ACE-TRADER: ADVERSARIAL CO-EVOLUTION FOR GENERATOR–CRITIC TRADING SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Generator–critic trading systems pair an LLM-based trader that makes trading decisions with a red team agent that evaluates these decisions and provides feedback for revision. However, improving individual trading decisions does not ensure that both agents learn from experience and improve together over time. This raises an important question: how can the trader and red-Team select useful experience and learn from each other's evolving behavior? On one hand, experience that benefits decisions under one market condition may become ineffective or harmful under another. On the other hand, trading outcomes reflect both agents' actions, making it difficult to derive appropriate experience for each role. In this work, we propose ACE-Trader, an adversarial co-evolution trading framework. Specifically, we first develop separate context-dependent memory selectors that learn to identify useful experience for the current trading decision. Both selectors receive shared feedback from net-return differences between complete memory-assisted and memory-free trading pipelines, with online updates emphasizing recent outcomes. We further design a co-evolution mechanism that links role-specific memories through shared trading records. We use these records to evaluate the Red-Team's risk assessments and the effects of the Trader's revisions against realized market outcomes, then derive experience for their respective memories. By linking this experience to the corresponding records, we enable the Trader to improve its proposals based on prior critiques and the Red-Team to adjust its review focus as the Trader's behavior changes. Experiments on U.S. stock trading show that ACE-Trader outperforms all 22 active baselines in cumulative return and achieves the highest Sharpe ratio and lowest maximum drawdown across both active methods and passive benchmarks. Code is available at `https://anonymous.4open.science/r/ACE-Trader-B1BC` for anonymous review.

1 INTRODUCTION

Recent advances in large language models (LLMs) (Brown et al., 2020; Ouyang et al., 2022) and agentic systems (Yao et al., 2023; Wu et al., 2024) have spurred growing interest in building autonomous agents for financial trading. Existing LLM-based trading agents analyze market information and make trading decisions based on their analyses (Yu et al., 2025; 2024; Zhang et al., 2024; Xiao et al., 2024). To improve subsequent trading decisions, reflection-based trading agent (Yu et al., 2025; Zhang et al., 2024) reviews past decisions and their realized outcomes to identify useful lessons, which they store in memory to guide subsequent decisions, as shown in Fig. 1(a) . However, self-reflection may overlook weaknesses in the assumptions underlying the agent's own decisions(Liang et al., 2024). Generator–critic systems address this limitation by assigning decision-making and critique to different agents (Gong et al., 2025). In financial trading, a trader agent serves as the generator, proposing trading decisions and their rationales, while a red-team agent serves as the critic, evaluating these proposals and providing feedback for revision before execution (Xiao et al., 2024), as shown in Fig. 1(b). Improving this collaboration over time requires both agents to learn from their trading interactions and use the resulting experience to guide subsequent decisions.

However, reflecting on the overall trading outcome can lead agents to store misleading lessons in their respective memories: a profitable trade may conceal an inappropriate revision by the trader or an incorrect critique by the red-team, while a losing trade may obscure a useful red-team warn-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

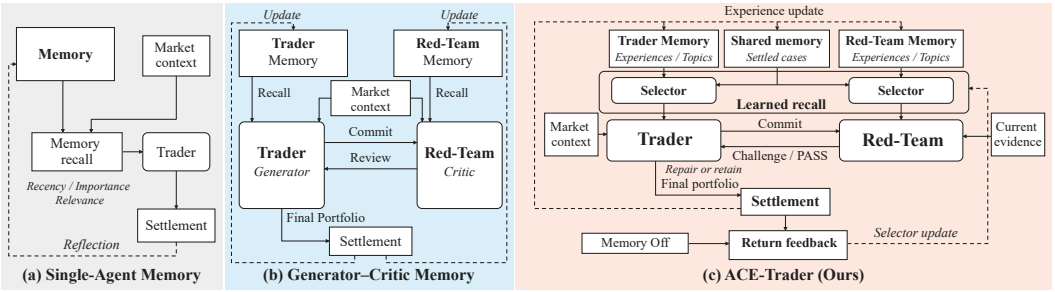


Figure 1: **Comparison of memory mechanisms in trading agents.**

ing.(Foerster et al., 2018) Even when appropriate lessons are extracted, they may not benefit subsequent decisions under different market conditions(Ang & Bekaert, 2002; Zhao et al., 2023). Yet common memory retrieval mechanisms select experience based on recency, importance, or semantic relevance (Yu et al., 2025; Zhang et al., 2024), without directly assessing its usefulness for the current trading decision. For example, recalling a lesson about reducing exposure during a downturn may lead an agent to remain overly cautious during a market recovery. These limitations raise a central question: *how can a trader and a red-team learn appropriate lessons for their respective roles and select experience that benefits the current decision?* The problem is non-trivial due to two key challenges.

First, *how can Trader and Red-Team adapt memory selection to changing market conditions?* The usefulness of past experience depends on the current market state. For the Trader, buying after a price decline may be useful in a range-bound market but lead to further losses during a sustained downtrend. For the Red-Team, a warning to reduce exposure during high volatility may prompt unnecessary position reductions once volatility subsides. Selecting useful experience therefore requires assessing its value in the current context, which cannot be determined from its past success alone. We propose separate lightweight, context-dependent memory selectors for the Trader and the Red-Team that learn which experience benefits the current trading decision. Each selector combines a frozen semantic encoder with an ensemble of trainable feedforward networks to score candidate memories using the current decision context, including market conditions, together with memory content and recorded outcomes. The highest-scoring candidate is selected for the corresponding agent. Both selectors receive shared feedback from the difference in net returns between the complete memory-assisted and memory-free trading pipelines. They are updated separately after trading outcomes are realized, with greater weight assigned to recent outcomes to adapt selection to changing market conditions.

Second, *how can Trader and Red-Team learn from each other's evolving behavior rather than update their experience in isolation?* While memory selection determines which past experience guides the current trading decision, the Trader and Red-Team must also learn from the proposal, critique, and subsequent revision to improve future decisions. For example, a Red-Team warning may teach the Trader to avoid a previously overlooked concentration risk in later portfolios. Once the Trader corrects this behavior, the Red-Team should adjust its review focus rather than repeat warnings about a problem that has already been addressed. Such adaptation requires each agent to learn not only from market outcomes but also from the other agent's actions and responses. However, reflecting separately may lead the Trader to overlook how the Red-Team's warning shaped its revision, and the Red-Team to judge its warning without considering how the Trader responded. In this work, we propose a memory co-evolution mechanism that links the Trader's and Red-Team's role-specific experience through shared memory, enabling each agent to adapt to the other's evolving behavior. We design a shared case memory to record the Trader's original proposal, the Red-Team's critique, the Trader's response, and the realized market outcome. To guide role-specific memory updates, we use these records to evaluate the Red-Team's risk assessments and the effect of the Trader's revisions on portfolio returns. Based on these evaluations, we derive experience about portfolio construction and revision for the Trader and risk assessment for the Red-Team. We store this experience in their respective memories and link it to the corresponding shared memory, so each agent can revisit how the other acted and responded. We further consolidate recurring experience into long-term memory with explicit applicability conditions. When selected for subsequent decisions, this experience helps

the Trader improve its proposals in response to prior critiques and the Red-Team adjust its review focus as the Trader's behavior changes.

In summary, our contributions are fourfold. First, we propose **ACE-Trader**, a framework for **A**dversarial **C**o-**E**volution. To the best of our knowledge, this is the first framework to explicitly support co-evolution between a Trader and a Red-Team through shared memory adaptation for financial trading. Second, we develop lightweight, context-dependent memory selectors trained with shared net-return feedback from complete memory-assisted and memory-free trading pipelines. Third, we design a co-evolution mechanism that links the Trader's and Red-Team's role-specific memories through shared trading cases, enabling each agent to learn from and adapt to the other's evolving behavior. Fourth, we evaluate ACE-Trader on U.S. stock trading and use ablation studies, hyperparameter analysis, and case studies to examine its performance and the roles of its key components.

2 Related Work

AI methods for trading. Traditional machine learning methods, including deep learning, use numerical market data to forecast returns and construct portfolios (Heaton et al., 2017; Gu et al., 2020; Ozbayoglu et al., 2020). LLM-based single-agent systems combine financial tools and language-based reasoning to analyze market information and generate trading decisions (Yu et al., 2025; Zhang et al., 2024). Generator–critic systems separate proposal generation from evaluation, using feedback from a critic to guide revisions(Gong et al., 2025). In financial trading, review and debate mechanisms similarly allow proposed decisions to be assessed and revised before execution (Yu et al., 2024; Xiao et al., 2024). However, revising individual proposals does not by itself ensure that the agents learn from past interactions and improve together over time.

Memory-based adaptation for trading. Memory-augmented trading agents retain experience from past decisions to guide later decisions (Yu et al., 2025; Zhang et al., 2024). They organize this experience into layered or diversified memories and retrieve it according to recency, importance, or relevance. Recently, some methods further reflect on observed trading outcomes to revise investment beliefs and risk preferences (Yu et al., 2024). However, retrieving a recent or relevant memory does not establish whether it will benefit the current decision under changing market conditions. Moreover, independent reflections can lose the connection between one agent's feedback and the other's response, making it difficult for the agents to learn from each other's evolving behavior. ACE-Trader combines context-dependent memory selection with shared trading records that connect the Trader's and Red-Team's experience.

