FLAG-Trader: Fusion LLM-Agent with Gradient-based Reinforcement Learning for Financial Trading

Guojun Xiong[1], Zhiyang Deng[2], Keyi Wang[3], Yupeng Cao[2], Haohang Li[2], Yangyang Yu[2], Xueqing Peng[7], Mingquan Lin[4], Kaleb E Smith[5], Xiao-Yang Liu Yanglet[3,6], Jimin Huang[7], Sophia Ananiadou[8], Qianqian Xie[7,*]

[1]Harvard University, [2]Stevens Institute of Technology, [3]Columbia University, [4]University of Minnesota, [5]NVIDIA, [6]Rensselaer Polytechnic Institute, [7]TheFinAI, [8]University of Manchester

[*]**Correspondence:** xqq.sincere@gmail.com

Abstract

Large language models (LLMs) fine-tuned on multimodal financial data have demonstrated impressive reasoning capabilities in various financial tasks. However, they often struggle with multi-step, goal-oriented scenarios in interactive financial markets, such as trading, where complex agentic approaches are required to improve decision-making. To address this, we propose FLAG-Trader, a unified architecture integrating linguistic processing (via LLMs) with gradient-driven reinforcement learning (RL) policy optimization, in which a partially fine-tuned LLM acts as the policy network, leveraging pre-trained knowledge while adapting to the financial domain through parameter-efficient fine-tuning. Through policy gradient optimization driven by trading rewards, our framework not only enhances LLM performance in trading but also improves results on other financial-domain tasks. We present extensive empirical evidence to validate these enhancements.

1 Introduction

Algorithmic financial trading represents a critically complex decision-making domain that perpetually grapples with the intertwined challenges of synthesizing heterogeneous market signals and dynamically refining strategies (Hambly et al., 2023; Yu et al., 2025; Li et al., 2023). Traditional reinforcement learning (RL) approaches, despite their theoretical grounding in Markov Decision Processes (MDPs), confront three fundamental limitations when deployed in financial markets. Firstly, their inability to coherently model multimodal market states—spanning sequential price movements, quantitative technical indicators, and unstructured textual sentiments—compromises data integration (Zhang et al., 2019; Nassirtoussi et al., 2014). Secondly, non-stationary data distributions inherent to financial systems systematically erode strategy generalizability across market regimes (Zhang et al., 2019). Thirdly, the heavy reliance on manually crafted technical indicators (e.g., MACD, RSI) and complex feature engineering (Liang et al., 2018) introduces subjective biases, leads to information loss, and reduces the robustness of real-time decision-making, especially in volatile market conditions.

The emergence of Large Language Models (LLMs) offer significant potential for financial decision-making by addressing key limitations of RL-based trading strategies. Leveraging their transformer architecture, they serve as multimodal feature extractors, integrating time-series and textual data, capturing long-range dependencies, and generalizing across market regimes, while also extracting nuanced sentiment signals without relying on manually crafted features (Chen et al., 2021; Yang et al., 2023a; Jin et al., 2023; Wood et al., 2021; Yu et al., 2024; Deng et al., 2023). Nonetheless, adapting LLMs for trading presents key challenges. First, their deployment often relies on agentic frameworks (Li et al., 2024b, 2023; Yu et al., 2025), which incur high implementation and operational costs due to their complex architecture. Second, LLMs are primarily trained for static text generation, making them ill-suited for sequential decision-making in trading. This prompts us to the following question:

> *Can we design a framework that seamlessly integrates LLMs' reasoning with RL's reward-driven optimization to tackle the challenges of financial sequential decision-making?*

To resolve these interconnected challenges, we propose FLAG-Trader, a unified architecture integrating linguistic processing (via LLMs) with gradient-driven RL policy optimization, as shown in Figure 1. This framework advances two synergistic innovations: a parameter-efficient fine-tuning module that jointly encodes temporal market data

Findings of the Association for Computational Linguistics: ACL 2025, pages 13921–13934

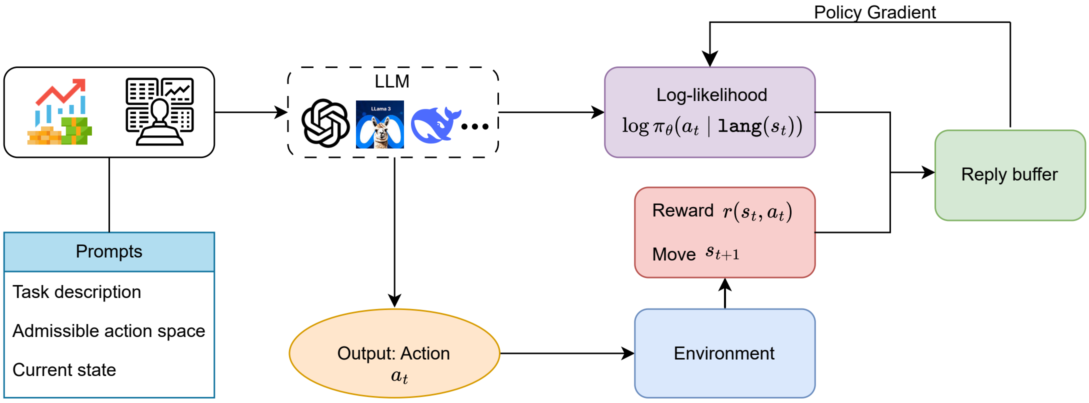

Figure 1: A high-level overview of our LLM-based reinforcement learning setup for financial trading. The environment provides the current state s_t. A prompt containing task details, the action space, and the current state is fed into the LLM, which outputs a trading action a_t. The action is executed in the environment, yielding a reward $r(s_t, a_t)$ and next state s_{t+1}. The log-likelihood $\log_{\pi_\theta}(a_t|\texttt{lang}(s_t))$ is then leveraged by a policy gradient method (e.g., PPO), with experience tuples stored in a replay buffer for iterative updates.

and textual streams into unified state representations and a hybrid RL component that explicitly incorporates external environment reward gradients into policy updates, ensuring alignment with trading performance metrics. Our contributions are summarized as follows.

First, we propose the FLAG-TRADER framework, where a partially fine-tuned LLM acts as the policy network, leveraging pre-trained knowledge while adapting to the financial domain through parameter-efficient fine-tuning. The model processes market data using a textual state representation, enabling it to interpret and respond to market conditions effectively. Rather than fine-tuning the entire LLM, only a subset of its parameters is updated, balancing domain adaptation and knowledge retention. This design allows FLAG-TRADER to make informed trading decisions while remaining computationally efficient and preserving the LLM's general reasoning capabilities.

Second, we conduct extensive experiments to evaluate FLAG-TRADER across multiple financial trading tasks. Our results demonstrate that FLAG-TRADER consistently outperforms both the buy-and-hold strategy and LLM-agentic baselines, particularly in terms of cumulative return and Sharpe ratio, which we prioritize for financial performance assessment. Notably, our approach enables a small-scale (135M parameter) open-source LLM to surpass much larger proprietary models, highlighting the effectiveness of RL fine-tuning in optimizing LLM-driven trading strategies. These findings underscore the potential of integrating LLMs with RL to enhance financial decision-making while maintaining computational efficiency.

