Orchestration Framework for Financial Agents: From Algorithmic Trading to Agentic Trading

Jifeng Li[1], **Arnav Grover**[2], **Abraham Alpuerto**[3], **Yupeng Cao**[4], **Xiao-Yang Liu**[1]*
[1]SecureFinAI Lab, Columbia University, [2]Purdue University,
[3]Rensselaer Polytechnic Institute, [4]Stevens Institute of Technology

Abstract

The financial market is a mission-critical playground for AI agents due to its temporal dynamics and low signal-to-noise ratio. Building an effective algorithmic trading system may require a professional team to develop and test over the years. In this paper, we propose an orchestration framework for financial agents, which aims to democratize financial intelligence to the general public. We map each component of the traditional algorithmic trading system to agents, including planner, orchestrator, alpha agents, risk agents, portfolio agents, backtest agents, execution agents, audit agents, and memory agent. We present two in-house trading examples. For the stock trading task (hourly data from 04/2024 to 12/2024), our approach achieved a return of 20.42%, a Sharpe ratio of 2.63, and a maximum drawdown of -3.59%, while the S&P 500 index yielded a return of 15.97%. For the BTC trading task (minute data from 27/07/2025 to 13/08/2025), our approach achieved a return of 8.39%, a Sharpe ratio of 0.38, and a maximum drawdown of -2.80%, whereas the BTC price increased by 3.80%.

1 Introduction

From floor trading with chalkboards and open outcry to telephone order routing, and then to algorithmic trading, the market microstructure has reorganized how orders are created, conveyed, and executed [24, 3, 9]. The algorithmic trading (AT) system [24] follows a pipeline from processing financial data, extracting trading signals, portfolio management to execution and evaluation. Designing an effective AT system may require a professional team to develop and test over years. Recent works have demonstrated the great potential of AI agents: reasoning-and-acting [29], self-teaching for using tools [20], generative agents [19], reflection and memory [21], and multi-agent role coordination [10]. The financial market is a particularly challenging playground for AI agents due to its unique features of temporal dynamics and low signal-to-noise ratio. In particular, **agentic trading** is a mission-critical task in a high-stakes domain.

In this paper, we propose an end-to-end orchestration framework for financial agents, which maps the components of the traditional AT system to agents and democratizes financial intelligence for the general public. First, we map each component of the AT approach to agents, including planner, orchestrator, alpha agents, risk agents, portfolio agents, backtest agents, execution agents, audit agents, and memory agent. Second, we use the Model Context Protocol (MCP)[2] for control messages between the orchestrator and agents and the Agent-to-Agent protocol (A2A) for peer-to-peer communication among agents, while a memory agent records states, prompts, tool calls, and decisions.

*Corresponding author.
[2]MCP standardizes how applications provide context and tools to LLMbased agents.

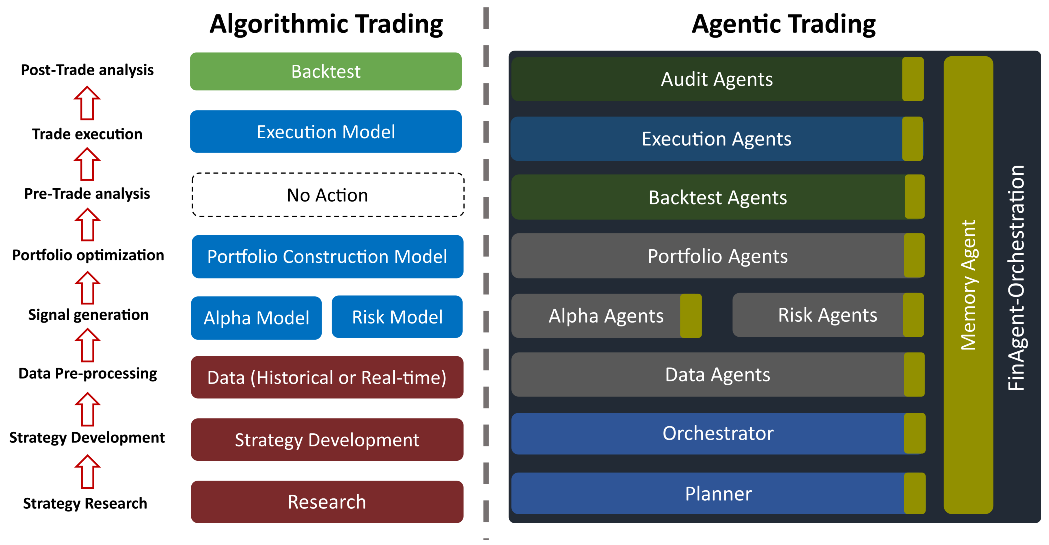

Figure 1: Agentic trading vs. algorithmic trading: we map the AT components to agents in our FinAgent orchestration framework, where a memory agent provides the contexts to other agents.

Finally, we develop two homegrown trading examples. For the stock trading task backtested from 04/2024 to 01/2025 (hourly data), our agents achieve a return of 20.42%, volatility of 11.83% and Sharpe ratio of 2.63 with max drawdown of -3.59%, while the S&P 500 index has a return of 15.97%; however, the equally weighted method with weekly rebalance has a return of 47.46%. For the BTC trading task backtested from 27/07/2025 to 13/08/2025 (minute data), our agents achieve a return of 8.39%, volatility of 24.23% and Sharpe ratio of 0.378 with max drawdown of -2.80%, while the BTC price increased 3.80%.

2 Proposed Framework for FinAgent Orchestration

2.1 Overview

We build a FinAgent orchestration framework structured around multiple agent pools, each orchestrating a stage (data, alpha, risk, portfolio, execution) so that the system runs end-to-end from raw data to trading orders. We use several LLM models (e.g., GPT-4o [18, 17], Llama3 [5], FinGPT [13]) to power different agents. Data agents pull from multiple sources (e.g., Polygon and yfinance) [1, 23]. We compare coverage, consistency, and delay across sources and keep the better ones; we then align time and symbols, clean errors and gaps, and form simple features [12, 28]. The cleaned data is fed into alpha and risk agents. Alpha agents propose signal structures, while tool-based modules compute the numerical signals, and risk agents compute exposures and limits [24, 3, 9]. We check signals with different metrics (e.g., rank-IC), rolling tests, and a walk-forward backtest [12, 9, 28].Signal diagnostics (e.g., rank-IC) are computed by tool modules and never exposed to LLMs. Approved signals are fed into portfolio agents.