3 Preliminaries

We consider daily portfolio trading over a universe of N assets. At decision step t, the input consists of market information o_t, including historical prices, news, and technical indicators, and account state b_t, including cash, existing holdings, net asset value, and portfolio exposure. The output is a portfolio plan a_t that specifies `BUY`, `SELL`, or `HOLD` decisions for selected assets, together with target portfolio weights expressed as percentages of the account's net asset value after transaction costs. The total absolute exposure across long and short positions is limited to 100% of net asset value. All decisions and financial tool queries use only information available at the decision cutoff.

After executing a_t, we evaluate the realized net return r_t over the settlement interval as the relative change in account value after transaction costs (Liu et al., 2021). Transaction costs apply to the actual amounts traded when opening, adjusting, closing, or reversing positions. Cash, holdings, and account value are carried forward without daily resets. Each trading episode consists of one decision–execution–settlement cycle within this continuous account trajectory. The objective is to improve cumulative net return subject to the portfolio constraints.

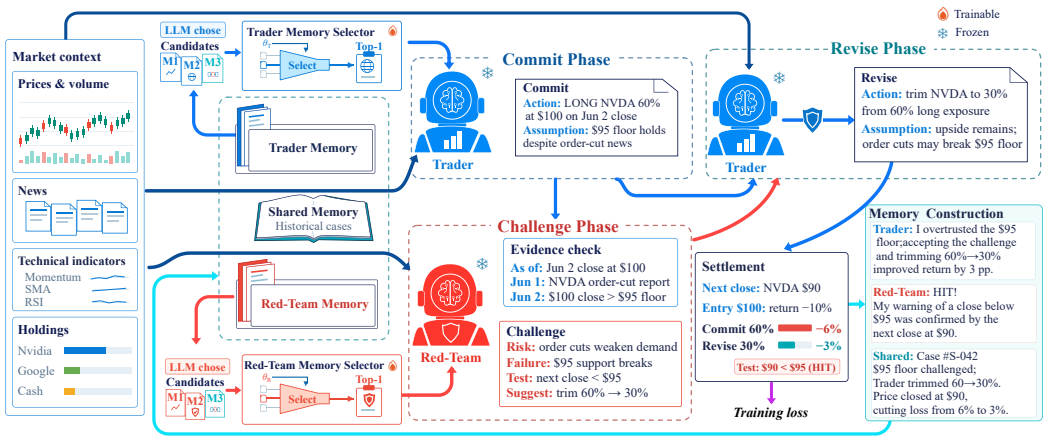


Figure 2: Overview of ACE-Trader.

4 Proposed Approach

4.1 Overview

We illustrate the overall framework of ACE-Trader in Figure 2. ACE-Trader consists of two main modules: *memory selection and memory-guided trading*, and *memory co-evolution*. For memory-guided trading, we use separate selectors to choose experience for the Trader and Red-Team based on the current decision context. With memory guidance, the Trader proposes a portfolio, the Red-Team reviews the proposal using current evidence, and the Trader revises the portfolio in response. For memory co-evolution, we use the realized outcomes to evaluate whether the Red-Team's predicted risks occurred and whether the Trader's revisions improved net return over the original proposal. We then construct role-specific memories and shared memories which link them to shared interaction records, allowing each agent to learn from the other's actions and responses.

4.2 Memory Selection and Memory-Guided Trading

At trading step t, the Trader and Red-Team receive market information o_t and account state b_t. Each agent maintains private memory containing its role-specific experience and can access shared memory containing past trading interactions. We first describe how each agent selects memory, then explain how the memory guides the Commit–Challenge–Revise workflow.

4.2.1 Selecting Useful Memories

Recalling candidates. Before the Trader commits a portfolio and before the Red-Team reviews it, each role recalls candidates for its current decision context. We retrieve candidate memories separately for each phase $u \in \{\mathrm{Com}, \mathrm{Cha}, \mathrm{Rev}\}$, corresponding to commit, challenge, and revise phase. Retrieval uses the current market and account information, together with any proposal, feedback, or tool evidence already available in that phase. Candidates come from the corresponding agent's private memory and the shared memory. We denote the resulting candidate set by $\mathcal{M}_t^u$.

Encoding candidate representations. For each candidate $m \in \mathcal{M}_t^u$, we construct semantic features $\mathbf{e}_{t,m}^u$ from its text and the phase-specific retrieval context using a frozen semantic encoder E and a linear projection, as detailed in Appendix E.3. We concatenate these features with market and portfolio features and the memory's recorded outcomes:

$$\mathbf{v}_{t,m}^u = \mathrm{Concat}\left(\mathbf{e}_{t,m}^u, \mathbf{f}_t^u, \mathbf{h}_m\right). \tag{1}$$

Here, $\mathbf{f}_t^u$ contains current market features from o_t and portfolio features specific to phase u. In Commit, the portfolio features describe the actual pre-trade holdings in b_t. In Challenge and Revise, they describe the portfolio that would result from applying the proposed trading actions to b_t, without executing those actions. The vector $\mathbf{h}_m$ contains the historical outcomes recorded in memory m.

Scoring and selecting memories. The Trader and Red-Team use separate selectors, each comprising K feedforward networks. The Trader shares its selector across Commit and Revise, but retrieves and selects memory separately in each phase. Each network scores candidate m using $\mathbf{v}^u_{t,m}$, and we combine the outputs as

$$s^u_{t,m} = \mu^u_{t,m} + \alpha\sigma^u_{t,m}, \tag{2}$$

where $\mu^u_{t,m}$ and $\sigma^u_{t,m}$ are the mean and standard deviation of the network outputs, and $\alpha \geq 0$ is a hyperparameter controlling the disagreement bonus. We rank candidates by $s^u_{t,m}$ and supply the highest-scoring candidate as m^u_t. If the candidate set $\mathcal{M}^u_t$ is empty, no memory is supplied and $m^u_t = \varnothing$.

4.2.2 Memory-Guided Adversarial Trading

For each phase u, we construct a prompt P^u_t containing the phase-specific instructions and available information. The operators Trader and $\mathrm{RedTeam}$ each take a prompt as input and produce their outputs through the corresponding tool-assisted workflow.

Trader commit. We construct P^{Com}_t from market information o_t, account state b_t, and the historical observations in memory m^{Com}_t. The Trader uses this prompt to produce an initial trading plan and its rationale:

$$\left(a^{\mathrm{Com}}_t, d^{\mathrm{Com}}_t\right) = \mathrm{Trader}\left(P^{\mathrm{Com}}_t\right). \tag{3}$$

Here, a^{Com}_t specifies the proposed trading actions and target portfolio weights, and d^{Com}_t states the supporting reasons and assumptions. For each assumption, d^{Com}_t records an observable condition that would invalidate it and the time horizon for checking that condition. Both are passed to the Red-Team.

Evidence-based challenge. The Red-Team selects memory m^{Cha}_t based on the current proposal and market information. We construct P^{Cha}_t from o_t, b_t, the proposal a^{Com}_t, its rationale d^{Com}_t, and memory m^{Cha}_t, supplemented with linked case details when applicable. The Red-Team examines the proposal's assumptions and risks using current evidence gathered through news search and market-data tools:

$$c_t = \mathrm{RedTeam}\left(P^{\mathrm{Cha}}_t\right). \tag{4}$$

Here, c_t contains the review evidence and either a challenge or `PASS`. The Red-Team issues a challenge only when current evidence contradicts a specific assumption in the Trader's proposal. Each challenge includes suggested portfolio adjustments and a risk warning specifying an observable condition and when it should be evaluated. When the evidence is insufficient, the Red-Team returns `PASS` with any monitoring suggestions.

Trader revise. The Trader retrieves and selects memory again using the original proposal, market information o_t, and Red-Team feedback c_t. We construct P^{Rev}_t from o_t, b_t, a^{Com}_t, d^{Com}_t, c_t, and the newly selected memory m^{Rev}_t. The Trader uses this prompt to produce its final plan and response:

$$\left(a^{\mathrm{Rev}}_t, d^{\mathrm{Rev}}_t\right) = \mathrm{Trader}\left(P^{\mathrm{Rev}}_t\right). \tag{5}$$

Here, a^{Rev}_t is the final trading plan, and d^{Rev}_t explains the Trader's response. The Trader revises disputed positions or provides additional current evidence to retain them. If the Red-Team returns `PASS`, we skip Revise without further memory retrieval, selection, or a Trader call, and set $a^{\mathrm{Rev}}_t = a^{\mathrm{Com}}_t$. The final plan is $a_t = a^{\mathrm{Rev}}_t$.

4.3 Memory Co-Evolution

We use each trading interaction and its observed outcomes to construct experience for both roles. By evaluating the original proposal, the Red-Team's review, and the Trader's response together, we preserve how the agents influenced each other's decisions. The resulting role-specific memories are linked through shared cases for use in subsequent interactions.

Settlement and evaluation. We execute only the final plan a^{Rev}_t and carry forward the resulting cash and holdings. Once the settlement outcomes are observed, we compare its net return with the return that the original proposal a^{Com}_t would have produced. Both calculations start from the same account state b_t and use the same observed prices and transaction-cost rules. Their difference

indicates whether responding to the review improved net return, rather than merely whether the final plan was profitable.