2 Related Work

RL in Finance. RL has shown promise for financial decision-making, spanning Q-learning approaches for Sharpe ratio maximization (Gao and Chan, 2000), dynamic asset allocation (Jangmin et al., 2006), deep Q-learning (Jeong and Kim, 2019), tabular SARSA (de Oliveira et al., 2020), policy-based portfolio optimization (Shi et al., 2019), and actor-critic methods (Ye et al., 2020) enhanced by adversarial training (Liang et al., 2018) and transformer-based architectures (Huang et al., 2024). Recent research efforts in RL for financial applications have been greatly aided by open-source frameworks like FinRL (Liu et al., 2022), which standardize implementations and provide reproducible benchmarks. Comprehensive surveys (Hambly et al., 2023; Sun et al., 2023) further showcase advances in both methodological rigor and real-world deployment. Despite these advances, RL-based trading still requires large training data, struggles with non-stationary markets, and faces challenges incorporating multimodal information in real time.

LLMs in Finance. A growing trend is the integration of LLMs into financial decision-making. Hybrid systems like FinCon (Yu et al., 2025) and TradingGPT (Li et al., 2023) leverage language understanding to enhance trading agents, while domain-specific models such as FINBERT (Araci, 2019; Yang et al., 2020), FLANG (Shah et al., 2022) and OPEN-FINLLMS (Xie et al., 2024b) have excelled at financial text tasks through specialized pre-training. Recent efforts include machine reading comprehension (Zhang and Zhang, 2023), open-source financial LLMs (Liu et al., 2023), BloombergGPT with domain-adapted tokenization (Wu et al., 2023), and InvestLM (Yang et al., 2023b) featuring numerical reasoning—achieving strong results in sentiment analysis (Huang et al., 2023), earnings call interpretation (Xie et al., 2023), and regulatory document processing. Additionally, FINBEN (Xie et al., 2024a), benchmark study for LLMs in finance, have emerged to comprehensively evaluate model performance across various financial tasks. However, LLM-based methods often lack sequential decision-making mechanisms, are computationally expensive (especially with RL), and struggle with non-stationary market conditions.

LLM Agents for Sequential Decision Making. The integration of LLMs with agentic frameworks has opened new avenues for financial decision-making. For instance, FINMEM (Yu et al., 2024) introduced memory-augmented LLM agents for portfolio management, FINAGENT (Zhang et al., 2024) leveraged hierarchical structures in high-frequency trading, and multi-agent systems like FINROBOT (Yang et al., 2024) and FINCON (Yu et al., 2025) emphasize contextual adaptation and collaboration. Meanwhile, fine-tuning LLMs and vision-language models (VLMs) with reinforcement learning has proven effective in complex tasks: LLaRP (Szot et al., 2023) positions LLMs as generalizable policies for embodied tasks, and RL-tuned VLMs (Zhai et al., 2024) enhance multi-step decision-making. However, LLMs remain computationally expensive for real-time deployment, and risk-sensitive trading demands robustness to non-stationary markets, calling for careful model complexity and balanced exploration-exploitation.

3 Problem Statement

We define the financial decision-making process as a finite horizon partially observable Markov decision process (POMDP) with time index $\{0, \cdots, T\}$, represented by the tuple: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma)$, where each component is described in detail below.

State. The state space $\mathcal{S} = \mathcal{X} \times \mathcal{Y}$ consists of two components: market observations and trading account balance, i.e., $s_t = (m_t, b_t) \in \mathcal{S}$. Specifically, $m_t = (P_t, N_t) \in \mathcal{X}$ represents the *market observation process*, including stock price P_t at time t, and financial news sentiment or macroeconomic indicators N_t; $b_t = (C_t, H_t) \in \mathcal{Y}$ represents the *trading account balance*, including available cash C_t at time t, and number of stock shares H_t.

Action. The agent chooses from a discrete set of trading actions $\mathcal{A} = \{\texttt{Sell} : -1, \texttt{Hold} : 0, \texttt{Buy} : 1\}$, where $a_t = -1$ denotes selling all holdings (liquidate the portfolio), $a_t = 0$ denotes holding (no trading action), and $a_t = 1$ represents buying with all available cash (convert all cash into stocks).

State Transition. The state transition dynamics are governed by a stochastic process $s_{t+1} \sim \mathcal{T}(\cdot | s_t, a_t)$. The trading account evolves according to the following equations:

- If `Sell`: $C_{t+1} = C_t + H_t P_{t+1}, \quad H_{t+1} = 0.$
- If `Hold`: $C_{t+1} = C_t, \quad H_{t+1} = H_t.$
- If `Buy`: $C_{t+1} = 0, \quad H_{t+1} = H_t + \frac{C_t}{P_{t+1}}.$

Reward. The agent receives a reward based on the daily trading profit & loss (PnLs):

$$R(s_t, a_t) = SR_t - SR_{t-1},$$

where SR_t denotes the Sharpe ratio at day t, computed by using the historical PnL from time 0 to time t. Moreover, PnL at time t is calculated as

$$pnl_t := (C_t - C_{t-1}) + (H_t P_t - H_{t-1} P_{t-1}).$$

Then, the Sharpe ratio SR_t at time t can be calculated as:

$$SR_t := \frac{\mathbb{E}[PnL_1, \cdots, PnL_t] - r_f}{\sigma[PnL_1, \cdots, PnL_t]}, \tag{1}$$

where $\mathbb{E}[PnL_1, \cdots, PnL_t]$ is the sample average of daily PnL up to time t, r_f is the risk-free rate, and $\sigma[PnL_1, \cdots, PnL_t]$ is the sample standard deviation of daily PnL up to time t.

The goal is to find an admissible policy π to maximize the expected value of cumulative discounted reward, i.e.,

$$\max_{\pi} V^{\pi}(s) = \mathop{\mathbb{E}}_{\substack{s_0 = s, a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{T} \gamma^t R_t \right], \tag{2}$$

where R_t is a shortened version $R(s_t, a_t)$ and $\gamma \in (0, 1]$ is the discount factor controlling the importance of future rewards.

