017)
26. Leung, T., Li, X.: Optimal mean reversion trading with transaction costs and stop-loss exit. International Journal of Theoretical and Applied Finance **18**(03), 1550020 (2015)
27. Liang, E., et al.: RLlib: Abstractions for distributed reinforcement learning. In: International Conference on Machine Learning (ICML) (2018)
28. Ma, C., Sarkissian, S., Zhang, Y.: Crypto carry. BIS Working Papers 1087, Bank for International Settlements (2023)
29. Makarov, I., Schoar, A.: Trading and arbitrage in cryptocurrency markets. Journal of Financial Economics **135**(2), 293–319 (2020). https://doi.org/10.1016/j.jfinec.2019.10.011
30. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
31. Moody, J., Saffell, M.: Learning to trade via direct reinforcement. IEEE Transactions on Neural Networks **12**(4), 875–889 (Jul 2001). https://doi.org/10.1109/72.935097, https://www.cs.utexas.edu/~shivaram/readings/b2hd-MoodySaffell2001.html
32. Nevmyvaka, Y., Feng, Y., Kearns, M.: Reinforcement learning for optimized trade execution. In: Proceedings of the 23rd International Conference on Machine Learning (ICML). Pittsburgh, PA, USA (2006), https://www.cis.upenn.edu/~mkearns/papers/rlexec.pdf
33. Ning, B., Lee, K.: Advanced statistical arbitrage with reinforcement learning. arXiv preprint arXiv:2403.12180 (2024)
34. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, New York, NY (1994)
35. Raffin, A., et al.: Stable-Baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research **22**(268), 1–8 (2021)
36. Roychoudhury, R., Bhagtani, R., Daftari, A.: Pairs trading using clustering and deep reinforcement learning. Available at SSRN 4504599 (2023)
37. Sarkar, S.: Harnessing Deep Q-Learning for enhanced statistical arbitrage in high-frequency trading: A comprehensive exploration. arXiv preprint arXiv:2311.10718 (2023)
38. Shaleva, E., Rossi, M.: Perpetual futures in crypto markets: Mechanisms and market impact. Journal of Digital Finance (2024)
39. Shiller, R.J.: Measuring asset values and cash flows. Journal of Finance and Quantitative Analysis (1993), original theoretical framework for perpetual contracts.
40. Silver, D., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
41. Spooner, T., Fearnley, J., Savani, R., Koukorinis, A.: Market making via reinforcement learning (2018). https://doi.org/10.48550/arXiv.1804.04216, https://arxiv.org/abs/1804.04216, aAMAS 2018

42. Sun, S., Wang, R., An, B.: Reinforcement learning for quantitative trading. ACM Transactions on Intelligent Systems and Technology **14**(3), 1–29 (2023)
43. Sutton, R.S.: Learning to predict by the methods of temporal differences. Machine Learning **3**(1), 9–44 (1988)
44. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, 2nd edn. (2018)
45. Thrun, S.: Efficient exploration in reinforcement learning. Tech. rep., Carnegie Mellon University (1992)
46. Vergara, G., Kristjanpoller, W.: Deep reinforcement learning applied to statistical arbitrage investment strategy on cryptomarket. Applied Soft Computing **153**, 111255 (2024)
47. Yang, H., Malik, A.: Reinforcement learning pair trading: A dynamic scaling approach. Journal of Risk and Financial Management **17**(12), 555 (2024)
48. Yang, H., Liu, X.Y., Zhong, S., Walid, A.: Deep reinforcement learning for automated stock trading: An ensemble strategy. In: ACM International Conference on AI in Finance (ICAIF '20) (2020). https://doi.org/10.1145/3383455.3422540, https://openfin.engineering.columbia.edu/sites/default/files/content/publications/ensemble.pdf
49. Yu, C., Liu, J., Nemati, S., Sun, J.: Reinforcement learning in healthcare: A survey. ACM Computing Surveys **54**(6), 1–36 (2021)

A Detailed Implementation and Selection Rationales

A.1 Selection Criteria for Trading Platforms

Our evaluation focused on three pillars to ensure the system's robustness in a live DeFi environment:

- **API Performance**: Prioritizing exchanges with robust WebSocket streams to minimize the latency overhead associated with REST polling.
- **Matching Architecture**: Favoring Central Limit Order Book (CLOB) protocols over Automated Market Makers (AMMs) to provide a richer state representation for feature engineering.
- **Fee Transparency**: Requiring clearly defined fee tiers to internalize transaction costs as binding constraints during the reinforcement learning process.

A.2 Exploratory Hyperliquid S3 Data Pipeline

To evaluate the feasibility of a native Hyperliquid training environment, we developed a data pipeline for raw chain archives:

- **Data Processing**: Raw archives from the Hyperliquid mainnet S3 bucket (`s3://hl-mainnet-node-data`) were processed using streaming decompression to handle the 58.2 GB dataset efficiently.
- **Implementation Outcome**: The resulting dataset only covered the period from March 22, 2025, which verified the necessity of using Binance Futures as the primary training proxy for long-term historical analysis.

A.3 Consolidated Data Dictionary (Exploratory)

The following table defines the schema and alignment policies established during the processing of the Hyperliquid raw archives:

Table 5. Data Dictionary for Raw Hyperliquid S3 Archives

Field	Type	Description / Alignment Policy
`time`	`datetime64`	Primary index in UTC Unix ms.
`open, high, low, close`	`float64`	Capped by IQR; Forward-filled (`ffill`) for bars with no trades.
`volume`	`float32`	Aggregate base asset volume; filled with 0.0 for inactivity.
`fundingRate`	`float64`	Hourly rate merged via `merge_asof` (backward).

A.4 Example of the signal.json File

The following data depicts a sample signal in JSON-format that has been used to execute the trades in Hummingbot:

```
{
    "timestamp": "2025-12-19T13:00:36Z",
    "pair": {
        "asset1": "ETH",
        "asset2": "BTC"
    },
    "markets": {
        "exchange": "hyperliquid_perpetual",
        "asset1": "ETH-USDT",
        "asset2": "BTC-USDT"
    },
    "weights": {
        "asset1": 0.012192172929644585,
        "asset2": -0.012192172929644585
    },
    "notional_usd": 100
}
```


A.5 Signal-to-execution workflow

The following diagram portrays the execution workflow from signal (weight) generation to the submission in the venue.

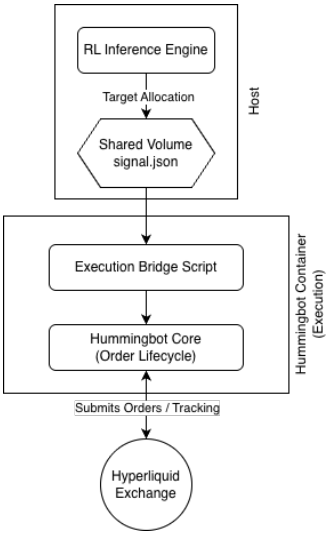

Fig. 8. Signal-to-execution workflow. The RL inference engine writes target allocations to signal.json on a shared host–container volume. An execution bridge script inside Hummingbot polls and validates the signal, converts target weights into orders, and Hummingbot submits and tracks these orders via the Hyperliquid connector.