We also check the risk warnings in c_t against the conditions and evaluation times specified before execution. A warning is a *hit* if the observed outcomes satisfy its stated condition under the specified evaluation rule. It is a *false alarm* if its evaluation horizon has ended and sufficient observations show that the condition was not satisfied. Warnings that cannot yet be assessed remain *unresolved*. For the assumptions in d_t^{Com}, we check whether their stated failure conditions are met within the corresponding horizons. An assumption that fails without a corresponding valid Red-Team challenge is recorded as a *miss*. Risk occurrence and revision benefit are evaluated separately: a correct warning can still be followed by a revision that reduces net return.

Constructing shared and role-specific memories. We store the complete interaction, risk labels, and return comparison as a shared memory case. Using predefined rules, we then extract a short-term memory item for each role. The Trader's item records its original and final trading decisions, its response to the review, and the net-return difference between the two plans. When the final plan underperforms the original proposal, the item explicitly records that the revision reduced net return, even if the final plan itself was profitable. The Red-Team's item records the challenged or missed assumptions, the evidence examined, and the corresponding risk labels. Both items include the market context and shared case identifier, preserving the connection between the proposal, critique, response, and outcome.

Each item is added to the corresponding role's private short-term memory. Periodically, an LLM summarizes recurring lessons from recent entries for long-term memory, specifying when each lesson applies and retaining references to its supporting cases. These lessons distinguish whether a risk occurred from whether the response improved net return. When a lesson is recalled, the agent can revisit the linked interaction to understand the other role's actions and responses without accessing its private memory.

4.4 Memory Selector Update

At each trading step t, we run the complete trading workflow with and without memory using the same market information o_t and pre-decision account state b_t. We simulate the execution of their final plans on copies of b_t over the same settlement interval, using identical market prices and transaction-cost rules. Once the outcomes are observed, the net-return difference provides a shared binary label, with near-tied outcomes excluded. Each selector is trained on the saved representations of its selected memories using recency-weighted binary cross-entropy. The trading LLMs and semantic encoder remain frozen. Details of supervision and the training objective are provided in Appendix B.

5 Experiments

We evaluate ACE-Trader on daily U.S. stock trading to examine portfolio performance and the contribution of memory-based adaptation under a continuous-account protocol.

5.1 Experimental Setup

Dataset. We evaluate ACE-Trader on a daily U.S. stock-trading benchmark comprising 33 equities across 11 sectors. Historical prices and financial news are drawn from FNSPID (Dong et al., 2024), which contains 29.7 million stock-price records and 15.7 million time-aligned news articles. Data from 2021-01-01 to 2023-12-31 are used for task-specific training of baselines that require it and for warming up ACE-Trader, without a separate validation set. Testing covers 2026-01-01 to 2026-06-30, using prices, news, and associated asset information collected from Financial Modeling Prep (FMP) for the same stock universe. The test period also falls after the documented training-data cutoffs of all evaluated LLM checkpoints. Data construction and temporal filtering are detailed in Appendix C.1, C.2.

Baselines. We compare ACE-Trader with 22 active baselines organized into six categories. (1) *Machine Learning methods (ML)*: LightGBM (Ke et al., 2017), XGBoost (Chen & Guestrin, 2016),

Table 1: Portfolio performance on Stock.

Category	Model	Cum. Ret. (%)↑	Ann. Ret. (%)↑	Sharpe↑	MDD. (%)↓	Vol. (%)↓	HR. (%)↑	Turnover
Passive	NASDAQ Buy & Hold	**+12.82**	**+28.29**	1.340	12.84	20.10	**57.38**	0.000
	Equal-Weight Buy & Hold	+8.68	+18.77	1.375	5.33	13.14	52.48	0.000
ML	LightGBM	+4.86	+10.30	0.502	13.13	26.48	50.74	1.341
	XGBoost	-21.18	-38.84	-1.610	29.53	28.07	50.16	1.329
	LSTM	+2.81	+5.90	0.399	9.25	18.60	47.52	1.044
	GRU	+0.53	+1.09	0.148	11.16	17.42	49.59	1.028
	MLP	-1.64	-3.36	-0.124	10.74	16.56	51.07	1.046
	TRA	+6.47	+13.83	0.662	16.23	23.83	51.31	0.643
	HIST	+5.29	+11.24	0.619	8.78	20.65	51.48	0.895
RL	FinRL-DQN	+9.59	+20.82	1.067	15.11	19.50	49.34	0.821
	FinRL-A2C	-16.84	-31.67	-2.245	18.88	16.35	47.45	0.915
	FinRL-PPO	-7.40	-14.68	-0.820	12.38	17.51	49.10	1.074
General LLMs	GLM 5.3 Flash	-6.30	-12.58	-0.453	13.26	23.60	48.60	1.362
	DeepSeek V4 Flash	-19.72	-36.47	-1.387	25.05	29.57	50.41	1.287
	Qwen3.6-35B-A3B	-13.18	-25.32	-1.046	16.81	24.96	50.33	1.393
Financial LLMs	FinGPT	-10.84	-21.10	-1.003	18.45	21.38	49.26	1.514
	FinCast	-1.30	-2.66	0.040	14.68	27.72	54.10	1.636
	Kronos	-30.45	-52.76	-3.753	32.65	19.45	46.55	1.936
Single-agent	FinAgent	-15.95	-30.16	-1.495	24.29	22.34	49.43	1.316
	FLAG-Trader	-2.62	-5.33	-0.159	15.93	20.86	50.00	1.370
Multi-agents	TradingAgents	-9.42	-18.49	-0.769	19.88	23.11	50.41	1.343
	LLMoE	-7.33	-14.55	-0.795	13.22	17.80	47.95	1.337
	TradExpert	+0.10	+0.20	0.072	6.55	**10.70**	48.43	0.610
	MM-DREX	-5.27	-10.58	-0.615	13.59	16.10	48.11	1.350
Ours	**ACE-Trader**	+12.44	+27.40	**2.023**	**5.21**	12.35	**57.38**	0.153

The best and second-best results across all methods, including passive benchmarks, are shown in **bold** and underlined, respectively. Turnover is reported without ranking.

LSTM (Hochreiter & Schmidhuber, 1997), GRU (Cho et al., 2014; Chung et al., 2014), MLP (Hornik et al., 1989), TRA (Lin et al., 2021), and HIST (Xu et al., 2021), which predict asset returns using structured market features, temporal representations, or cross-asset relationships. (2) *Reinforcement-learning trading methods (RL)*: FinRL-DQN, FinRL-A2C, and FinRL-PPO (Liu et al., 2021), which learn trading policies through interactions with a portfolio environment. (3) *Frontier general-purpose LLMs*: GLM 5.3 Flash, DeepSeek V4 Flash (Xu et al., 2026), and Qwen3.6-35B-A3B, which directly generate trading decisions without task-specific fine-tuning. (4) *Finance-specific LLMs*: FinGPT (Yang et al., 2023), FinCast (Zhu et al., 2025), and Kronos (Shi et al., 2026), which use financial-domain pretraining or adaptation to generate decisions or market forecasts. (5) *Single-agent trading systems*: FinAgent (Zhang et al., 2024) and FLAG-Trader (Xiong et al., 2025), which organize market analysis and decision making around one trading agent, supported by tools, memory, reflection, or policy optimization. (6) *Multi-agent trading systems*: TradingAgents (Xiao et al., 2024), LLMoE (Liu & Lo, 2025), TradExpert (Ding et al., 2024), and MM-DREX (Chen et al., 2025), which combine specialized assessments through collaboration, debate, or expert coordination.

We also report two passive references. *Market Buy-and-Hold* follows the Nasdaq Composite throughout the evaluation period. *Equal-Weight Buy-and-Hold* allocates capital equally across the candidate stocks at the beginning of the period and holds the resulting portfolio without active rebalancing. Further baseline descriptions are provided in Appendix D.

Evaluation protocol. All methods are evaluated chronologically using the same information cutoffs and continuous-account simulator. Trading decisions are made and executed at each trading day's opening, with a one-way transaction cost of 5 basis points on traded notional and a gross-exposure limit of 100% of net asset value. Cash and holdings persist across trading days. ACE-Trader produces `BUY`, `SELL`, or `HOLD` decisions with target weights; baseline outputs follow the conversions in Appendix C.3. Offline training uses only historical data, without test-period hyperparameter selection. Trading feedback is evaluated through the next common trading day's close and is used for memory and selector updates only after becoming observable. Portfolio-return metrics use the continuous account's daily net asset values after costs. Execution and feedback details are provided in Appendices C.4 and B.

Evaluation metrics. We report cumulative return (Markowitz, 1952), annualized return (Markowitz, 1952), Sharpe ratio (Sharpe, 1966), maximum drawdown(MDD) (Magdon-Ismail et al., 2004), an-

Table 2: Ablation study of memory selection and co-evolution.

Model	TS	RS	CE	Cum. Ret. (%)↑	Sharpe↑	MDD. (%)↓	Vol. (%)↓	HR. (%)↑
Variant 1	×	×	×	+5.53	0.823	7.81	14.84	53.38
Variant 2	✓	×	×	+6.90	1.028	7.52	14.41	58.20
Variant 3	×	✓	×	+8.56	1.261	6.33	14.25	62.30
Variant 4	✓	✓	×	+10.07	1.529	6.36	13.57	55.74
ACE-Trader	✓	✓	✓	+12.44	2.023	5.21	12.35	57.38

TS: Trader selector; RS: Red-Team selector; CE: Co-Evolution. ✓ and × indicate whether a component is included or excluded, respectively.

nualized volatility (Markowitz, 1952), hit rate(HR) (Pesaran & Timmermann, 1992), and average daily turnover (DeMiguel et al., 2009) All portfolio-return metrics are calculated from the continuous account trajectory after transaction costs. Their definitions are given in Appendix C.5.