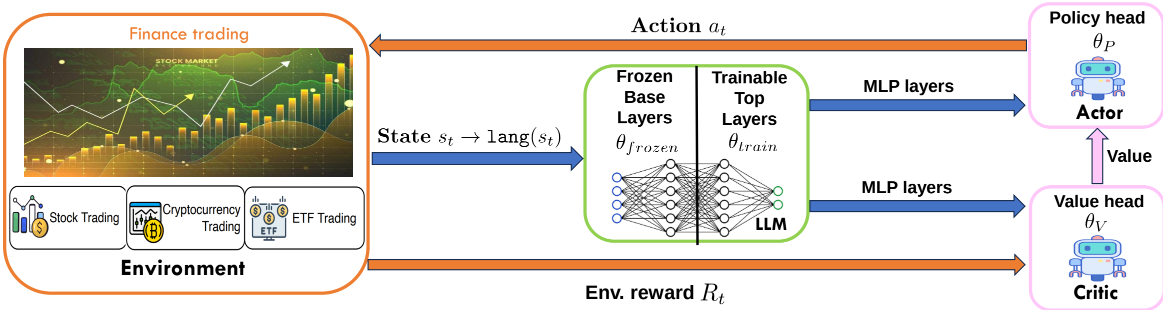

Figure 2: The FLAG-TRADER pipeline for financial trading, utilizing an LLM-based actor-critic architecture. The LLM consists of **frozen base layers** $\theta_{\mathtt{frozen}}$ that retain pre-trained knowledge and **trainable top layers** $\theta_{\mathtt{train}}$ for financial decision-making. Both the POLICY_NET and VALUE_NET share these trainable layers while maintaining separate *policy head* θ_P and *value head* θ_V, which are updated by policy gradient method.

Our goal is to train an LLM agent parameterized by θ to find the optimized policy π_θ for (2), i.e.,

$$a_t \sim \pi_\theta(\cdot|s_t) = \mathrm{LLM}(\mathtt{lang}(s_t); \theta), \qquad (3)$$

where $\mathtt{lang}(s_t)$ are the prompts generated by converting state s_t into structured text. The proposed pipeline is illustrated in Figure 1.

4 FLAG-TRADER

To tackle the challenge of directly fine-tuning an LLM for both alignment and decision-making, we introduce FLAG-TRADER, a fused LLM-agent and RL framework for financial stock trading. In FLAG-TRADER, a partially fine-tuned LLM serves as the policy network, leveraging its pre-trained knowledge while adapting to the financial domain through parameter-efficient fine-tuning, as shown in Figure 2. The model processes financial information using a textual state representation, allowing it to interpret and respond to market conditions effectively. Instead of fine-tuning the entire network, only a subset of the LLM's parameters is trained, striking a balance between adaptation and knowledge retention. In the following, we will present the prompt input design and the detailed architecture of FLAG-TRADER.

4.1 Prompt Input Design

The first stage of the pipeline involves designing a robust and informative prompt, denoted as $\mathtt{lang}(s_t)$, which is constructed based on the current state s_t to guide the LLM in making effective trading decisions. The prompt is carefully structured to encapsulate essential elements that provide context and ensure coherent, actionable outputs. It consists of four key components: a *task description*, which defines the financial trading objective, outlining the problem domain and expected actions; a *legible action space*, specifying the available trading decisions (`Sell`," `Hold`," "Buy"); a *current state representation*, incorporating market indicators, historical price data, and portfolio status to contextualize the decision-making process; and an *output action*, which generates an executable trading decision. This structured prompt ensures that the LLM receives comprehensive input, enabling it to produce well-informed and actionable trading strategies, as illustrated in Figure 3.

Financial Stock Trading

Task: Assist traders in making optimal buy, hold, or sell decisions for a stock portfolio. The goal is to maximize long-term returns while managing risk. The agent should execute buy if the stock is undervalued based on historical trends, sell if it is overvalued or risk is high, and hold if market conditions are uncertain. Trades should minimize transaction costs and align with market momentum indicators.

Legible Actions: Choose from "{Buy, Sell, Hold}" based on market conditions and risk assessment.

Current State: *Given the following market data:*

```
{{
 "historical_prices": [{', '.join([str(val)[:6] for val in obs.cpu().tolist()[0]])}],
 "account_status": {{
  "cash_balance": {info['cash']},
  "asset_position": {info['asset']},
  "total_account_value": {info['total']}
 }},
 "previous_decision_metrics": {{
  "recent_rewards": {[str(val)[:6] for val in info['rewards_memory'][-10:]]},
  "net_values": {[str(val)[:10] for val in info['asset_memory'][-10:]]},
  "actions": {info.get('action_memory', [])[-10:]},
 }}
}}
```

Output Action. Format your answer as a JSON as in the following examples: {'Action': Sell}, {'Action': Buy}, {'Action': Hold}.

Figure 3: The format of input prompt. It contains the task description, the legible action set, the current state description, and the output action format.

4.2 FLAG-TRADER Architecture

To incorporate parameter-efficient fine-tuning into the policy gradient framework, we partition the intrinsic parameters of the LLM into two distinct components: the frozen parameters inherited from pretraining, denoted as $\theta_{\texttt{forzen}}$, and the trainable parameters, denoted as $\theta_{\texttt{train}}$. This separation allows the model to retain general language understanding while adapting to financial decision-making with minimal computational overhead. Building upon this LLM structure, we introduce a policy network and a value network, both of which leverage the trainable top layers of the LLM for domain adaptation while sharing the frozen layers for knowledge retention. The overall architecture is illustrated in Figure 2.

4.2.1 Policy Network Design

The policy network is responsible for generating an optimal action distribution over the trading decision space $\mathcal{A}$, conditioned on the observed market state. It consists of three main components:

State Encoding. To effectively process financial data using the LLM, the numerical market state s is first converted into structured text using a predefined template[1]

$$\texttt{lang}(s) = \text{"Price: \$}p\text{, Vol: }v\text{, RSI: }r\text{,..."}. \quad (4)$$

This transformation enables the model to leverage the LLM's textual reasoning capabilities, allowing it to understand and infer trading decisions in a structured, language-based manner.

LLM Processing. The tokenized text representation of the state is then passed through the LLM backbone, which consists of: 1) **Frozen layers** (preserve general knowledge): Token embeddings $E = \text{Embed}(\texttt{lang}(s))$ pass through LLM frozen layers, i.e.,

$$h^{(1)} = \text{LLM}_{1:N}(E; \theta_{\texttt{frozen}}). \quad (5)$$

These layers preserve general knowledge acquired from pretraining, ensuring that the model maintains a strong foundational understanding of language and reasoning. 2) **Trainable layers** (domain-specific adaptation): The output from the frozen layers is then passed through the trainable layers, which are fine-tuned specifically for financial decision-making, i.e.,

$$h^{(2)} = \text{LLM}_{N+1:N+M}(h^{(1)}; \theta_{\text{train}}). \quad (6)$$

This structure enables efficient adaptation to the financial domain without modifying the entire LLM, significantly reducing training cost while maintaining performance.

Policy Head. Finally, the processed representation is fed into the policy head, which outputs a probability distribution over the available trading actions according to

$$\texttt{logits} = \text{POLICY\_NET}(h^{(2)}, \theta_P) \in \mathbb{R}^{|\mathcal{A}|}, \quad (7)$$

where θ_P is the parameter of POLICY_NET, with action masking for invalid trades:

$$\pi(a|s) = \begin{cases} 0 & a \notin \mathcal{A}, \\ \frac{\exp(\texttt{logits}(a))}{\sum_{a' \in \mathcal{A}} \exp(\texttt{logits}(a'))} & \text{otherwise.} \end{cases} \quad (8)$$

This ensures that actions outside the valid set $\mathcal{A}$ (e.g., selling when no stocks are held) have zero probability, preventing invalid execution.