We backtest long-only and long-short rules under capital and turnover constraints [9, 3]. The system keeps only signals, risk rules, and portfolios that pass these checks [26, 11]. Next, we run simulated or live trading and adjust settings by market. Evaluation and attribution agents verify equity curves, drawdowns, contributions, and write a full log [26, 11]. The planner and the orchestrator use these logs to update the plan for the next loop, while avoiding any use of evaluation-window outcomes in agent prompts. And the memory module keeps the state for reuse [31, 30, 32].

2.2 Control Messages and Agents' Communications

Control Messages. All agent pools are controlled by the orchestrator through MCP. The orchestrator sends small control messages that describe the task (node type, task id, declared inputs with schemas, policy flags, timeout, retry budget) and waits for a reply (acknowledgement, status, logs, artifact ids); it also tracks health with heartbeats and monitors completion until the tasks finish [15, 14, 26, 11]. MCP exposes each agent pool as a tool-like endpoint with a unified requestresponse schema.[15, 14].

Inside a pool, a manager agent breaks a task into subtasks and assigns them to subordinate agents for cooperative execution; partial results are merged and returned upstream [10, 25].

Agents' communications. After a task is issued, each agent pool uses A2A to talk with the memory agent to read prior context and to upload logs and key results, so that state and rationale persist across runs [31, 30, 26, 11]. Memory stores only structural summaries, not evaluation-window labels. Within a pool, agents of the same or different types also coordinate over A2A to complete the task; messages use simple types (ask, tell, propose, confirm) with role tags and context ids, following standard agent communication patterns [6, 34]. Agents share progress at fixed intervals; if a collaborator fails, a peer can take over or the orchestrator can reassign the job. All peer exchanges are time-stamped and stored in memory for replay and audit [31, 30].

3 Homegrown Trading Examples

3.1 Stock Trading and Crypto Trading Tasks

Stock backtesting pipeline. We employ a unified plan graph and interface; only data sources, horizons, and scheduling change. For stocks, data agents fetch hourly bars from Polygon and yfinance, dedupe and align calendars, correct anomalies, and compute baseline features (returns, momentum, volatility, volume ratios) [1, 23, 12, 28]. Following our prompt-design constraints, the Alpha Agents propose factor structures based only on published literature and do not access any evaluation-window data. All numerical signal construction and return mapping are handled by tool-based modules. Risk agents compute exposures (market, sector, name) and enforce constraints (concentration, volatility, drawdown). Portfolio agents test long-only and longshort rules under capital and turnover limits; execution translates weights to orders with slippage/transaction cost models and reconciles fills. Evaluation runs walk-forward backtesting, attribution, and metric aggregation using the same plan graph, while keeping realized returns and performance metrics hidden from LLM agents [24, 3, 9].
Crypto backtesting pipeline. The BTC pipeline reuses the same plan graph and message schema, but with minute-level bars and intraday prompts. Data agents ingest minute bars from Polygon [1], dedupe and align timestamps, and aggregate features on a decision clock. Following the same prompt-design rules, the Alpha Agents propose short-horizon microstructure factor structures (order flow imbalance, spread, volume spikes) based only on prior literature and do not access any evaluation-window data. All numerical feature transformations and signal computations are carried out by tool-based modules. Risk agents impose stricter caps on realized volatility, position size, and drawdown, and apply drift or volatility gates before execution. Portfolio sizing follows a long-flat regime with turnover control. Execution sends orders only when all gates pass and logs fills considering latency. Evaluation aggregates results to daily metrics (volatility, Sharpe, drawdown) while keeping realized returns and performance outcomes hidden from LLM agents, maintaining the same data path as the stock pipeline [12, 26, 11].

3.2 Backtesting Performance

Experimental setup.

We run one orchestration pipeline across stocks and BTC. The initial assets are all \$100,000, and transactions are conducted in units of total dollar amount. The plan graph is similar, while schedules and rebalancing methods differ by markets. For stocks we backtest on a static seven-stock universe (AAPL, MSFT, GOOGL, JPM, TSLA, NVDA, META) from 09/2022 to 01/2025 by hourly. We use GPT-4o [18, 17] to survey prior factor studies and to draft feature lists; in all prompts we restrict materials to published sources and do not expose any test-period market data, so the LLM cannot leak labels or prices from the backtest window. Baselines are the S&P 500 (SPY), QQQ, IWM, VTI, and an equal-weighted portfolio (weekly rebalance). For BTC we use minute bars from 05/2025 to 08/2025; positions update every minute, and trades trigger only when drift and rule gates are met; the baseline is Buy&Hold.

Stocks (multi–stock agentic portfolio). Table 1 and Fig. 2 report that the equally weighted benchmark attains the highest total return (47.46%, Sharpe ratio 3.37). Ours delivers total return 20.42%, the lowest volatility (11.83%) and the smallest max drawdown (-3.59%), with Sharpe ratio 2.63 (ETF range 0.79–1.86). The profile is risk–controlled and consistent with execution gated by the risk and portfolio pools.

Table 1: Comparison of trading performance. Arrows indicate: ↑=higher is better; ↓=lower is better (for Max Drawdown, less negative is better). B&H denotes a Buy & Hold strategy, where other ETFs are purchased at the start and held for the entire evaluation period. EW refers to an equally weighted portfolio constructed across all selected stocks or assets. Sharpe ratios are computed from daily/weekly returns with $R_f = 0$ due to short evaluation horizons. MDD: maximum drawdown.