Implementation details. Both the Trader and Red-Team use DeepSeek V4 Flash, with MiniLM encoding candidate memories. Each selector comprises $K = 3$ feedforward networks, and we set $\alpha = \gamma = 0.8$. The trading LLMs and semantic encoder remain frozen. Further implementation details for ACE-Trader and the baselines are provided in Appendix E.

5.2 Performance Evaluation

Table 1 reports the portfolio performance on Stock. We make three observations. First, ACE-Trader achieves the highest cumulative return among active methods, outperforming the strongest baseline, FinRL-DQN, by 2.85 percentage points. This result supports the effectiveness of the complete framework for portfolio trading. Second, ACE-Trader outperforms direct prompting with the same DeepSeek V4 Flash backbone, increasing cumulative return from -19.72% to 12.44%. This comparison supports the value of the memory-guided Trader–Red-Team workflow beyond using the backbone alone. Third, ACE-Trader achieves the highest Sharpe ratio and lowest maximum drawdown across all methods, including passive benchmarks. Although its cumulative return is slightly below NASDAQ Buy-and-Hold, its lower drawdown and volatility indicate a more favorable risk–return trade-off.

5.3 Ablation Study

To assess the contributions of context-dependent memory selection and memory co-evolution, we compare ACE-Trader with four variants in Table 2. All variants retain the Commit–Challenge–Revise workflow. Variants 1–4 disable memory co-evolution and use neither selector, only the Trader selector, only the Red-Team selector, or both selectors, respectively. The complete model additionally enables memory co-evolution. The remaining experimental settings are unchanged, and the implementation of each ablated variant is detailed in Appendix E.6.

We make three observations. First, the Trader selector improves profitability even without memory co-evolution. Variant 2 outperforms Variant 1 across all reported metrics, while Variant 4 achieves higher cumulative return and Sharpe ratio than Variant 3. These comparisons support the value of selecting relevant experience for portfolio construction and revision.

Second, the Red-Team selector improves risk-adjusted performance. Variant 3 outperforms Variant 1 across all reported metrics. Adding the Red-Team selector alongside the Trader selector also improves return and Sharpe ratio while reducing maximum drawdown and volatility, as shown by Variant 4 versus Variant 2. These results support the value of selecting relevant experience for the review process.

Third, enabling memory co-evolution improves all reported metrics relative to Variant 4, increasing cumulative return by 2.37 percentage points. This supports the contribution of learning from linked trading experience beyond memory selection alone. The complete model achieves the highest return and Sharpe ratio with the lowest maximum drawdown and volatility, although Variant 3 retains the highest hit rate.

5.4 Hyperparameter Analysis

We examine the effects of α in memory scoring and γ in selector training. When varying α, we fix $\gamma = 0.8$; when varying γ, we fix $\alpha = 0.8$. Figure 3 compares ACE-Trader (Memory On) with a memory-free baseline (Memory Off). The baseline runs the same Commit–Challenge–Revise workflow without recalled memories and carries forward its own cash and holdings throughout testing.

Impact of α. Cumulative return increases as α rises from 0 to 0.8, but drops sharply at 1.0. The improvement over $\alpha = 0$, which ranks memories solely by their mean predictions, supports the value of disagreement-guided exploration. However, the decline at $\alpha = 1.0$ suggests that placing too much weight on disagreement can be counterproductive. The highest return is achieved at $\alpha = 0.8$.

Impact of γ. Small values of γ yield limited returns. Increasing γ gives greater weight to older feedback within the fixed training window, with the highest return achieved at $\gamma = 0.8$. Performance declines at $\gamma = 1.0$, where all valid trading steps within the window receive equal weight. These results suggest that using older feedback while assigning greater weight to recent outcomes is more effective than focusing only on the latest feedback or weighting all valid trading steps equally.

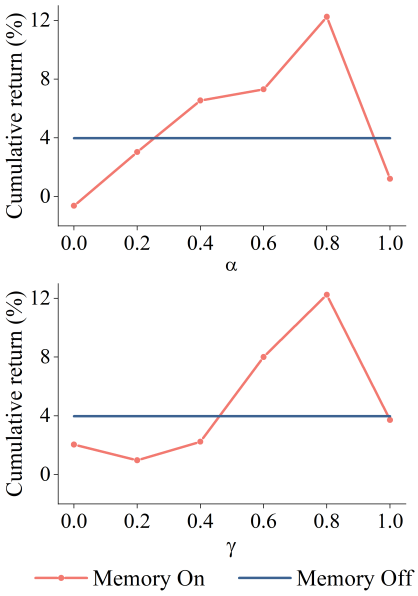


Figure 3: Impact of α and γ.

5.5 Case Study: Memory Selection

We examine the 2026-01-05 interaction, where three memory selections draw on role-specific records from the December 15, 2023 episode. During Commit, the Trader recalls a portfolio experience from a market with broadly positive momentum to support its JPM and BHP purchases without copying historical position sizes. During Challenge, the Red-Team selects the corresponding AVGO false-alarm record to assess whether a CAT downgrade is supported by price deterioration. During Revise, the Trader independently reselects its portfolio record and cites the earlier false alarm in its response. Given CAT's 2.96% rise and positive MACD (Appel, 1985), it judges the downgrade insufficient to justify reducing its existing shares.

By the January 6 close, CAT rises by 1.13%, so the warning of a decline of at least 2.5% is classified as a false alarm. The original and final plans remain identical, both returning 0.15%, with no incremental revision gain. This case illustrates how recalled experience supports retaining a proposal when current evidence does not justify a change. An additional case study in Appendix A illustrates a beneficial portfolio revision.

6 Conclusions

In this paper, we proposed ACE-Trader, an adversarial co-evolution framework for generator–critic trading systems. ACE-Trader employs separate context-dependent memory selectors to guide the Trader and Red-Team, learning from shared net-return feedback to adapt memory selection to changing market conditions. We further developed a memory co-evolution mechanism that separately evaluates whether predicted risks occur and whether portfolio revisions improve net return, then derives role-specific experience linked through shared trading cases. These links preserve the connection between one agent's feedback and the other's response, allowing both roles to learn from their interactions without fine-tuning the underlying LLMs. Experiments on U.S. stock trading show that ACE-Trader outperforms all 22 active baselines in cumulative return and achieves the highest Sharpe ratio and lowest maximum drawdown across both active methods and passive benchmarks.

AI USE STATEMENT

Generative AI tools were used to improve the readability and language of the manuscript and to assist with software code writing and editing. The research ideas, hypotheses, methodology, and experimental design were developed by the authors, and the interpretation of experimental results was conducted by the authors. All AI-assisted code and manuscript revisions were reviewed and verified by the authors. The authors take responsibility for the final content of this work, including all AI-assisted text and code.

REPRODUCIBILITY STATEMENT

We provide the data construction and evaluation protocol in Appendix C, the settlement feedback and online update procedure in Appendix B, and implementation and baseline configurations in Appendix E. Anonymous source code is provided with the submission to facilitate reproduction of the reported results.

REFERENCES

Andrew Ang and Geert Bekaert. International asset allocation with regime shifts. *The Review of Financial Studies*, 15(4):1137–1187, July 2002. doi: 10.1093/rfs/15.4.1137. URL https://doi.org/10.1093/rfs/15.4.1137.

Gerald Appel. *The moving average convergence-divergence trading method: advanced version*. Scientific Investment Systems, 1985.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 33:1877–1901, 2020.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *KDD*, pp. 785–794, 2016.

Yang Chen, Yueheng Jiang, Zhaozhao Ma, Yuchen Cao, Jacky Keung, Kun Kuang, Leilei Gan, Yiquan Wu, and Fei Wu. MM-DREX: Multimodal-driven dynamic routing of LLM experts for financial trading. *arXiv preprint arXiv:2509.05080*, 2025.

Kyunghyun Cho, Bart Van Merriënboer, Çağlar Gulçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *EMNLP*, pp. 1724–1734, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Victor DeMiguel, Lorenzo Garlappi, and Raman Uppal. Optimal versus naive diversification: How inefficient is the 1/n portfolio strategy? *The review of Financial studies*, 22(5):1915–1953, 2009.

Qianggang Ding, Haochen Shi, Jiadong Guo, and Bang Liu. Tradexpert: Revolutionizing trading with mixture of expert llms. *arXiv preprint arXiv:2411.00782*, 2024.

Zihan Dong, Xinyu Fan, and Zhiyuan Peng. Fnspid: A comprehensive financial news dataset in time series. In *KDD*, pp. 4918–4927, 2024.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI*, volume 32, 2018.

Nanxu Gong, Xinyuan Wang, Wangyang Ying, Haoyue Bai, Sixun Dong, Haifeng Chen, and Yanjie Fu. Unsupervised feature transformation via in-context generation, generator-critic llm agents, and duet-play teaming. In *IJCAI*, pp. 2820–2828, 2025. doi: 10.24963/ijcai.2025/314.