4.2.2 Value Network Design

The value network serves as the critic in the RL framework, estimating the expected return of a given state to guide the policy network's optimization. To efficiently leverage the shared LLM representation, the value network shares the same backbone as the policy network, processing the textual state representation through the frozen and trainable layers (4)–(6). This design ensures efficient parameter utilization while maintaining a structured and informative state encoding. After passing through the LLM processing layers, the output $h^{(2)}$ is fed into a separate value prediction head, which maps the extracted features to a scalar value estimation:

$$V(s) = \text{VALUE\_NET}(h^{(2)}, \theta_V) \in \mathbb{R}^1, \quad (9)$$

where θ_V is the parameter of VALUE_NET.

4.3 Online Policy Gradient Learning

The policy and value networks in FLAG-TRADER are trained using an online policy gradient approach, ensuring that the model continuously refines its decision-making ability. The learning process follows an iterative cycle of state observation, action generation, reward evaluation, and policy optimization. The parameters of the model are updated using stochastic gradient descent (SGD), leveraging the computed policy and value losses to drive optimization.

At each training step, we define two key loss functions, i.e., *policy loss* $\mathcal{L}_P$: measures how well the policy network aligns with the expected

[1]To simplify notation, we use $\texttt{lang}(s_t)$ to represent both the state encoding and the prompt, acknowledging this slight abuse of notation for convenience.

advantage-weighted log probability of actions; *value loss* $\mathcal{L}_V$: ensures that the value network accurately estimates the expected return.

Remark 4.1. The definitions of *policy loss* and *value loss* may vary across different actor-critic (AC) algorithms. Here, we present a general formulation for clarity and ease of expression. Notably, our framework is designed to be flexible and adaptable, making it compatible with a wide range of AC algorithms.

Based on these loss functions, the model updates the respective network parameters using backpropagation as follows.

Update Policy Head. The policy network parameters θ_P are updated via SGD to minimize the *policy loss* $\mathcal{L}_P$

$$\theta_P \leftarrow \theta_P - \eta \nabla_{\theta_P} \mathcal{L}_P, \tag{10}$$

where η is the learning rate for updating policy head θ_P.

Update Value Head. The value network parameters θ_V are optimized via SGD to minimize the temporal difference (TD) error over policy loss $\mathcal{L}_V$

$$\theta_V \leftarrow \theta_V - \eta \nabla_{\theta_V} \mathcal{L}_V. \tag{11}$$

Update Trainable LLM Layers. The trainable LLM parameters $\theta_{\texttt{train}}$ are updated via SGD jointly based on both the policy and value losses, i.e., $\mathcal{L}_P$ and $\mathcal{L}_V$, allowing the shared LLM representation to align with optimal decision-making:

$$\theta_{\texttt{train}} \leftarrow \theta_{\texttt{train}} - \beta \nabla_{\theta_{\texttt{train}}} (\mathcal{L}_P + \mathcal{L}_V), \tag{12}$$

where β is the learning rate for LLM parameter $\theta_{\texttt{train}}$.

The updates in (10)–(12) are performed iteratively until the stopping criteria are met, as outlined in Algorithm 1. This iterative learning process effectively balances exploration and exploitation, enhancing policy performance while maintaining stability. To mitigate overfitting and policy divergence, we employ Proximal Policy Optimization (PPO), which constrains updates by limiting the divergence from previous policies, ensuring more controlled and reliable learning. The detailed procedure of how to compute *policy loss* $\mathcal{L}_P$ and *value loss* $\mathcal{L}_P$ can be found in Appendix A.

Algorithm 1 FLAG-TRADER

Require: Pre-trained LLM with parameter $\theta := (\theta_{\texttt{frozen}}, \theta_{\texttt{train}})$, environment dynamics $\mathcal{T}$, reward function $\mathcal{R}$;
Initialize policy network θ_P and value network θ_V with shared LLM trainable layers $\theta_{\texttt{train}}$;
Initialize experience replay buffer $B \leftarrow \emptyset$
for iteration $t = 1, 2, \ldots,$ **do**
  Fetch the current state s_t from the environment and construct an input prompt $\texttt{lang}(s_t)$;
  Pass prompt $\texttt{lang}(s_t)$ through LLM;
  POLICY_NET outputs a_t from action space {“buy,” “sell,” “hold”} based on (8);
  Execute action a_t in the environment and observe reward $r(s_t, a_t)$ and transition to new state s_{t+1};
  Store experience tuple (s_t, a_t, r_t, s_{t+1}) in replay buffer B;
  if $t \mod \tau = 0$ **then**
    Update policy head θ_P according to (10);
    Update value head θ_V according to (11);
    Update the trainable LLM layers $\theta_{\texttt{train}}$ according to (12).
  end if
end for
Return: Fine-tuned POLICY_NET(θ_P).

5 Experiments

This section describes the overall experimental design and environmental setup for comparing the performance of different trading agents under consistent conditions.

5.1 Experiment Setup

For our single-asset trading tasks, we adopt two baselines: the buy-and-hold strategy and the LLM-based trading agent from INVESTORBENCH (Li et al., 2024a), which integrates 13 proprietary or open-source large language models. Our proposed model, FLAG-TRADER (built on a 135M-parameter LLM), is then evaluated against these baselines for a comprehensive performance comparison.

We focus on five stocks and one crypto: Microsoft Corporation (MSFT), Johnson & Johnson (JNJ), UVV Corporation (UVV), Honeywell International Inc. (HON), Tesla, Inc. (TSLA) and Bitcoin (BTC). As summarized in Table 1, each agent's performance is measured across these assets. All language models use a temperature of 0.6 during inference to balance consistency and creativity in their responses.

We report four metrics-Composite Return (CR), Sharpe Ratio (SR), Annualized Volatility (AV), and Maximum Drawdown (MDD), and select final re-

Table 1: Performance of single-asset trading with different LLMs as backbone model across six assets.