Metric	Ours	SPY	QQQ	IWM	VTI	EW	BTC (Ours)	BTC (B&H)
Total Return ↑	20.42%	16.60%	21.59%	11.45%	16.29%	**47.46%**	**8.39%**	3.80%
Annual Return ↑	31.08%	25.07%	32.94%	17.10%	24.59%	**76.07%**	–	–
Volatility ↓	**11.83%**	13.49%	18.38%	21.61%	13.72%	22.54%	**24.23%**	25.82%
Sharpe Ratio ↑	2.63	1.86	1.79	0.79	1.79	**3.37**	**0.378**	0.170
MDD (%) ↑	**-3.59**	-8.89	-14.13	-11.60	-9.06	-16.21	**-2.80**	-5.26

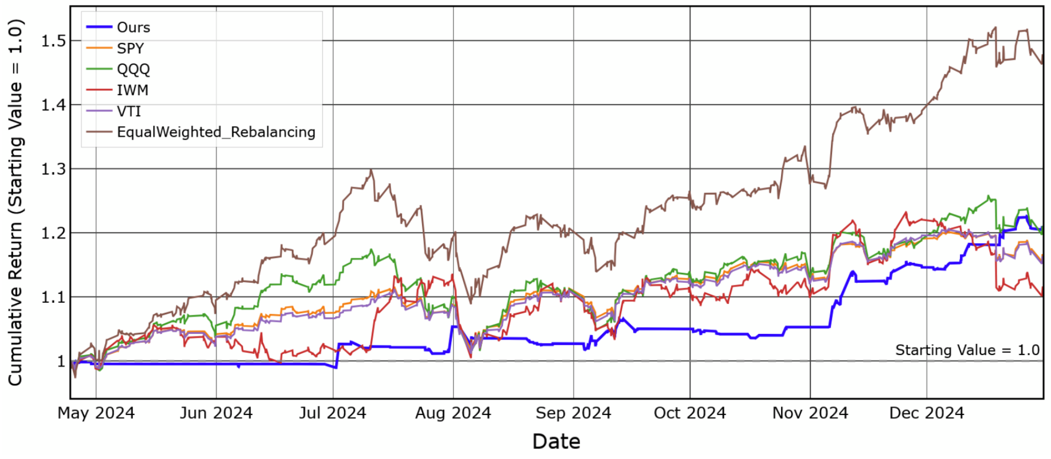

Figure 2: Seven-stock cumulative returns with the test window from 24/04/2024 to 31/12/2024 (within the 2022–2024 sample, the scrolling training window size is 3 months). The agentic strategy shows lower volatility and a smaller max drawdown, while the equally-weighted benchmark attains the highest total return. ETF baselines: SPY, QQQ, IWM, VTI. Metrics are reported in Table 1.

Cryptocurrency (BTC, minute data). Fig. 3 and Table 1 show that, on BTC/USDT, the Buy&Hold benchmark ends at +3.80% while our strategy finishes at approximately +8.4%. Over this short 17-day window, the BTC strategy earns about 4.6 percentage points of excess return over Buy-and-Hold, with lower realized volatility and smaller max drawdown. This mirrors the equity experiment, where the agentic strategy trades off some total return for tighter risk. $(\text{Excess} = \text{Ours} - \text{Buy\&Hold})$.

4 Conclusion

We proposed an orchestration framework for FinAgents as a step from traditional algorithmic trading frameworks [24, 3, 9] toward agent-based trading. We mapped the components of the traditional algorithmic trading system into agents. with standard control/peer channels and a shared memory record for observability and auditability [15, 34, 12, 32, 30, 31, 26, 11]. Using one unified pipeline across a static seven-stock equity universe and BTC, we observe on equities a total return over the S&P 500 index and ETFs with the lowest volatility and smallest max drawdown; on BTC minute data, Ours is +8.39% versus Buy&Hold +3.80%, with lower variance and partial capture of a short trend. In our small-scale experiments, a single agent-based plan worked on both a stock universe and BTC, and produced returns with lower volatility and drawdowns than simple benchmarks.

Future work includes longer horizons and markets, ablations on gating/memory/messaging, adaptive planner updates under regime shifts [7], broader information sources for signals [4, 16], and releasing benchmarks and logs for replication.

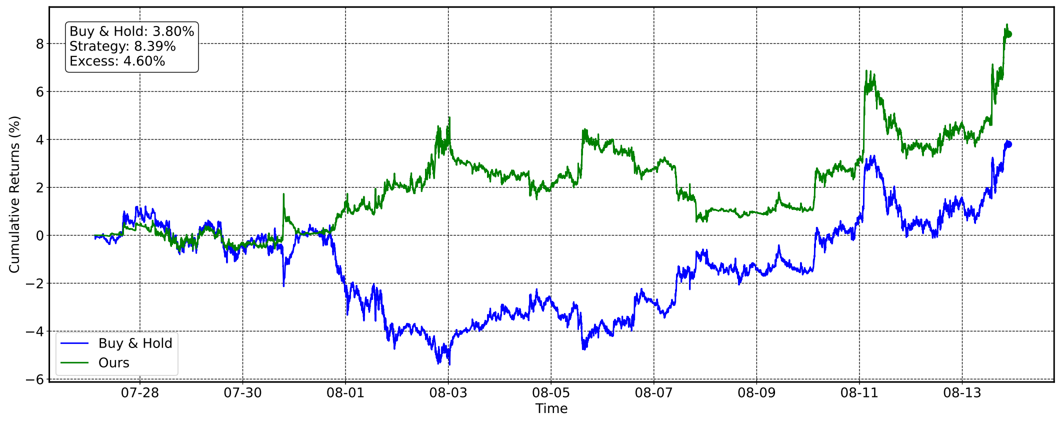

Figure 3: BTC test results (07–27 to 08–13 in 2025, the scrolling training window is 7 days). Cumulative returns: Buy&Hold $+3.80\%$, Ours $+8.39\%$, Excess $+4.59\%$. Excess $=$ Ours $-$ Buy&Hold (percentage points).

References

[1] Polygon.io api documentation. https://polygon.io/docs/stocks/getting-started. Accessed: 2025-08-11.

[2] B. A. alpha-arena: A benchmark for ai investing abilities. https://github.com/AmadeusGB/alpha-arena, 2024. GitHub Repository.

[3] Irene Aldridge. *High-Frequency Trading: A Practical Guide to Algorithmic Strategies and Trading Systems*. Wiley, Hoboken, NJ, 2013.

[4] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. Deep learning for event-driven stock prediction. *AAAI*, 2015.