Shihao Gu, Bryan Kelly, and Dacheng Xiu. Empirical asset pricing via machine learning. *The Review of Financial Studies*, 33(5):2223–2273, 2020. doi: 10.1093/rfs/hhaa009. URL https://doi.org/10.1093/rfs/hhaa009.

James B. Heaton, Nick G. Polson, and Jan Hendrik Witte. Deep learning for finance: Deep portfolios. *Applied Stochastic Models in Business and Industry*, 33(1):3–12, 2017. doi: 10.1002/asmb.2209. URL https://doi.org/10.1002/asmb.2209.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *NeurIPS*, 30, 2017.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate. In *EMNLP*, pp. 17889–17904, 2024.

Hengxu Lin, Dong Zhou, Weiqing Liu, and Jiang Bian. Learning multiple stock trading patterns with temporal routing adaptor and optimal transport. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pp. 1017–1026, 2021.

Kuan-Ming Liu and Ming-Chih Lo. Llm-based routing in mixture of experts: A novel framework for trading. *arXiv preprint arXiv:2501.09636*, 2025.

Xiao-Yang Liu, Hongyang Yang, Jiechao Gao, and Christina Dan Wang. Finrl: Deep reinforcement learning framework to automate trading in quantitative finance. In *Proceedings of the second ACM international conference on AI in finance*, pp. 1–9, 2021.

Malik Magdon-Ismail, Amir F Atiya, Amrit Pratap, and Yaser S Abu-Mostafa. On the maximum drawdown of a brownian motion. *Journal of applied probability*, 41(1):147–161, 2004.

Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *NeurIPS*, 35:27730–27744, 2022.

Ahmet Murat Ozbayoglu, Mehmet Ugur Gudelek, and Omer Berat Sezer. Deep learning for financial applications: A survey. *Applied Soft Computing*, 93:106384, 2020. doi: 10.1016/j.asoc.2020.106384. URL https://doi.org/10.1016/j.asoc.2020.106384.

M Hashem Pesaran and Allan Timmermann. A simple nonparametric test of predictive performance. *Journal of Business & Economic Statistics*, 10(4):461–465, 1992.

William F Sharpe. Mutual fund performance. *The Journal of business*, 39(1):119–138, 1966.

Yu Shi, Zongliang Fu, Shuo Chen, Bohan Zhao, Wei Xu, Changshui Zhang, and Jian Li. Kronos: A foundation model for the language of financial markets. In *AAAI*, volume 40, pp. 25366–25373, 2026.

Qwen Team. Qwen3. 6-35b-a3b: Agentic coding power, now open to all, 2026.

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 conversation. *COLM*, 2024.

Yijia Xiao, Edward Sun, Di Luo, and Wei Wang. Tradingagents: Multi-agents llm financial trading framework. *arXiv preprint arXiv:2412.20138*, 2024.

Guojun Xiong, Zhiyang Deng, Keyi Wang, Yupeng Cao, Haohang Li, Yangyang Yu, Xueqing Peng, Mingquan Lin, Kaleb E Smith, Xiao-Yang Liu, et al. Flag-trader: Fusion llm-agent with gradient-based reinforcement learning for financial trading. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 13921–13934, 2025.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Wentao Xu, Weiqing Liu, Lewen Wang, Yingce Xia, Jiang Bian, Jian Yin, and Tie-Yan Liu. Hist: A graph-based framework for stock trend forecasting via mining concept-oriented shared information. *arXiv preprint arXiv:2110.13716*, 2021.

Hongyang Yang, Xiao-Yang Liu, and Christina Dan Wang. Fingpt: Open-source financial large language models. *arXiv preprint arXiv:2306.06031*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *ICLR*, 2023.

Yangyang Yu, Zhiyuan Yao, Haohang Li, Zhiyang Deng, Yuechen Jiang, Yupeng Cao, Zhi Chen, Jordan W. Suchow, Zhenyu Cui, Rong Liu, Zhaozhuo Xu, Denghui Zhang, Koduvayur Subbalakshmi, Guojun Xiong, Yueru He, Jimin Huang, Dong Li, and Qianqian Xie. FinCon: A synthesized LLM multi-agent system with conceptual verbal reinforcement for enhanced financial decision making. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 137010–137045, Red Hook, NY, USA, 2024. Curran Associates, Inc. doi: 10.52202/079017-4354. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/f7ae4fe91d96f50abc2211f09b6a7e49-Paper-Conference.pdf.

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*, 11(6):3443–3459, 2025. doi: 10.1109/TBDATA.2025.3593370. URL https://doi.org/10.1109/TBDATA.2025.3593370.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Wentao Zhang, Lingxuan Zhao, Haochong Xia, Shuo Sun, Jiaze Sun, Molei Qin, Xinyi Li, Yuqing Zhao, Yilei Zhao, Xinyu Cai, Longtao Zheng, Xinrun Wang, and Bo An. A multimodal foundation agent for financial trading: Tool-augmented, diversified, and generalist. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4314–4325. Association for Computing Machinery, 2024. doi: 10.1145/3637528.3671801.

Lifan Zhao, Shuming Kong, and Yanyan Shen. Doubleadapt: A meta-learning approach to incremental learning for stock trend forecasting. In *KDD*, pp. 3492–3503, 2023.

Zhuohang Zhu, Haodong Chen, Qiang Qu, and Vera Chung. Fincast: A foundation model for financial time-series forecasting. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pp. 4539–4549, 2025.

A Case Study: Portfolio Revision

Table 3 presents the 2026-06-17 interaction, illustrating how the Red-Team's review corrects a trading rationale before execution. The Trader treats AMZN's oversold RSI as a buying opportunity, but the Red-Team identifies a gap in this reasoning: the low RSI does not establish that the persistent downtrend is reversing. The review confronts the rebound thesis with recent price evidence and prompts a targeted correction. The Trader cancels only the additional AMZN purchase, retaining its existing shares and leaving the JPM and XOM sell orders unchanged.

AMZN's subsequent decline satisfies the pre-specified risk condition, confirming the warning as a hit. In the five-trading-day fixed-holdings comparison, canceling the purchase increases net return from -0.0148% to +0.0745%, turning a small evaluated loss into a gain. The improvement comes from avoiding the loss on the additional AMZN exposure and its transaction cost, rather than changing unrelated trades. The case therefore shows how evidence-based review translates a valid risk

Table 3: Case from the initial proposal to the revised decision.

Phase	Decision, evidence, and outcome
Trader: Commit	**Proposal:** Buy additional AMZN worth 2% of NAV; sell JPM worth 1% and XOM worth 1.5% of NAV. **Rationale:** AMZN's RSI of 23.35 suggests an oversold rebound.
RT: Challenge	**Evidence:** AMZN's 10- and 20-day returns are -5.01% and -8.42%, and its price is 6.61% below the 20-day moving average. **Challenge:** Low RSI alone does not justify adding exposure while the downtrend persists. **Risk condition:** AMZN's return at the end of the five-trading-day horizon is below -3%.
Trader: Revise	**Accept the challenge and cancel the additional AMZN purchase.** Retain the existing AMZN shares without selling them. Keep the JPM and XOM sell orders unchanged.
Risk evaluation	AMZN returns -4.42% over the evaluation window ending on June 25, satisfying the pre-specified risk condition. The warning is a **hit**.
Revision evaluation	The original proposal yields -0.0148%, whereas the revised plan yields +0.0745%. Canceling the purchase improves net return by **0.0893%**, turning a small evaluated loss into a gain.

assessment into a beneficial, targeted revision. Together with the CAT case, it illustrates that the Trader need not automatically accept or reject challenges: it can retain a proposal when the evidence for changing it is insufficient and revise it when a specific risk is supported.

B Settlement Feedback and Online Updates

Constructing supervision. At each trading step t, we run ACE-Trader and a memory-free reference using the same market information o_t and pre-decision account state b_t. The reference runs its own Commit–Challenge–Revise workflow without recalling private memories or shared cases. We shadow-execute the two final plans, meaning that each is simulated on a copy of b_t over the same settlement interval using identical market prices, execution rules, and transaction-cost rules. The reference is initialized from the current b_t at every step, rather than carrying forward its previous simulated cash and holdings. Only the memory-assisted final plan is executed in the persistent trading account.

Once the settlement outcomes are observed, we compare the net returns of the memory-assisted and memory-free plans. We assign $\zeta_t = 1$ if the memory-assisted plan outperforms the reference by more than a tolerance $\epsilon \geq 0$, and $\zeta_t = 0$ if it underperforms by more than ϵ. Trading steps whose return difference lies within $[-\epsilon, \epsilon]$ are excluded from training.

Each training example pairs a selected memory's representation $\mathbf{v}^u_{t,m}$, saved at selection time, with ζ_t. Both selectors receive the same label but train on their own selections, including the Trader's Commit and Revise selections when both occur. A role that selects no memory contributes no new training example for that step. The shared label compares memory use across the complete workflow at the current account state; it does not measure the separate contribution of an individual memory.