Model	MSFT				JNJ				UVV			
	CR↑	SR↑	AV↓	MDD↓	CR↑	SR↑	AV↓	MDD↓	CR↑	SR↑	AV↓	MDD↓
Buy & Hold	15.340	1.039	24.980	9.428	13.895	1.343	17.500	9.847	36.583	2.112	29.299	15.406
Financial Domain Models												
Palmyra-Fin-70B	14.697	0.897	27.518	9.428	5.748	0.450	19.317	9.367	37.875	2.039	31.200	15.967
Proprietary Models												
GPT-o1-preview	17.184	0.962	30.000	9.428	13.561	1.086	20.864	9.847	41.508	2.147	32.479	9.633
GPT-4	16.654	0.932	30.022	9.428	13.712	1.103	20.894	9.860	31.791	1.640	32.567	10.434
GPT-4o	12.461	0.924	22.653	6.647	9.099	0.875	17.471	7.169	8.043	0.496	27.241	14.889
Open-Source Models												
Qwen2.5-72B-Instruct	7.421	0.588	21.238	6.973	14.353	1.140	20.995	9.812	37.178	1.822	34.223	13.365
Llama-3.1-70B-Instruct	17.396	1.335	21.892	7.045	13.868	1.121	20.779	9.825	35.981	1.728	34.986	15.406
DeepSeek-67B-Chat	13.941	0.834	28.081	7.850	14.426	1.185	20.450	9.825	29.940	1.481	33.964	15.407
Yi-1.5-34B-Chat	22.093	1.253	29.613	9.428	14.004	1.180	19.938	9.847	20.889	1.020	34.417	14.936
Qwen2.5-32B-Instruct	-0.557	-0.041	22.893	8.946	2.905	0.292	16.725	7.169	-1.623	-0.097	27.973	17.986
DeepSeek-V2-Lite (15.7B)	11.904	0.694	28.796	16.094	-7.482	-0.670	18.773	17.806	33.560	1.703	33.099	12.984
Yi-1.5-9B-Chat	19.333	1.094	29.690	9.428	18.606	1.611	19.409	10.986	49.415	2.410	34.446	11.430
Llama-3.1-8B-Instruct	22.703	1.322	28.855	7.385	13.988	1.486	20.460	9.969	41.108	1.981	34.866	16.429
Qwen-2.5-Instruct-7B	-10.305	-0.724	23.937	23.371	21.852	0.980	37.425	9.573	11.752	0.853	22.988	15.451
FLAG-TRADER												
SmolLM2-135M-Instruct	20.106	1.373	24.932	9.428	33.724	3.344	17.174	9.320	46.799	1.463	67.758	35.039

[1] The Buy & Hold strategy is a passive investment approach commonly used as a baseline strategy, where an investor purchases stocks and holds onto them for an extended period regardless of market fluctuations.
[2] An upward arrow (↑) next to a metric indicates that higher values signify better performance, while a downward arrow (↓) indicates that lower values are preferable.
[3] The numbers highlighted in red indicate the best-performing outcomes for the corresponding metrics.

sults from the test trajectory corresponding to the median of these metrics. If the median values arise from different epochs, we prioritize the run producing the median SR. Due to varying data availability, warm-up and test periods may differ. For the trading tasks of five stocks, the warm-up period is July 1, 2020, to September 30, 2020, and the test period is October 1, 2020, to May 6, 2021. On the other hand, the warm-up period of BTC trading is from 2023-02-11 to 2023-04-04 and the test period is from 2023-04-05 to 2023-11-05.

We deploy LLMs using the vllm framework, with configurations depending on model size. Small-scale models (under 10B parameters) run on two RTX A6000 GPUs (48GB each), mid-scale models (10B–65B parameters) require four RTX A6000 GPUs, and large-scale models (over 65B parameters) use eight A100 GPUs (80GB each). These setups provide sufficient resources for both inference and training, enabling a fair comparison of trading performance across different assets. FLAG-TRADER is trained by using PPO algorithm, which is detailed in Algorithm 2 in Appendix A.

5.2 Evaluation Metrics

We use four widely recognized financial metrics (Hull, 2007) to evaluate and compare the investment performance of various LLM backbones across different tasks: Cumulative Return (CR), Sharpe Ratio (SR), Annualized Volatility (AV), and Maximum Drawdown (MDD). As CR and SR focus more on long-term gains and risk-adjusted returns, they are typically considered more important than AV and MDD for assessing asset trading performance. Accordingly, we treat CR and SR as our primary metrics for the final evaluation.

Cumulative Return (CR) % measures the total value change of an investment over time by summing logarithmic return calculated from daily PnL:

$$\textbf{CR} = \sum_{t=1}^{T} \log\left(1 + \frac{PnL_t}{C_{t-1} + H_{t-1}P_{t-1}}\right), \quad (13)$$

where PnL_t is the PnL at time t, and $C_{t-1} + H_{t-1}P_{t-1}$ is the account balance at time $t-1$. Notice that higher values indicate better strategy effectiveness.

Sharpe Ratio (SR) assesses risk-adjusted returns by dividing the average excess return (R_p) over the risk-free rate (r_f) by its volatility (σ_p):

$$\textbf{SR} = \frac{R_p - r_f}{\sigma_p}. \quad (14)$$

Notice that higher ratios signify better performance.

Annualized Volatility (AV) % and Daily Volatility (DV) % quantify return fluctuations; AV is

Table 2: Performance of single-asset trading with different LLMs as backbone model across six assets

Model	HON				TSLA				BTC			
	CR↑	SR↑	AV↓	MDD↓	CR↑	SR↑	AV↓	MDD↓	CR↑	SR↑	AV↓	MDD↓
Buy & Hold	33.256	2.347	23.967	9.195	39.244	0.869	75.854	37.975	21.821	0.683	37.426	20.796
Financial Domain Models												
Palmyra-Fin-70B	20.016	1.464	22.974	6.824	-6.661	-0.222	50.379	25.820	-20.812	-1.212	20.036	27.782
Proprietary Models												
GPT-o1-preview	13.162	0.776	28.511	11.558	34.499	0.796	72.822	35.490	34.060	1.114	35.846	17.075
GPT-4	34.342	2.005	28.779	9.195	45.246	1.190	63.896	25.031	22.396	0.828	31.699	17.206
GPT-4o	38.540	2.418	26.782	8.979	45.946	1.348	57.281	21.631	14.330	0.532	31.304	17.278
Open-Source Models												
Qwen2.5-72B-Instruct	34.309	2.000	28.779	9.292	39.112	1.075	61.136	26.985	0.549	0.325	1.979	0.897
Llama-3.1-70B-Instruct	43.944	2.646	27.903	8.993	37.545	0.891	70.815	29.813	20.440	0.758	31.604	17.813
DeepSeek-67B-Chat	32.536	1.909	28.628	10.782	35.647	0.885	67.660	33.359	28.307	0.891	37.219	17.944
Yi-1.5-34B-Chat	30.743	1.823	28.335	9.195	35.364	0.808	73.561	35.490	13.620	0.434	36.778	22.790
Qwen2.5-32B-Instruct	26.332	1.980	22.348	5.261	21.336	0.729	49.157	20.704	11.566	0.869	15.608	7.984
DeepSeek-V2-Lite (15.7B)	16.686	0.974	28.771	16.806	31.458	0.744	68.524	35.404	4.804	0.153	36.846	20.562
Yi-1.5-9B-Chat	29.028	1.700	28.682	12.588	31.350	0.703	74.895	37.975	7.953	0.253	36.799	26.545
Llama-3.1-8B-Instruct	39.079	2.320	28.299	10.341	35.622	0.832	71.936	36.383	20.521	0.646	37.240	21.104
Qwen-2.5-Instruct-7B	4.291	0.285	24.933	14.156	41.203	0.925	74.862	37.975	19.477	0.612	37.289	20.796
FLAG-TRADER												
SmolLM2-135M-Instruct	34.342	2.429	23.913	10.872	50.394	1.362	64.004	37.975	45.511	1.734	30.903	24.440

[1] The Buy & Hold strategy is a passive investment approach commonly used as a baseline strategy, where an investor purchases stocks and holds onto them for an extended period regardless of market fluctuations.
[2] An upward arrow (↑) next to a metric indicates that higher values signify better performance, while a downward arrow (↓) indicates that lower values are preferable.
[3] The numbers highlighted in red indicate the best-performing outcomes for the corresponding metrics.

derived by scaling DV (*standard deviation of daily logarithmic returns*) by the square root of the annual trading days (252):

$$\mathbf{AV} = \mathbf{DV} \times \sqrt{252}. \tag{15}$$

This metric highlights potential return deviations across the year.