[5] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407, 2024.

[6] FIPA. Fipa agent communication language (acl): Message structure specification. https://www.fipa.org/specs/fipa00061/SC00061G.html, 2002. Accessed: 2025-08-29.

[7] Xin Guo, Tze Leung Lai, Howard Shek, and Samuel Po-Shing Wong. *Quantitative Trading: Algorithms, Analytics, Data, Models, Optimization*. Chapman and Hall/CRC, 2017.

[8] M. Jin et al. When ai meets finance (stockagent): Large language model-based stock trading in simulated real-world environments. *arXiv preprint*, 2024.

[9] Robert Kissell. *The Science of Algorithmic Trading and Portfolio Management*. Academic Press, San Diego, CA, 2013.

[10] Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008, 2023.

[11] Yuante Li, Xu Yang, Xiao Yang, Minrui Xu, Xisen Wang, Weiqing Liu, and Jiang Bian. R&d-agent-quant: A multi-agent framework for data-centric factors and model joint optimization. *arXiv preprint arXiv:2505.15155*, 2025. URL https://arxiv.org/abs/2505.15155.

[12] Xiao-Yang Liu, Jingyang Rui, Jiechao Gao, Liuqing Yang, Hongyang Yang, Christina Dan Wang, Jian Guo, and Zhaoran Wang. FinRL-Meta: A universe of near-real market environments for data-driven deep reinforcement learning in quantitative finance. In *NeurIPS 2021 Workshop on Data-Centric AI*, 2022.

[13] Xiao-Yang Liu, Guoxuan Wang, Hongyang Yang, and Daochen Zha. FinGPT: Democratizing internet-scale data for financial large language models. *Workshop on Instruction Tuning and Instruction Following, NeurIPS*, 2023.

[14] Model Context Protocol Working Group. Model context protocol (mcp): Specification and documentation repository. https://github.com/modelcontextprotocol/modelcontextprotocol, 2025. Accessed: 2025-08-29.

[15] Model Context Protocol Working Group. Model context protocol (mcp): Official documentation. https://modelcontextprotocol.io/, 2025. Accessed: 2025-08-29.

[16] Amir H. Nassirtoussi, Saeed Aghabozorgi, Teh Ying Wah, and David C.L. Ngo. Text mining for market prediction: A systematic review. *Expert Systems with Applications*, 41(16), 2014.

[17] OpenAI. Gpt-4o system card, 2024. URL https://openai.com/index/gpt-4o-system-card/. Accessed 2025-09-01.

[18] OpenAI. Hello gpt-4o, 2024. URL https://openai.com/index/hello-gpt-4o/. Accessed 2025-09-01.

[19] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22, 2023.

[20] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

[21] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

[22] Virat Singh. Ai hedge fund: An ai hedge fund team. `https://github.com/virattt/ai-hedge-fund`, 2025. GitHub repository, accessed 2025-11-16.

[23] Ran Taha et al. yfinance python library. `https://pypi.org/project/yfinance/`. Accessed: 2025-08-11.

[24] Philip Treleaven, Michal Galas, and Vidhi Lalchand. Algorithmic trading review. *Communications of the ACM*, 56(11):76–85, 2013.

[25] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

[26] Yijia Xiao, Edward Sun, Di Luo, and Wei Wang. TradingAgents: Multi-agents llm financial trading framework. *arXiv preprint arXiv:2412.20138*, 2025.

[27] F. Xiong et al. Quantagent: Seeking holy grail in trading by self-improving large language model. *arXiv preprint*, 2025.

[28] Xiao Yang, Weiqing Liu, Dong Zhou, Jiang Bian, and Tie-Yan Liu. Qlib: An ai-oriented quantitative investment platform. *arXiv preprint arXiv:2009.11189*, 2020.

[29] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[30] Yangyang Yu, Zhiyuan Yao, Haohang Li, Zhiyang Deng, Yuechen Jiang, Yupeng Cao, Zhi Chen, Jordan Suchow, Zhenyu Cui, Rong Liu, et al. Fincon: A synthesized llm multi-agent system with conceptual verbal reinforcement for enhanced financial decision making. *Advances in Neural Information Processing Systems*, 37:137010–137045, 2024.

[31] Yangyang Yu, Haohang Li, Zhi Chen, Yuechen Jiang, Yang Li, Jordan W Suchow, Denghui Zhang, and Khaldoun Khashanah. Finmem: A performance-enhanced llm trading agent with layered memory and character design. *IEEE Transactions on Big Data*, 2025.

[32] Hengxi Zhang, Zhendong Shi, Yuanquan Hu, Wenbo Ding, Ercan E. Kuruoglu, and Xiao-Ping Zhang. Optimizing trading strategies in quantitative markets using multi-agent reinforcement learning. In *arXiv preprint arXiv:2303.11959*, 2023.

[33] R. Zhang et al. Contesttrade: A multi-agent trading system based on internal contest mechanism. *arXiv preprint*, 2025.

[34] Changxi Zhu, Mehdi Dastani, and Shihan Wang. A survey of multi-agent deep reinforcement learning with communication. *arXiv preprint arXiv:2203.08975*, 2022. URL `https://arxiv.org/abs/2203.08975`.

A More Test Details

Crypto backtesting pipeline. The BTC experiment uses the same agent classes and DAG topology as the equity pipeline, but runs on minute-level intraday data. Data Agents load raw OHLCV bars, funding rates, and open interest from Polygon [1], deduplicate timestamps, align to the exchange calendar, and build rolling microstructure features on a fixed decision clock (e.g., k-minute windows). All features are computed only from information available at or before each decision time; evaluation-window labels, summary statistics, and functions that depend on future data are not exposed to any agent.

Alpha Agents coordinate signal design. Given the feature schema and prior crypto microstructure literature, they specify short-horizon factor structures (e.g., order-flow imbalance, bid–ask spread and depth, volume or volatility spikes, funding-rate or open-interest signals) and issue tool calls to construct these factors numerically. Alpha Agents do not compute predictions and do not observe realized returns. Forecasting (directional or return-based) is performed by tool-based ML modules, such as Ridge or tree ensembles, trained on rolling in-sample windows with adjacent validation windows under chronological splits.