Recency-weighted objective. For role $i \in \{\mathrm{Tr}, \mathrm{RT}\}$, let $\mathcal{B}^i_t$ contain the representations saved for its memory selections at trading step t. A step is valid for the role when its feedback is available and not excluded, and $\mathcal{B}^i_t$ is nonempty. The k-th ensemble member, parameterized by $\theta_{i,k}$, predicts the probability $p_{\theta_{i,k}}(\mathbf{v})$ of positive feedback. Its loss for a valid trading step is

$$\ell_t(\theta_{i,k}) = \frac{1}{|\mathcal{B}^i_t|} \sum_{\mathbf{v} \in \mathcal{B}^i_t} \mathrm{BCE}\left(p_{\theta_{i,k}}(\mathbf{v}), \zeta_t\right), \tag{6}$$

where BCE denotes binary cross-entropy. Averaging within each trading step prevents steps with more selections from receiving extra training weight.

Each selector retains records from the W most recent observed settlement days, or all observed settlement days when fewer than W are available. Only valid trading steps within this window contribute to training; days without valid feedback are not replaced by older days outside the window. After settling step t, let $t_1 > \cdots > t_J$ denote the contributing trading steps from newest to oldest, with $J \leq W$. For $J > 0$, the training objective is

$$\mathcal{L}_t(\theta_{i,k}) = \frac{\sum_{j=1}^{J} \mathrm{pow}(\gamma, d_{t,j}) \ell_{t_j}(\theta_{i,k})}{\sum_{j=1}^{J} \mathrm{pow}(\gamma, d_{t,j})}, \tag{7}$$

where $d_{t,j}$ denotes the distance between the settlement days of steps t_j and t, and $\mathrm{pow}(x, n)$ denotes the n-th power of x. The weights depend on settlement-day distance rather than the rank of valid records, and the denominator normalizes their sum. The window size W and decay factor γ are hyperparameters.

Optimization. When valid feedback becomes available for trading step t, we update each ensemble member by minimizing $\mathcal{L}_t(\theta_{i,k})$, provided that its selector has usable records within the current window. Feedback excluded by the tolerance rule does not trigger an update. The saved representations and labels remain fixed, while network parameters and optimizer states are retained between updates. The semantic encoder and trading LLMs remain frozen. Numerical settings and implementation details are provided in Appendix E.4.

C Dataset and Evaluation Details

C.1 Data Sources and Stock Universe

We evaluate ACE-Trader on a daily U.S. stock-trading benchmark comprising 33 equities across 11 sectors. Historical prices and financial news are obtained from FNSPID (Dong et al., 2024), a large-scale financial dataset containing 29.7 million stock-price records and 15.7 million time-aligned news articles. The stock universe consists of U.S.-listed equities with valid OHLCV records and SEC total-asset metadata, covering large-, mid-, and small-size strata where available.

The selected tickers are `AIT`, `AMT`, `AMZN`, `APD`, `AWR`, `BHP`, `BKNG`, `CAT`, `CMC`, `COP`, `CUBE`, `DUK`, `ELF`, `EXEL`, `FOXA`, `FSLY`, `GE`, `GOOGL`, `HOG`, `JNJ`, `JPM`, `KHC`, `MSFT`, `NRG`, `NVDA`, `RDN`, `RJF`, `RRC`, `TGNA`, `VICI`, `WMT`, `XOM`, and `ZBH`. This asset universe remains fixed throughout training and testing.

Historical training uses data from 2021-01-01 to 2023-12-31, without a separate validation set. Testing covers 2026-01-01 to 2026-06-30. All data for this later period, including prices, financial news, and associated asset information, are obtained from Financial Modeling Prep (FMP). The later test period is intended to mitigate potential temporal leakage in retrospective LLM-based trading evaluation. Both periods use the same feature definitions, cleaning procedures, and temporal filtering rules.

C.2 Input Construction and Causal Preprocessing

Prices and features. Price records retain raw open, high, low, close, and volume (OHLCV), together with adjusted closing prices for corporate-action auditing. Records without a valid date or closing price are removed. Missing open, high, or low values are replaced with the closing price from the same record, and missing trading volume is set to zero. Prices are never interpolated across dates.

Each asset input contains its sector, market capitalization, five completed daily OHLCV bars, and technical indicators computed over trailing 30- and 60-day windows. Historical price windows end at the close of the preceding trading day. The same feature construction is applied to the FNSPID historical data and the FMP test data.

Financial news. The historical FNSPID news corpus contains 26,305 unique asset–article records for the selected stock universe. News for the later test period is obtained from FMP. For each trading date, the news cutoff is set to 20:00 in the `America/New_York` timezone on the preceding day. We retain at most the two most recent asset-specific records published within the seven days preceding

Table 4: Mapping absolute predicted returns to position-size levels.

Absolute predicted return	Position-size level $q_{t,j}$
$0.2\% \leq \lvert\hat{r}_{t,j}\rvert \leq 0.6\%$	1
$0.6\% < \lvert\hat{r}_{t,j}\rvert \leq 1.2\%$	2
$1.2\% < \lvert\hat{r}_{t,j}\rvert \leq 1.8\%$	3
$1.8\% < \lvert\hat{r}_{t,j}\rvert \leq 2.4\%$	4
$\lvert\hat{r}_{t,j}\rvert > 2.4\%$	5

this cutoff. An asset with no eligible record receives an empty news list. The timestamp-filtered offline records are fixed for each decision date and shared across methods.

Account context and information availability. The trading input also includes the current account state: available cash, existing holdings, net asset value, gross exposure, and net exposure. The current decision date and the next settlement date are provided, but future settlement prices are excluded. All decisions, memory retrieval, and financial tool queries are restricted to information available at the corresponding decision cutoff. Outcome-dependent experience becomes available only after its supporting market outcomes have been observed.

C.3 Unified Portfolio Construction

All active methods use the same asset universe, daily execution frequency, portfolio constraints, and transaction-cost settings. Each method retains its internal architecture and decision process. Method-specific adapters translate its native outputs into executable adjustments to the inherited account. Trading actions and target portfolio weights are interpreted relative to the current holdings.

Return forecasts. For methods that predict asset returns, we apply the following return-to-position conversion to each asset. A predicted return of at least 0.2% produces a long decision, a predicted return of at most -0.2% produces a short decision, and a prediction between these thresholds produces a flat decision. For a non-flat decision, the absolute predicted return determines the position-size level $q_{t,j}$ according to Table 4, where t indexes the trading step and j indexes the asset. Flat decisions receive a position-size level of zero. These conversions use only model predictions and never access realized future returns.

Reinforcement-learning actions. For an RL policy, the signed action is clipped to $[-1, 1]$. Its sign determines the trading direction, and its absolute magnitude is multiplied by five to obtain a position-size level in $[0, 5]$. A zero action produces a flat position.

Direct trading decisions. ACE-Trader outputs `BUY`, `SELL`, or `HOLD` decisions. For assets to be adjusted, `weight_pct` specifies the signed target position value as a percentage of the account's net asset value after transaction costs, rather than the percentage to buy or sell. Positive and negative target weights denote long and short positions, respectively. These continuous target weights are not discretized into the five position-size levels.

A `BUY` can open or increase a long position or cover a short position. A `SELL` can reduce a long position or open or increase a short position. The executed quantity is determined by the difference between the target holding and the current holding. Position retention under `HOLD` or omission follows the rules below.

For baseline outputs expressed as a target-position direction $s_{t,j}$ and a position-size value $q_{t,j}$, we retain the conversion $w_{t,j} = 0.2 s_{t,j} q_{t,j}/5$. Here, $s_{t,j}$ denotes a long, short, or zero target position, rather than a `BUY`, `SELL`, or `HOLD` action. This conversion is not applied to ACE-Trader's directly specified target weights.

Portfolio constraints and position retention. Both long and short positions are permitted. The resulting portfolio, including both adjusted and retained positions, is subject to a gross-exposure limit of 100% of net asset value. A `HOLD` decision or omission of an asset preserves its existing share quantity without trading. Target weights are applied only to positions explicitly designated for adjustment.

An explicit zero target for a position to be adjusted closes that position. Neither `HOLD` nor omission is interpreted as a zero target. Explicit zero targets produced by a baseline's output adapter remain zero-exposure instructions.

Unknown assets are discarded without affecting existing holdings. If an output cannot be parsed or contains no valid decision, no trades are executed and the current holdings are retained. Unallocated capital remains in cash.

C.4 Trading and Evaluation Protocol

Continuous-account execution. Each evaluation trajectory starts without active positions, and its net asset value is normalized to one for reporting. At each trading step, the trading plan is applied to the inherited account. For ACE-Trader, only the final memory-assisted plan is executed in the continuous trading account. The paired shadow executions used for selector supervision do not alter this account, and the memory-free reference does not maintain a separate account across trading steps.

For each position explicitly designated for adjustment, the target position value is obtained from its target weight and the account's net asset value after transaction costs. Dividing this value by the execution price gives the target share quantity. Only the difference between the target and current share quantities is traded. Positions assigned `HOLD` or omitted from the plan retain their current share quantities and are not rebalanced to their previous portfolio weights.

Transaction costs. A one-way transaction-cost rate of 5 basis points is applied to traded notional. Costs are charged when positions are opened, adjusted, closed, or reversed. For a retained asset, costs apply only to the executed adjustment. A position reversal counts both the closing and opening amounts. No additional full-liquidation cost is imposed at the end of each trading day. Borrowing fees, additional slippage, and cash interest are not modeled, and unallocated cash earns zero return. The same cost assumptions apply to every active method.

Chronological evaluation. Task-specific offline training uses only the historical training period, with no separate validation set and no method-specific hyperparameter search on the test period. Baseline configurations follow Appendix E.5. The test period is processed in chronological order.