Max Drawdown (MDD) % calculates the largest drop from peak to trough of the value of balance account:

$$\mathbf{MDD} = \max \left(\frac{V_{\text{peak}} - V_{\text{trough}}}{V_{\text{peak}}} \right). \tag{16}$$

Notice that lower values indicate lesser risk and higher strategy robustness.

5.3 Experimental Results

- **FLAG-Trader achieves superior stock trading performance.** In contrast to the baseline agent, which is built on a purely LLM-agentic framework, FLAG-TRADER incorporates a RL-based post-training phase that enhances its decision-making capabilities. This additional training allows the model to better adapt to dynamic and complex market conditions. As shown in Table 1 and Table 2, FLAG-TRADER consistently surpasses the baseline across a range of evaluation metrics. These results highlight its improved adaptability, stronger optimization behavior, and overall robustness when deployed in diverse and unpredictable financial environments.

- **FLAG-TRADER enables small-scale models to surpass large-scale counterparts.** While increasing model size generally enhances financial decision-making and robustness— as seen with large proprietary models (e.g., GPT-o1-preview) in the baseline framework-FLAG-TRADER leverages an RL-based training pipeline to enable a 135M-parameter open-source model to outperform significantly larger models in financial trading tasks. This demonstrates that a well-designed training strategy can bridge or even surpass the performance gap typically associated with model scale.

6 Conclusion

In this paper, we introduced FLAG-TRADER, a novel framework that integrates LLMs with RL for financial trading. In particular, FLAG-TRADER leverages LLMs as policy networks, allowing for natural language-driven decision-making while benefiting from reward-driven optimization through RL fine-tuning. Our framework enables small-scale LLMs to surpass larger proprietary models by efficiently adapting to market conditions via a structured reinforcement learning approach. Through extensive experiments across multiple stock trading scenarios, we demonstrated that FLAG-TRADER consistently outperforms baseline methods, including LLM-agentic frameworks and conventional RL-based trading agents. These results highlight the potential of integrating LLMs with RL to achieve adaptability in financial decision-making.

Limitations and Potential Risk

Despite its promising results, FLAG-TRADER has several limitations. First, while our approach enhances the decision-making ability of LLMs, it remains computationally expensive, particularly when fine-tuning on large-scale datasets. Reducing computational overhead while maintaining performance is an important direction for future research. Second, financial markets exhibit high volatility and non-stationarity, posing challenges for long-term generalization. Future work should explore techniques such as continual learning or meta-learning to enhance model adaptability in evolving conditions. Third, while FLAG-TRADER effectively integrates textual and numerical data, its reliance on structured prompts could introduce biases in decision-making. Improving prompt design or exploring retrieval-augmented methods may further enhance robustness. Lastly, real-world trading requires stringent risk management, and FLAG-TRADER optimizes for financial returns without explicitly incorporating risk-sensitive constraints. Extending the framework to integrate risk-aware objectives and dynamic portfolio optimization could provide more robust and practical trading solutions.

Acknowledgements

The authors acknowledge UFIT Research Computing, NVAITC, and HPG for providing computational resources and support that have contributed to the research results reported in this publication. URL: http://www.rc.ufl.edu. Sophia Ananiadou is supported by the project JPNP20006 from New Energy and Industrial Technology Development Organization (NEDO). Keyi Wang and Xiao-Yang Liu Yanglet acknowledge the support from Columbia's SIRS and STAR Program, as well as The Tang Family Fund for Research Innovations in FinTech, Engineering, and Business Operations. Xiao-Yang Liu Yanglet also acknowledges the support from a NSF IUCRC CRAFT center research grant (CRAFT Grant 22017) for this research. The opinions expressed in this publication do not necessarily represent the views of NSF IUCRC CRAFT. Sophia Ananiadou has also been partially supported by project MIS 5154714 of the National Recovery and Resilience Plan Greece 2.0 funded by the European Union under the Next Generation EU Program.

References

Dogu Araci. 2019. Finbert: Financial sentiment analysis with pre-trained language models. *arXiv preprint arXiv:1908.10063*.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. 2021. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097.

Renato Arantes de Oliveira, Heitor S Ramos, Daniel Hasan Dalip, and Adriano César Machado Pereira. 2020. A tabular sarsa-based stock market agent. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–8.

Xiang Deng, Vasilisa Bashlovkina, Feng Han, Simon Baumgartner, and Michael Bendersky. 2023. What do llms know about financial markets? a case study on reddit market sentiment analysis. In *Companion Proceedings of the ACM Web Conference 2023*, pages 107–110.

Xiu Gao and Laiwan Chan. 2000. An algorithm for trading and portfolio management using q-learning and sharpe ratio maximization. In *Proceedings of the international conference on neural information processing*, pages 832–837. Citeseer.

Ben Hambly, Renyuan Xu, and Huining Yang. 2023. Recent advances in reinforcement learning in finance. *Mathematical Finance*, 33(3):437–503.

Allen H Huang, Hui Wang, and Yi Yang. 2023. Finbert: A large language model for extracting information from financial text. *Contemporary Accounting Research*, 40(2):806–841.

Yuling Huang, Xiaoxiao Wan, Lin Zhang, and Xiaoping Lu. 2024. A novel deep reinforcement learning framework with bilstm-attention networks for algorithmic trading. *Expert Systems with Applications*, 240:122581.

John Hull. 2007. *Risk Management and Financial Institutions*. John Wiley & Sons.

O Jangmin, Jongwoo Lee, Jae Won Lee, and Byoung-Tak Zhang. 2006. Adaptive stock trading with dynamic asset allocation using reinforcement learning. *Information Sciences*, 176(15):2121–2147.

Gyeeun Jeong and Ha Young Kim. 2019. Improving financial trading decisions using deep q-learning: Predicting the number of shares, action strategies, and transfer learning. *Expert Systems with Applications*, 117:125–138.

Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, and 1 others. 2023. Time-llm: Time series forecasting by reprogramming large language models. *arXiv preprint arXiv:2310.01728*.

Haohang Li, Yupeng Cao, Yangyang Yu, Shashidhar Reddy Javaji, Zhiyang Deng, Yueru He, Yuechen Jiang, Zining Zhu, Koduvayur Subbalakshmi, Guojun Xiong, and 1 others. 2024a. Investorbench: A benchmark for financial decision-making tasks with llm-based agent. *arXiv preprint arXiv:2412.18174*.

Yang Li, Yangyang Yu, Haohang Li, Zhi Chen, and Khaldoun Khashanah. 2023. Tradinggpt: Multi-agent system with layered memory and distinct characters for enhanced financial trading performance. *arXiv preprint arXiv:2309.03736*.