Risk Agents use tighter constraints than in the equity pipeline to reflect the higher volatility of BTC. They impose volatility targets, position-size caps, leverage limits, and drawdown thresholds. Additional drift and short-horizon volatility checks may block trades when intraday moves exceed predefined bounds. The Risk Agent outputs a single risk-adjusted exposure signal, which is passed to the Portfolio Agent.

Portfolio Agents map the risk-adjusted signal to long–flat target positions, apply smoothing to limit turnover, and enforce minimum-change thresholds to avoid small position updates. Execution Agents simulate intraday order placement using historical best bid/ask and depth snapshots, accounting for latency, fees, spreads, and partial fills. Orders are submitted only when all checks from Data, Alpha, Risk, and Portfolio have passed.

A Backtest Orchestrator runs this pipeline in a walk-forward setup, the same as in the equity case. Each block has a training window for ML tools, a validation window for hyperparameter and stability checks, and a separate test window used only for paper trading. Evaluation turns test-window trades into daily metrics such as realized volatility, Sharpe ratio, drawdown, and turnover [12, 26, 11]. Realized returns and performance outcomes are never given to any LLM, matching the data-flow rules used in the equity pipeline.

B BTC/USDT Trading Strategy

This appendix gives implementation details of the BTC/USDT high-frequency trading strategy used in our experiments. We summarize the feature design, the prediction model, the training procedure, and the basic safeguards against data leakage.

B.1 Feature Engineering and Data Preprocessing

We build a feature set with more than 100 inputs that describe price, volume, volatility, and trend at the 1-minute frequency. Basic price features include smoothed 5-minute returns, 15-minute realized volatility, 60-minute exponentially weighted moving (EWM) volatility, and the difference between the mid-price and the volume-weighted average price (VWAP). Technical indicators include relative strength index (RSI) with windows 14 and 30, MACD (line, signal, and histogram), and Bollinger Band features such as band position, band width, and simple overbought/oversold flags.

Momentum-related features cover multiple horizons $1/3/5/10/15/30/60/240$ minutes. They include price momentum, trend strength, trend consistency, trend alignment, breakout flags, price acceleration, momentum alignment, trend persistence, and a simple momentum quality score. Volatility features include proxies for volatility clustering (GARCH-style terms), volatility regime identifiers, within-bar price range, and range expansion signals. Additional features encode support and resistance levels, mean-reversion signals based on price z-scores, volume statistics, and pricevolume divergence.

All raw features are smoothed with an exponentially weighted moving average with decay parameter $\alpha = 0.4$. We then apply a RobustScaler transformation to reduce the influence of outliers and heavy tails. Finally, we remove low-variance features using a variance threshold of 0.01 and keep only the top 70% of features ranked by model importance. This reduces the number of inputs and makes the model more stable.

B.2 Model Architecture and Training Methodology

We use an XGBoost regression model to predict the next-minute return r_{t+1} from a vector of lagged features $\mathbf{x}_t$. The main hyperparameters are: 300 trees, maximum depth 6, learning rate 0.08, subsample rate 0.8, column subsample rate 0.8, ℓ_1 regularization 0.01, ℓ_2 regularization 0.05, minimum child weight 1, split penalty (gamma) 0.0, and up to 512 histogram bins.

Training follows a rolling walk-forward scheme. The model is retrained every 24 hours (1,440 minutes) using a minimum training window of 7 days (10,080 minutes), with a prediction horizon of 1 minute. At each step, historical data are split into a training window and a validation window. The validation window is used to monitor model performance and to adjust basic signal rules, but not to tune the strategy on realized test outcomes.

To avoid data leakage, all input features are computed using information up to time $t-1$, and the prediction target is the forward 1-minute return r_{t+1}, with at least a 2-minute gap between feature timestamps and labels. We use early stopping with a patience of 20 boosting rounds based on validation loss. At each retraining, we recompute feature importance on the training window and again keep only the top 70% most informative features for the next model fit.

B.3 Signal Generation and Market Regime Identification

The trading signal combines two parts: a model prediction and simple price-based rules (price action). The price-action part uses the following components:

- **Short-, medium-, and longer-horizon momentum:** a weighted sum of 1-, 5-, and 15-minute returns with weights 40%, 40%, and 20%. We multiply this sum by 10 so that it has a similar scale as other components.
- **Mean reversion:** a weighted sum of 20- and 60-period price deviations from a recent average (e.g., z-scores), multiplied by 5. This term is large when price moves far away from its recent level.
- **Breakout:** a discrete signal with value ± 3.0 when price breaks above or below recent highs or lows.
- **Trend following:** a term that combines agreement of trends across horizons (trend alignment) with trend strength, multiplied by 100 to reflect its importance.
- **Momentum acceleration:** the change in the 5-minute return from one step to the next, multiplied by 20.

We mix the model prediction with the price-action signal based on a simple measure of model signal quality q_t (for example, the absolute value of a standardized predicted return). The weight on the model is

$$w_{\text{model}}(q_t) = \begin{cases} 0.10, & q_t < 0.05, \\ 0.20, & 0.05 \le q_t < 0.10, \\ 0.40, & q_t \ge 0.10, \end{cases}$$

and the price-action part gets weight $1 - w_{\text{model}}(q_t)$. When the model signal is weak, we mostly follow price action; when the model signal is stronger, we give it more weight. We also classify simple market regimes using trend and volatility measures:

- **Strong trend:** trends across several horizons point in the same direction more than 75% of the time, and the trend strength score is above 0.1%.
- **Breakout:** price moves above or below a recent 20-period high or low.
- **Sideways:** trend strength is below 0.05% and a ratio of short-horizon to long-horizon volatility is above 1.2, suggesting choppy but directionless movement.

- **High volatility:** short-term volatility is more than 1.5 times long-term volatility.

Each regime uses a different mix of signal components. In strong-trend regimes, we emphasize momentum and trend signals and scale them by 1.5. In breakout regimes, we emphasize breakout and momentum-acceleration signals and double the breakout term. In sideways regimes, we put more weight on mean reversion and momentum and multiply the mean-reversion term by 1.2. In all other cases, we use a fixed mix of 70% momentum and 30% trend.