For ACE-Trader, the trading LLMs and semantic encoder remain frozen. New experience is stored only after the associated outcomes become observable, and selector updates use only feedback from completed settlement intervals. These updates affect subsequent decisions and follow the online learning procedure described in the method section.

Passive benchmarks. We additionally report Market Buy-and-Hold, which tracks the Nasdaq Composite over the test period, and Equal-Weight Buy-and-Hold, which allocates capital equally across the 33 stocks at the beginning of the period and holds the resulting portfolio without active rebalancing. These passive strategies bypass the active decision-conversion pipeline and serve as reference trajectories.

C.5 Evaluation Metrics

We report cumulative return, annualized return, Sharpe ratio, maximum drawdown, annualized volatility, hit rate, and average daily turnover. Let T denote the number of daily return observations in the continuous account trajectory, and let r_t^{net} denote the corresponding daily net portfolio return. The normalized account value after t completed intervals is $V_t = \prod_{s=1}^{t}(1 + r_s^{\mathrm{net}})$, with $V_0 = 1$. All portfolio-return metrics use returns after transaction costs, and annualization uses 252 trading days.

Cumulative return. Cumulative return is $V_T - 1$, measuring the total change in account value over the test trajectory.

Annualized return. Annualized return is $V_T^{252/T} - 1$, expressing the compounded portfolio return on an annualized basis.

Sharpe ratio. The Sharpe ratio is $\sqrt{252}\,\overline{r^{\mathrm{net}}}/\,\mathrm{std}(r^{\mathrm{net}})$, where $\overline{r^{\mathrm{net}}}$ and $\mathrm{std}(r^{\mathrm{net}})$ are the mean and standard deviation of daily net portfolio returns. The risk-free rate is set to zero.

Maximum drawdown. Maximum drawdown is the largest percentage decline from a preceding peak in account value: $\max_{1\le t\le T}\,(1 - V_t/\max_{0\le s\le t} V_s)$. Lower values indicate smaller peak-to-trough losses.

Annualized volatility. Annualized volatility is $\sqrt{252}\,\mathrm{std}(r^{\mathrm{net}})$, measuring the variability of net portfolio returns.

Hit rate. Hit rate measures directional accuracy over held asset–interval positions. Let $\rho_{t,j}$ denote the realized return of asset j over the corresponding execution–settlement interval, and let $w_{t,j}$ denote its signed portfolio weight after execution, including positions retained through `HOLD` or omission. We compute

$$\mathrm{HitRate} = \frac{\sum_{t,j} \mathbf{1}[w_{t,j}\rho_{t,j} > 0]}{\sum_{t,j} \mathbf{1}[|w_{t,j}| > 0,\ |\rho_{t,j}| > 0]}. \tag{8}$$

Zero-exposure positions and zero-return intervals are excluded from the denominator. Retained nonzero positions are included even when no trade is executed. This metric evaluates the direction of the position held, rather than the direction of the current trading action or the fraction of profitable portfolio days.

Average daily turnover. For each trading step, turnover τ_t is the total absolute traded notional divided by the pre-decision account net asset value. Average daily turnover is $T^{-1}\sum_{t=1}^{T}\tau_t$. Both opening and closing transactions are counted, without the conventional one-half factor.

D Further Descriptions of Baselines

We provide additional descriptions of the 22 active baselines, organized into the same six categories as in the main text, together with two passive benchmarks. Each method retains its original architecture and learning objective. We standardize its terminal interface with the trading environment: direct decision generators produce executable trading decisions, while forecasting models and reinforcement-learning policies are connected through the output-conversion procedure described in Appendix C.3. All active methods follow the same temporal information cutoff, asset universe, daily execution frequency, transaction costs, and portfolio constraints.

D.1 Conventional Forecasting Methods

These methods predict asset returns from structured market features. They capture nonlinear feature interactions, temporal dependencies, or cross-asset relationships, and their predictions are converted into portfolio decisions through the common execution interface.

LightGBM. LightGBM (Ke et al., 2017) is a histogram-based gradient-boosting framework that grows decision trees leaf-wise. It models nonlinear interactions among market features to produce asset-level return estimates.

XGBoost. XGBoost (Chen & Guestrin, 2016) constructs an additive ensemble of regression trees, with successive trees correcting the prediction errors of the existing ensemble. Regularization controls model complexity and reduces overfitting.

LSTM. LSTM (Hochreiter & Schmidhuber, 1997) encodes historical market sequences using gated recurrent memory cells. Its temporal representation is used to predict the future return of each asset.

GRU. GRU (Cho et al., 2014; Chung et al., 2014) models temporal dependencies through recurrent update and reset gates. It provides a compact gated alternative to LSTM for learning patterns in historical market sequences.

MLP. MLP (Hornik et al., 1989) maps flattened rolling market features to asset-level return predictions. It serves as a feedforward baseline without an explicit temporal or relational modeling component.

TRA. TRA (Lin et al., 2021) maintains multiple return predictors associated with different latent trading patterns. A temporal routing adaptor assigns market samples to the relevant predictors, while an optimal-transport objective encourages them to capture distinct patterns.

HIST. HIST (Xu et al., 2021) models cross-stock dependencies through concept-oriented representations. It combines information associated with predefined and latent market concepts to generate asset-level return forecasts.

D.2 Reinforcement-Learning Trading Methods

The FinRL baselines (Liu et al., 2021) observe structured market and account information and learn trading policies through sequential interactions with a portfolio environment.

FinRL-DQN. FinRL-DQN estimates the expected long-term return of discrete trading actions using a deep Q-network. The policy selects the action with the highest estimated value at each trading step.

FinRL-A2C. FinRL-A2C combines an actor that produces trading actions with a critic that estimates state values. An advantage signal guides the joint optimization of the policy and value networks.

FinRL-PPO. FinRL-PPO optimizes its trading policy using a clipped policy-gradient objective. The clipping mechanism limits excessively large policy changes during training.

D.3 Frontier General-Purpose LLMs

These models directly receive the structured market and account context and generate trading decisions without task-specific fine-tuning. They use the same direct-decision prompt template, temporal cutoff, asset universe, and output schema, without an additional trading-agent workflow.

GLM 5.3 Flash. GLM 5.3 Flash(Zeng et al., 2026) is prompted to analyze the supplied market context and directly generate trading actions and position sizes. Its outputs are evaluated through the common portfolio interface.

DeepSeek V4 Flash. DeepSeek V4 Flash (Xu et al., 2026) is evaluated as a separate direct-decision model using the same input template and output protocol as GLM 5.3 Flash. No task-specific parameter updates are performed.

Qwen3.6-35B-A3B. Qwen3.6-35B-A3B (Team, 2026) processes the complete trading prompt and directly produces structured trading decisions. The pretrained model is used without task-specific fine-tuning.

D.4 Finance-Specific LLMs

These baselines use financial-domain pretraining or adaptation to produce forecasts or trading decisions from financial text or market sequences. Direct decisions are passed to the common portfolio interface, while forecasting outputs are converted into executable trading actions and position sizes.

FinGPT. FinGPT (Yang et al., 2023) is a finance-oriented language model adapted to financial text and instruction-following tasks. It receives the structured trading context, and its generated response is interpreted under the common trading-decision schema.

FinCast. FinCast (Zhu et al., 2025) is a foundation model for financial time-series forecasting. It produces asset-level forecasts from historical market sequences, which are subsequently converted into portfolio decisions.

Kronos. Kronos (Shi et al., 2026) is a pretrained financial time-series model that captures temporal patterns in market sequences and predicts future price or return dynamics. Its forecasts are evaluated through the same portfolio-construction and execution procedure as the other forecasting baselines.

D.5 Single-Agent Trading Systems

These methods organize market analysis and decision making around one trading agent. Unlike direct prompting, they retain their agentic workflows and support decision making through tools, memory, reflection, or reinforcement learning.

FinAgent. FinAgent (Zhang et al., 2024) combines multimodal market information with tool-assisted analysis, diversified memory, and reflection. The agent integrates these intermediate analyses into a final trading recommendation.

FLAG-Trader. FLAG-Trader (Xiong et al., 2025) combines an LLM-based trading agent with gradient-based reinforcement learning. Trading feedback is used to optimize the policy associated with the agent's market representations and decision process.

D.6 Multi-Agent and Expert-Coordinated Trading Systems

These methods combine the assessments of specialized agents or financial experts through collaboration, debate, or context-dependent coordination. We retain their original role assignments and coordination mechanisms while evaluating their final recommendations through the common portfolio interface.

TradingAgents. TradingAgents (Xiao et al., 2024) distributes market analysis, strategy discussion, and risk assessment across specialized agents. Their assessments are coordinated through debate and risk-management stages before a final trading decision is produced.

LLMoE. LLMoE (Liu & Lo, 2025) uses an LLM-based router to select or combine predefined financial experts according to the current market context. The expert outputs are aggregated into a final trading recommendation.

TradExpert. TradExpert (Ding et al., 2024) assigns different analytical responsibilities or information sources to specialized LLM experts. A coordinating component integrates their outputs into a unified trading decision.

MM-DREX. MM-DREX (Chen et al., 2025) conditions expert coordination on multimodal market information. Its dynamic-routing mechanism selects or weights specialized LLM experts before aggregating their recommendations.