Yuan Li, Bingqiao Luo, Qian Wang, Nuo Chen, Xu Liu, and Bingsheng He. 2024b. Cryptotrade: A reflective llm-based agent to guide zero-shot cryptocurrency trading. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1094–1106.

Zhipeng Liang, Hao Chen, Junhao Zhu, Kangkang Jiang, and Yanran Li. 2018. Adversarial deep reinforcement learning in portfolio management. *arXiv preprint arXiv:1808.09940*.

Xiao-Yang Liu, Guoxuan Wang, and Daochen Zha. 2023. Fingpt: Democratizing internet-scale data for financial large language models. *arXiv preprint arXiv:2307.10485*.

Xiao-Yang Liu, Ziyi Xia, Jingyang Rui, Jiechao Gao, Hongyang Yang, Ming Zhu, Christina Wang, Zhaoran Wang, and Jian Guo. 2022. Finrl-meta: Market environments and benchmarks for data-driven financial reinforcement learning. *Advances in Neural Information Processing Systems*, 35:1835–1849.

Arman Khadjeh Nassirtoussi, Saeed Aghabozorgi, Teh Ying Wah, and David Chek Ling Ngo. 2014. Text mining for market prediction: A systematic review. *Expert Systems with Applications*, 41(16):7653–7670.

Raj Sanjay Shah, Kunal Chawla, Dheeraj Eidnani, Agam Shah, Wendi Du, Sudheer Chava, Natraj Raman, Charese Smiley, Jiaao Chen, and Diyi Yang. 2022. When flue meets flang: Benchmarks and large pre-trained language model for financial domain. *arXiv preprint arXiv:2211.00083*.

Si Shi, Jianjun Li, Guohui Li, and Peng Pan. 2019. A multi-scale temporal feature aggregation convolutional neural network for portfolio management. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1613–1622.

Shuo Sun, Rundong Wang, and Bo An. 2023. Reinforcement learning for quantitative trading. *ACM Transactions on Intelligent Systems and Technology*, 14(3):1–29.

Andrew Szot, Max Schwarzer, Harsh Agrawal, Bogdan Mazoure, Rin Metcalf, Walter Talbott, Natalie Mackraz, R Devon Hjelm, and Alexander T Toshev. 2023. Large language models as generalizable policies for embodied tasks. In *The Twelfth International Conference on Learning Representations*.

Kieran Wood, Sven Giegerich, Stephen Roberts, and Stefan Zohren. 2021. Trading with the momentum transformer: An intelligent and interpretable architecture. *arXiv preprint arXiv:2112.08534*.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*.

Qianqian Xie, Weiguang Han, Zhengyu Chen, Ruoyu Xiang, Xiao Zhang, Yueru He, Mengxi Xiao, Dong Li, Yongfu Dai, Duanyu Feng, and 1 others. 2024a. The finben: An holistic financial benchmark for large language models. *arXiv preprint arXiv:2402.12659*.

Qianqian Xie, Weiguang Han, Xiao Zhang, Yanzhao Lai, Min Peng, Alejandro Lopez-Lira, and Jimin Huang. 2023. Pixiu: A large language model, instruction data and evaluation benchmark for finance. *arXiv preprint arXiv:2306.05443*.

Qianqian Xie, Dong Li, Mengxi Xiao, Zihao Jiang, Ruoyu Xiang, Xiao Zhang, Zhengyu Chen, Yueru He, Weiguang Han, Yuzhe Yang, and 1 others. 2024b. Open-finllms: Open multimodal large language models for financial applications. *arXiv preprint arXiv:2408.11878*.

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

Hongyang Yang, Boyu Zhang, Neng Wang, Cheng Guo, Xiaoli Zhang, Likun Lin, Junlin Wang, Tianyu Zhou, Mao Guan, Runjia Zhang, and 1 others. 2024. Finrobot: An open-source ai agent platform for financial applications using large language models. *arXiv preprint arXiv:2405.14767*.

Yi Yang, Yixuan Tang, and Kar Yan Tam. 2023b. Investlm: A large language model for investment using financial domain instruction tuning. *arXiv preprint arXiv:2309.13064*.

Yi Yang, Mark Christopher Siy Uy, and Allen Huang. 2020. Finbert: A pretrained language model for financial communications. *arXiv preprint arXiv:2006.08097*.

Yunan Ye, Hengzhi Pei, Boxin Wang, Pin-Yu Chen, Yada Zhu, Ju Xiao, and Bo Li. 2020. Reinforcement-learning based portfolio management with augmented asset movement prediction states. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 1112–1119.

Yangyang Yu, Haohang Li, Zhi Chen, Yuechen Jiang, Yang Li, Denghui Zhang, Rong Liu, Jordan W Suchow, and Khaldoun Khashanah. 2024. Finmem: A performance-enhanced llm trading agent with layered memory and character design. In *Proceedings of the AAAI Symposium Series*, volume 3, pages 595–597.

Yangyang Yu, Zhiyuan Yao, Haohang Li, Zhiyang Deng, Yuechen Jiang, Yupeng Cao, Zhi Chen, Jordan Suchow, Zhenyu Cui, Rong Liu, and 1 others. 2025. 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.

Yuexiang Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Shengbang Tong, Yifei Zhou, Alane Suhr, Saining Xie, Yann LeCun, Yi Ma, and 1 others. 2024. Fine-tuning large vision-language models as decision-making agents via reinforcement learning. *arXiv preprint arXiv:2405.10292*.

Wentao Zhang, Lingxuan Zhao, Haochong Xia, Shuo Sun, Jiaze Sun, Molei Qin, Xinyi Li, Yuqing Zhao, Yilei Zhao, Xinyu Cai, and 1 others. 2024. Finagent: A multimodal foundation agent for financial trading: Tool-augmented, diversified, and generalist. *arXiv preprint arXiv:2402.18485*.

Yuzhe Zhang and Hong Zhang. 2023. Finbert–mrc: financial named entity recognition using bert under the machine reading comprehension paradigm. *Neural Processing Letters*, 55(6):7393–7413.

Zihao Zhang, Stefan Zohren, and Stephen J. Roberts. 2019. Deep reinforcement learning for trading. In *The Journal of Financial Data Science*.

A Additional Algorithmic Details: FLAG-TRADER with PPO

In this section, we outline a detailed procedure for training the FLAG-TRADER architecture via PPO, where the POLICY_NET (actor) and the VALUE_NET (critic) share a subset of trainable parameters from a LLM, with $\theta = \big(\theta_{\texttt{train}}, \theta_P, \theta_V\big)$. We define $\theta_{policy} = \big(\theta_{\texttt{train}}, \theta_P\big)$ and $\theta_{value} = \big(\theta_{\texttt{train}}, \theta_V\big)$ for simplicity.

Advantage Estimation. We use the Generalized Advantage Estimation (GAE) to compute the advantage function A_t:

$$A_t \;=\; \sum_{k=0}^{T-1} (\gamma\lambda)^k \big[r_{t+k} + \gamma V_{\theta_{value}}(s_{t+k+1}) - V_{\theta_{value}}(s_{t+k})\big], \tag{17}$$

where γ is the discount factor, and λ is the GAE parameter.