B.4 Position Sizing and Risk Management

Position size depends on the market regime we detect. We use a base size factor of 1.8 in strong-trend regimes, 2.5 in breakout regimes, 0.7 in sideways regimes, and 0.8 in high-volatility regimes. When the momentum score is higher than 70% of its past values, we increase this base size by up to 30%. Before trading, we first center and scale the raw signals using median absolute deviation (MAD), which measures how far values are from the middle, and then apply the hyperbolic tangent (tanh) function to keep very large values in a reasonable range. Entry thresholds are different in each regime: we use approximate percentile levels of the signal (45th percentile for trend regimes, 50th for breakout, 35th for sideways, and 40th for mixed regimes). Position limits keep each single position between 3% and 5% of capital and limit total leverage to 4.0.

Risk control combines fixed limits with rules based on cumulative loss (drawdown). We reduce positions when realized drawdowns grow: if drawdown goes beyond 1%, 2%, and 3%, we cut position sizes by 20%, 30%, and 50%, respectively. In addition, we check drawdowns at each step: if drawdown is above 1.5% or 2.5%, we immediately cut 50% or 75% of the current position. Each trade has a volatility-based stop-loss centered at -0.8%, scaled between 0.5 and 1.5 times this level depending on current volatility. A maximum drawdown limit of -3.0% closes all positions when it is reached. To reduce trading frequency and avoid reacting to very short-lived noise, we smooth the final trading signal in two steps with exponentially weighted moving averages with decay parameters $\alpha_1 = 0.25$ and $\alpha_2 = 0.15$. We also use a band of width 0.08 where small changes do not change the position, and we require a minimum holding time of 8 minutes before reversing or closing positions.

B.5 Backtesting Results and Performance Evaluation

We test the strategy over a 17-day window on BTC/USDT (about 23,500 one-minute observations). Over this period, the strategy reaches a cumulative return of 8.39%, while a Buy-and-Hold benchmark gains 3.80%, so the excess return is $+4.59$ percentage points. On common risk measures, the strategy also improves on the benchmark. It has a Sharpe ratio of 0.380 versus 0.168 for Buy-and-Hold and a Calmar ratio of 166.06 versus 23.30. The maximum drawdown is smaller: -2.80% for the strategy versus -5.26% for Buy-and-Hold. The annualized volatility is slightly lower at 24.23% (Buy-and-Hold: 25.82%).

Trading activity is moderate for a high-frequency setting: there are 17 trades in total, or about 1.04 trades per day. The average holding time per trade is 16.07 hours, and the median holding time is 0.65 hours. The win rate is 64.7% (Buy-and-Hold: 58.8%), and the average daily return is 0.48% (Buy-and-Hold: 0.23%). Over the 17-day test window, the strategy achieves 8.39% cumulative return versus 3.80% for Buy-and-Hold, with lower volatility and max drawdown.

C Agentic Trading Projects

Table 2 shows that several open-source agent-based trading systems have attracted notable developer interest. The two multiagent frameworks TradingAgents and AI Hedge Fund have about 24,800 and 42,300 GitHub stars and 4,600 and 7,500 forks, compared with smaller projects such as QuantAgent (306 stars, 71 forks) and ContestTrade (465 stars, 124 forks). All six projects have recent commits in late 2025, so they are being maintained rather than left idle.

Projects with more community activity also tend to use more agent-style designs. TradingAgents, AI Hedge Fund, and ContestTrade all include multiagent analysis in their trading types and expose orchestration as a clear module; five of the six repositories provide some form of persistent memory. These higher-usage projects also cover several markets (equities, crypto, and derivatives) and are released as reusable frameworks or applications rather than single-use examples. Table 2 shows that

Table 2: Comparative Attributes of Financial Agent Projects. Key: ✓ means the attribute is present; ✗ means it is not.

Attribute	**Quant Agent [27]**	**Alpha Arena [2]**	**Trading Agents [26]**	**AI Hedge Fund [22]**	**Contest Trade [33]**	**Stock Agent [8]**
Repository Attributes						
GitHub Stars	306	549	24800	42300	465	402
GitHub Forks	71	126	4600	7500	124	89
Last Update	11/09/2025	10/20/2025	10/09/2025	10/11/2025	10/13/2025	11/02/2025
License	MIT	MIT	Apache-2.0	MIT	Apache-2.0	MIT
Project Specifications						
Markets	Equities, Forex, Crypto, Commodities	Crypto (Bitcoin, Ethereum)	Equities (Any Ticker)	Equities (Any Ticker)	Equities (CN, US Markets)	Equities (Any Ticker)
Trading Types	Technical Analysis, Pattern Recognition, Trend Analysis	Long Trading, Paper Trading	Multi-Agent Analysis, Fundamental, Technical, Sentiment	Multi-Agent Analysis, Valuation, Risk Management	Event-Driven, Multi-Agent Analysis, Factor Portfolio	LLM-Based Trading Simulation, Real-World Events
Agents	4	2-6	6	18	2+	1
Tech Stack	Python 3.10, conda, LangGraph	Python, LangChain, Streamlit	Python, LLM, ReAct	Python, Poetry, TypeScript, Ollama	Python, LLM, ReAct Loop	Python 3.9, conda
Project Type	Framework	Benchmark	Framework	Application	Framework	Application
Agent Capabilities						
Memory	✓	✗	✓	✓	✓	✓
Orchestration	✓	✗	✓	✓	✓	✗

several open-source projects with higher community activity already adopt multi-agent designs and some form of persistent memory.

D Prompt Design, Context Protocols, and Memory Integration

Data Agent Stocks
Prompt:
You are the Data Agent for US equities. Your task is to fetch, clean, align, and normalize training-window market data from approved providers (Polygon, yfinance). You may not infer or reference any future information beyond the requested end date. You may not compute returns on the evaluation window. Return only cleaned features, metadata, and data-quality diagnostics.