D.7 Passive Benchmarks

Market Buy-and-Hold. This benchmark follows the Nasdaq Composite throughout the test period and provides a reference for broad market performance.

Equal-Weight Buy-and-Hold. This benchmark allocates capital equally across all candidate stocks at the beginning of the test period and holds the resulting portfolio without active rebalancing.

E Implementation Details

E.1 Experimental Environment and Backbones

Experiments are conducted on Ubuntu 22.04 with 16 NVIDIA A800-SXM4-80GB GPUs. Baseline architectures, optimization settings, and inference configurations are specified in Appendix E.5.

The Trader uses Deepseek v4 Flash, and the Red-Team uses Deepseek v4 Flash. Candidate memories are encoded using MiniLM. The trading LLMs and semantic encoder remain frozen throughout historical warm-up and evaluation. Only the role-specific selector networks receive parameter updates.

E.2 Historical Warm-up and Initialization

Historical warm-up uses data from 2021-01-01 to 2023-12-31, without a separate validation set. Testing covers 2026-01-01 to 2026-06-30 and follows the data construction and chronological evaluation protocol described in Appendies C.4 and C.1. No method-specific hyperparameter search is performed on the test period.

At the beginning of historical warm-up, the selector networks are randomly initialized using the default PyTorch initialization, and all private and shared memory stores are initially empty. We then process the 2021–2023 warm-up period once in chronological order. After each completed settlement, the resulting experience is added to memory and each eligible selector performs one online update using its current recency-weighted training buffer.

Evaluation starts from an all-cash account with no active positions, and net asset value is normalized to one for reporting. Cash, holdings, and account value are subsequently carried forward without daily account resets. During testing, memory construction and selector updates use only outcomes from completed settlement intervals and affect subsequent decisions.

E.3 Memory Retrieval and Selection

Candidate retrieval. Each invocation retrieves candidates from the corresponding role's private memory and the shared case memory. The retrieval procedure and candidate limits are 10.

Selector architecture. The Trader and Red-Team maintain separate selectors, each consisting of $K = 3$ feedforward networks. The phase-specific retrieval query and candidate memory are encoded into 384-dimensional embeddings using the frozen encoder E. We concatenate the two embeddings and their element-wise product to obtain a 1,152-dimensional vector. A linear layer maps this vector to 16 dimensions, followed by a Tanh activation, producing the semantic features $\mathbf{e}^u_{t,m}$ used in Eq. 1.

We then concatenate this 16-dimensional semantic feature with the corresponding market, portfolio, and recorded-outcome features to form the complete selector input. The resulting input dimension is 42 for the Trader and 43 for the Red-Team. Each complete feature vector is then fed into every network of the corresponding selector ensemble.

Each Trader network has dimensions $42 \to 32 \to 16 \to 1$, whereas each Red-Team network has dimensions $43 \to 32 \to 16 \to 1$. The hidden layers use LeakyReLU activations with a negative slope of 0.01. The final layer outputs a logit during training, and a sigmoid is applied at inference time to obtain the predicted probability.

Selection rule. Candidates are scored by the ensemble mean prediction plus the disagreement bonus defined in Section 4.2. The disagreement-bonus coefficient is $\alpha = 0.8$.

For every completed selection, the selected candidate and its selection-time feature vector are saved for subsequent feedback.

Memory retention and consolidation. The Trader and Red-Team maintain separate short-term experience stores and long-term lessons. The short-term memory of each role is capped at 24 experience records. Long-term consolidation is triggered after every 10 newly added short-term expe-

rience records. The resulting long-term lessons are stored separately and do not count toward the 24-record short-term memory capacity.

During consolidation, summaries of recent experience are provided to an LLM to extract recurring lessons with explicit applicability conditions. Each lesson retains references to its supporting shared cases and distinguishes whether a risk occurred from whether the Trader's revision improved net return. These references allow either role to revisit the corresponding interaction without accessing the other role's private memory.

E.4 Selector Training Settings

We set the feedback tolerance to $\epsilon = 0.0005$ (0.05 percentage points), the training window to $W = 5$ observed settlement days, and the decay factor to $\gamma = 0.8$. All selector hyperparameters are determined using only the 2021–2023 historical period and fixed before the 2026 test period. The objective defined in Appendix B is implemented using `BCEWithLogitsLoss`. We use Adam with a learning rate of 0.01, weight decay of 10^{-4}, betas $(0.9, 0.999)$, and a numerical stability constant of 10^{-8}. Each valid feedback event triggers one gradient update per network.

E.5 Baseline Configurations

For all open-source baselines, we use their official implementations or faithful interface wrappers, retaining the released architectures, optimizers, learning-rate schedules, and hyperparameters. Only dataset-dependent input and output dimensions are adapted to the common market features, asset universe, and trading-decision space. All methods are evaluated through the same portfolio post-processor and simulator, without method-specific hyperparameter search on the test period. The configurations are organized into the same six categories as in the main text.

Conventional forecasting methods. For LightGBM (Ke et al., 2017) and XGBoost (Chen & Guestrin, 2016), we retain their official baseline hyperparameters without task-specific tuning. LSTM (Hochreiter & Schmidhuber, 1997) and GRU (Cho et al., 2014; Chung et al., 2014) use two recurrent layers, a hidden dimension of 64, and no dropout. Both models are trained for at most 200 epochs with a learning rate of 10^{-3} and a batch size of 800. For MLP (Hornik et al., 1989), we follow the official Qlib configuration, using a learning rate of 2×10^{-3}, a batch size of 4,096, and at most 8,000 optimization steps. TRA (Lin et al., 2021) employs a two-layer LSTM backbone with a hidden dimension of 64 and three latent trading states. Its routing adaptor is a one-layer LSTM with a hidden dimension of 32. The sequence length and batch size are set to 60 and 1,024, respectively. HIST (Xu et al., 2021) uses a two-layer LSTM backbone with a hidden dimension of 64, mean squared error as the prediction loss, and a learning rate of 10^{-4}.

Reinforcement-learning trading methods. FinRL-DQN, FinRL-A2C, and FinRL-PPO (Liu et al., 2021) use the official FinRL implementations. FinRL-DQN retains the default DQN configuration. FinRL-A2C uses 5 rollout steps, a learning rate of 7×10^{-4}, and an entropy coefficient of 0.01. FinRL-PPO uses 2,048 rollout steps, a batch size of 64, a learning rate of 2.5×10^{-4}, and an entropy coefficient of 0.01.

Frontier general-purpose LLMs. For GLM 5.3 Flash(Zeng et al., 2026), DeepSeek V4 Flash (Xu et al., 2026), and Qwen3.6-35B-A3B (Team, 2026), we use the official identifiers `glm-5.3-flash`, `deepseek-v4-flash`, and `Qwen/Qwen3.6-35B-A3B`, respectively. All three models use their default reasoning configurations without task-specific fine-tuning. They receive the same structured trading prompt and produce decisions in the same output format.

Finance-specific LLMs. FinGPT (Yang et al., 2023) uses the officially released Llama-2-7B LoRA financial-forecaster checkpoint. FinCast (Zhu et al., 2025) uses its released pretrained checkpoint under the recommended zero-shot forecasting setting. Kronos (Shi et al., 2026) uses the officially released checkpoint and default inference configuration, with temperature 1.0, Top-p 0.9, and one forecast sample.

Single-agent trading systems. For FinAgent (Zhang et al., 2024), we preserve the officially released market-intelligence, diversified-memory, reflection, and decision modules. FLAG-Trader

(Xiong et al., 2025) uses SmolLM2-135M-Instruct as its backbone. Its learning rate, rollout length, discount factor, and PPO clipping coefficient are set to 5×10^{-4}, 40, 0.95, and 0.2, respectively. The model is trained for 13,860 steps.

Multi-agent and expert-coordinated trading systems. TradingAgents (Xiao et al., 2024) retains its original analyst roles, debate procedure, risk-management module, and default model assignments. LLMoE (Liu & Lo, 2025) uses a five-day lookback window, a Llama-3.2 router, and two sentiment-oriented experts. Each expert is a feedforward network with hidden dimensions of 128, 64, and 32, with dropout rates of 0.3 and 0.2 after the second and third layers, respectively. TradExpert (Ding et al., 2024) employs four specialized experts and one general expert based on LLaMA-2-7B. Its LoRA rank, scaling factor, and dropout are set to 8, 16, and 0.1, respectively. Training runs for 30 epochs with a learning rate of 10^{-4}, a batch size of 4, and 8 gradient-accumulation steps. MM-DREX (Chen et al., 2025) uses Qwen2.5-VL-72B-Instruct with four expert heads and a 100-day lookback window. Its LoRA rank, scaling factor, dropout, and learning rate are set to 16, 32, 0.05, and 3×10^{-5}, respectively. The discount factor and policy clipping coefficient are 0.99 and 0.15.

E.6 Ablation Settings

All variants retain the same Commit–Challenge–Revise workflow. When TS or RS is disabled, we retain the candidate retrieval procedure and supply its top-ranked candidate without learned reranking, keeping the one-memory budget unchanged. Disabling TS applies to both Commit and Revise. When CE is disabled, each role continues to update and consolidate its private memories from its own outputs and observed outcomes. Shared interaction records are retained for evaluation but are neither retrieved nor used to construct memories linking the two roles. The enabled selectors follow the same supervision and update procedure as the complete model.