Context:
role=DataAgent, assetType=equity, frequency="1h", lookbackDays=180

Input:
"symbols":["AAPL","MSFT","GOOGL",...],
"sourceHints":["polygon","yfinance"],
"requiredFields":["open","high","low","close","volume"]

Output (JSON):

```
{
  "cleanedDataRef": "s3://.../stocks_1h_clean.csv",
  "featureSummary": {
    "missingPct": 0.02, "outlierEvents": 14, "alignedCalendars": true
  },
  "qualityFlags": {
  "latencyChecked": true,
  ...
  }
}
```

Data Agent BTC
Prompt:
You are the Data Agent for BTC intraday data. Fetch, clean, deduplicate, and timestamp-align minute-level bars. You may not reference future data. Return training-window features only; do not compute P&L or returns beyond the provided lookback. BTC data typically comes from Polygon or Binance public APIs.

Context:
role=DataAgent, assetType=crypto, frequency="1m", lookbackDays=30

Input:
"symbols":["BTCUSDT"],
"sourceHints":["polygon","binance"],
"requiredFields":["open","high","low","close","volume"]

Output (JSON):

```
{
  "cleanedDataRef": "s3://.../btc_1m_clean.csv",
  "featureSummary": {
    "missingPct": 0.004, "dupTicksRemoved": 182, "timestampAligned": true
  },
  "qualityFlags": {
    "latencyChecked": true,
    ...
  }
}
```

Alpha Agent Stocks (Momentum)
Prompt:
You are the Alpha Agent for a daily US equity momentum strategy. Propose factor structures and signal formulations using only training-window feature summaries and published academic or industry priors. Do not use any evaluation-window data. Do not tune parameters based on Sharpe, IC, t-statistics, or any performance metric from the evaluation horizon. All numerical evaluation and backtesting are delegated to tools. Provide factor intuition, mathematical form, and expected direction only.

Context:
role=AlphaAgent, assetType=equity, horizonDays=5, family="momentum"

Input:
"featureSummary":"returns_20d":"std":0.18,
"returns_60d":"std":0.30,
"universe":["AAPL","MSFT","AMZN", ...],
"regimeHint":"calm"

Output (JSON):

```
{
  "factors": [
    {
      "name": "mom20_over_60",
      "family": "momentum",
      "type": "time_series",
      "horizonDays": 5,
      "lookbackDays": [20, 60],
      "formula": "returns_20d - returns_60d",
      "requiredInputs": ["returns_20d", "returns_60d"],
      "expectedDirection": "positive"
    },
    {
      "name": "cross_sectional_momentum",
      "family": "momentum",
      "type": "cross_sectional",
      "horizonDays": 5,
      "lookbackDays": [120],
      "formula": "rank(returns_120d)",
      "requiredInputs": ["returns_120d"],
      "expectedDirection": "positive"
    }
  ],
  "notes": {
    "usesTrainingWindowOnly": true,
    "noSharpeTuning": true,
    "leakageChecked": true
  }
}
```

Alpha Agent BTC (Microstructure)
Prompt:
You are the Alpha Agent for BTCUSD intraday trading. Propose microstructure-style factors (order-flow imbalance, realized volatility, spread, volume spikes) using only training-window summaries and tool-provided statistics. Do not use test labels, realized P&L, or backtest metrics. Do not search for the best Sharpe factor. Output factor definitions and expected direction only.

Context:
role=AlphaAgent, assetType=crypto, horizonMinutes=30,
family="microstructure"

Input (example):

```
featureSummary: {
  "ret_5m": {"std": 0.45},
  "ret_30m": {"std": 0.80},
  "of_imbalance_1m": {"mean": 0.02},
  "spread_bp_1m": {"mean": 3.1},
  ...
}
regimeHint: "highVol"
```

Output (JSON example):

```
{
  "factors": [
    {
      "name": "short_term_momentum_5m",
      "family": "momentum",
      "type": "time_series",
      "horizonMinutes": 15,
      "lookbackMinutes": 5,
      "formula": "ret_5m",
      "requiredInputs": ["ret_5m"],
      "expectedDirection": "positive"
    },
    {
      "name": "order_flow_imbalance_5m",
      "family": "microstructure",
      "type": "time_series",
      "horizonMinutes": 15,
      "lookbackMinutes": 5,
      "formula": "buy_volume_5m - sell_volume_5m",
      "requiredInputs": ["buy_volume_5m", "sell_volume_5m"],
      "expectedDirection": "positive"
    }
  ],
  "notes": {
    "usesTrainingWindowOnly": true,
    "noSharpeTuning": true
  }
}
```

Risk Agent Stocks
Prompt:
You are the Risk Agent for an equity portfolio. Interpret exposures and volatility signals using tool-provided covariance matrices and factor models. You may not access evaluation-window returns or future prices. You may not modify alpha signals directly; only produce risk reports and binary gates.

Context:
role=RiskAgent, assetType=equity, targetVol=0.15, maxLeverage=1.5

Input:
"alphaRef":"s3://.../us_alpha.parquet",
"sigmaRef":"s3://.../us_Sigma.parquet",
"exposureRef":"s3://.../us_B.parquet",
"constraints":"sectorLimit":0.30,"singleNameLimit":0.05

Output (JSON):

```
{
  "riskReport": {
    "forecastVol": 0.13,
    "grossLeverage": 1.20,
    "netBeta": 0.02,
    "sectorBreaches": []
  },
  "gates": {
    "vol_ok": true,
    "beta_ok": true,
    "sector_ok": true,
    "leakageChecked": true
  }
}
```

Risk Agent BTC
Prompt:
You are the Risk Agent for BTC intraday strategies. Using realized-vol and drawdown estimates from tools, check whether the proposed exposure satisfies volatility, position-size, and intraday drawdown limits. You may not compute P&L or see evaluation-window returns.

Context:
role=RiskAgent, assetType=crypto, targetVol=0.80, maxPositionPct=0.8

Input:
"positionPct":0.60,
"realizedVolEstimate":0.70,
"intradayDrawdownEstimate":0.18

Output (JSON):

```
{
  "riskReport": {
    "realizedVolEstimate": 0.70,
    "intradayDrawdownEstimate": 0.18
  },
  "gates": {
    "vol_ok": true,
    "dd_ok": true,
    "position_ok": true,
    "leakageChecked": true
  }
}
```

Portfolio Agent Stocks
Prompt:
You are the Portfolio Agent for US equities. Combine admissible alpha signals and risk diagnostics into portfolio weights under the constraints provided by the orchestration plan. Do not perform backtests or access realized returns. All numerical optimization is delegated to tools (e.g., convex solvers). You only specify the optimization problem and return the resulting weight vector and metadata.

Context:
role=PortfolioAgent, assetType=equity, turnoverLimit=0.20, tcModel="linear"

Input:
"alphaRef":"s3://.../us_alpha.parquet",
"riskGates":"vol_ok":true,"beta_ok":true,
"capital":100000

Output (JSON):

```
{
  "weights": {
    "AAPL": 0.12,
    "MSFT": 0.10,
    "GOOGL": 0.08,
    "others": "..."
  },
  "exposure": {
    "gross": 1.05,
    "net": 0.15
  },
  "notes": {
  "turnoverRespected": true, "riskGatesPassed": true
  }
}
```

Portfolio Agent BTC
Prompt:
You are the Portfolio Agent for BTC intraday trading. Adjust BTC exposure based on admissible microstructure signals and risk gates. No returns or P&L may be accessed. Output a single BTC weight or position percentage.

Context:
role=PortfolioAgent, assetType=crypto, turnoverLimit=0.50

Input:
"signalRef":"s3://.../btc_signal.parquet",
"riskGates":"vol_ok":true,"dd_ok":true,
"capital":100000

Output (JSON):

```
{
  "positionPct": 0.55,
  "exposure": {
    "gross": 0.55,
    "net": 0.55
  },
  "notes": {
    "turnoverRespected": true, "riskGatesPassed": true, ...
  }
}
```

Backtest Agent Stocks
Prompt:
You are the Backtest and Evaluation Agent for a daily equity strategy. Using stored signals and executed orders, compute portfolio-level metrics over the evaluation window. You are the only component that touches evaluation-window returns. Do not expose per-timestamp labels or raw P&L to any LLM agent; return only aggregated metrics and diagnostics.

Context:
role=BacktestAgent, assetType=equity, evalWindow="2024-05-01:2024-12-31"

Input:
"ordersRef":"s3://.../orders.parquet",
"priceRef":"s3://.../prices.parquet",
"metrics":["vol","sharpe","maxdd","turnover"]

Output (JSON):
```
{
  "metrics": {
    "vol": 0.1183,
    "sharpe": 2.63,
    "maxDrawdown": -0.0359,
    "turnover": 0.21
  },
  "notes": {
    "noTimestampPnL": true,
    "leakageChecked": true
  }
}
```

Backtest Agent BTC
Prompt:
You are the Backtest Agent for BTC minute-level strategies. Evaluate the strategy over the given intraday evaluation window. No timestamp-level P&L or trade-by-trade returns may be exposed; only aggregated metrics are allowed.

Context:
role=BacktestAgent, assetType=crypto, evalWindow="2025-07-27:2025-08-13"

Input:
"ordersRef":"s3://.../btc_orders.parquet",
"priceRef":"s3://.../btc_prices.parquet",
"metrics":["vol","sharpe","maxdd"]

Output (JSON):
```
{
  "metrics": {
    "vol": 0.412,
    "sharpe": 0.174,
    "maxDrawdown": -0.0091
  },
  "notes": {
    "noTimestampPnL": true,
    "leakageChecked": true
  }
}
```

E Context Protocols

The orchestration system uses structured context messages to pass information between agents. All contexts are serialized as JSON and follow the schema

$$C = \{\text{task\_id}, \text{agent\_role}, \text{run\_mode}, \text{time\_window}, \text{universe}, \text{inputs}, \text{tool\_outputs}, \text{diagnostics}, \text{uuid}\}. \tag{1}$$

Here `run_mode` $\in \{\text{train}, \text{test}, \text{live}\}$ and `time_window` records the lookback and horizon for the current step.

Each context message excludes:

- any raw price or return series from the test period;
- any labels or targets from future timestamps;
- any optimization objective that is tied directly to the evaluation window (for example, Sharpe ratio on the test set).

Numerical arrays are not sent directly. Instead, they are stored in data files or tables and referred to by identifiers (for example, data paths or dataset IDs). Only the Backtest Agent is allowed to load data that belong to the evaluation window. Contexts are written through the Memory Agent together with their `uuid` so that runs can be reproduced, checked, and replayed later.

F Memory Integration and UUID Protocols

The Memory Agent stores long-term states indexed by deterministic UUIDs. Each UUID is computed as a cryptographic hash of the agent role, task description, parameter configuration, and timestamp:

$$\text{UUID} = \text{SHA256}(\text{role}\|\text{task}\|\text{params}\|\text{time}). \tag{2}$$

This connects each memory record to a specific prompt and run setup, which helps with reproducibility and safety.

The UUID design supports:

- *immutability*: once written, memory entries are referred to only by their hash ID;
- *identity matching*: runs with the same configuration can look up compatible past states across orchestration cycles;
- *safe retrieval*: downstream agents query by UUID and receive only summarized metadata, not raw test-set values;
- *isolation*: training and evaluation memories use separate UUID namespaces to avoid mixing information.

A typical stored memory entry has the form

$$M = \{\text{uuid}, \text{agent\_role}, \text{plan\_step}, \text{features\_hash}, \text{metrics\_summary}, \text{timestamp}\}, \tag{3}$$

where `features_hash` is a non-invertible checksum of the input data and `metrics_summary` contains only aggregated statistics (for example, average IC by bucket or gate pass rates). Memory entries never store raw prices, raw returns, or full P&L paths, so they cannot be used to reconstruct evaluation-period labels.

G Leakage Prevention Summary

Across all agents, prompts and context protocols enforce a clear separation between LLM-based reasoning and numerical computation. LLM agents never receive evaluation-window returns, prices, or labels. Optimization and backtesting are implemented as deterministic tools behind the orchestration layer, and their outputs are filtered before any feedback is used in LLM prompts. UUID-based memory records make it easy to tell which run a record belongs to and to repeat the same run later, while storing only simple summaries that cannot be turned back into raw test data. These design choices lower the chance of data leakage and help keep our agentic trading experiments valid under walk-forward evaluation.