Probability Ratio. Let $\theta_{policy,\mathrm{old}}$ denote the parameters before the current update. The PPO probability ratio is

$$r_t(\theta_{policy}) \;=\; \frac{\pi_{\theta_{policy}}(a_t \mid s_t)}{\pi_{\theta_{policy,\mathrm{old}}}(a_t \mid s_t)}. \tag{18}$$

PPO Clipped Objective. PPO clips this ratio to prevent overly large updates. The surrogate objective is

$$\mathcal{L}_P(\theta_{policy}) \;=\; \mathbb{E}_t\Big[\min\big(r_t(\theta_{policy})\, A_t,\ \mathrm{clip}\big(r_t(\theta_{policy}),\, 1-\varepsilon,\, 1+\varepsilon\big)\, A_t\big)\Big], \tag{19}$$

where ε is a hyperparameter.

Value Function Loss. The critic (value network) is updated by minimizing the difference between the predicted value $V_{\theta_{value}}(s_t)$ and the target return R_t. A common choice is:

$$\mathcal{L}_V(\theta_{value}) \;=\; \mathbb{E}_t\Big[(V_{\theta_{value}}(s_t) - R_t)^2\Big]. \tag{20}$$

Combined Loss. We often add an entropy term to encourage exploration, yielding the overall objective:

$$\mathcal{L}_{\mathrm{total}}(\theta) \;=\; -\,\mathcal{L}_P(\theta_{policy}) \;+\; c_1\, \mathcal{L}_V(\theta_{value}) \;-\; c_2\, \mathcal{H}\big(\pi_{\theta_{policy}}\big), \tag{21}$$

where c_1 and c_2 are weighting coefficients, and $\mathcal{H}(\pi_{\theta_{policy}})$ represents the policy entropy.

Parameter Updates. At each iteration, we apply gradient descent on the total loss:

$$\theta_P \leftarrow \theta_P \;-\; \eta\, \nabla_{\theta_P} \mathcal{L}_P, \tag{22}$$

$$\theta_V \leftarrow \theta_V \;-\; \eta\, \nabla_{\theta_V} \mathcal{L}_V, \tag{23}$$

$$\theta_{\texttt{train}} \leftarrow \theta_{\texttt{train}} - \beta\, \nabla_{\theta_{\mathrm{train}}} \mathcal{L}_{\mathrm{total}}, \tag{24}$$

where η and β are learning rates for the policy head, value head, and trainable LLM layers respectively. The algorithm is summerized in Algorithm 2.

Algorithm 2 FLAG-TRADER with PPO

Input: Pre-trained LLM parameters $(\theta_{\texttt{frozen}}, \theta_{\texttt{train}})$; actor parameters θ_P; critic parameters θ_V; environment $\mathcal{E}$; discount factor γ; GAE parameter λ; PPO clip ε; learning rates η, β;
Initialize $\theta_{\text{train}}, \theta_P, \theta_V$; let $\theta_{\text{old}} \leftarrow \theta$
Initialize replay buffer $B \leftarrow \emptyset$
for iteration = 1 to max_iters **do**
  // Collect Rollouts
  for t = 1 to T **do**
    Fetch the current state s_t from the environment and construct an input prompt `lang`(s_t);
    Pass prompt `lang`(s_t) through LLM;
    POLICY_NET outputs a_t from action space {`"buy,"` `"sell,"` `"hold"`} based on (8);
    Execute action a_t in the environment and observe reward $r(s_t, a_t)$ and transition to new state s_{t+1};
    Store experience tuple (s_t, a_t, r_t, s_{t+1}) in replay buffer B;
  end for
  // Compute Advantage and Targets
  for each transition in B **do**
    Compute $V_{\theta_{value}}(s_t)$ and advantage A_t (e.g., via GAE)
  end for
  // Perform PPO Updates
  for update_epoch = 1 to K **do**
    Sample mini-batch $\mathcal{M}$ from B
    Compute probability ratio $r_t(\theta_{policy}) = \frac{\pi_{\theta_{policy}}(a_t|s_t)}{\pi_{\theta_{policy,\text{old}}}(a_t|s_t)}$;
    Compute PPO loss $\mathcal{L}_P(\theta_{policy}) = \mathbb{E}_t\Big[\min\big(r_t(\theta_{policy})\,A_t,\ \text{clip}\big(r_t(\theta_{policy}),\,1-\varepsilon,\,1+\varepsilon\big)\,A_t\big)\Big]$;
    Compute Value loss $\mathcal{L}_V(\theta_{value}) = \mathbb{E}_t\Big[(V_{\theta_{value}}(s_t) - R_t)^2\Big]$;
    Compute total loss $\mathcal{L}_{\text{total}}(\theta) = -\mathcal{L}_P(\theta_{policy}) + c_1\,\mathcal{L}_V(\theta_{value}) - c_2\,\mathcal{H}\big(\pi_{\theta_{policy}}\big)$;
    Perform gradient descent on each parameter group:

$$\theta_P \leftarrow \theta_P - \eta\,\nabla_{\theta_P}\mathcal{L}_P,$$
$$\theta_V \leftarrow \theta_V - \eta\,\nabla_{\theta_V}\mathcal{L}_V,$$
$$\theta_{\texttt{train}} \leftarrow \theta_{\texttt{train}} - \beta\,\nabla_{\theta_{\text{train}}}\mathcal{L}_{\text{total}};$$

  end for
  // Update old policy parameters
  Update $\theta = \big(\theta_{\texttt{train}}, \theta_P, \theta_V\big)$ by $\theta_{\text{old}} \leftarrow \theta$;
end for
Return: Fine-tuned POLICY_NET(θ_P).

B Additional Experimental Details

Hyperparameters for Finetuening FLAG-TRADER with PPO in Algorithm 2

Table 3: FLAG-TRADER with PPO Finetuning Hyperparameters and Settings.

Parameter	**Default Value**	**Description**
`total_timesteps`	13860	Total number of timesteps
`learning_rate`	5×10^{-4}	Learning rate of optimizer
`num_envs`	1	Number of parallel environments
`num_steps`	40	Steps per policy rollout
`anneal_lr`	True	Enable learning rate annealing
`gamma`	0.95	Discount factor γ
`gae_lambda`	0.98	Lambda for Generalized Advantage Estimation
`update_epochs`	1	Number of update epochs per cycle
`norm_adv`	True	Advantages whitening
`clip_coef`	0.2	Surrogate clipping coefficient
`clip_vloss`	True	Clipped loss for value function
`ent_coef`	0.05	Coefficient of entropy term
`vf_coef`	0.5	Coefficient of value function
`kl_coef`	0.05	KL divergence with reference model
`max_grad_norm`	0.5	Maximum gradient clipping norm
`target_kl`	None	Target KL divergence threshold
`dropout`	0.0	Dropout rate
`llm`	"SmolLM2-135M-Instruct"	Model to fine-tune
`train_dtype`	"float16"	Training data type
`gradient_accumulation_steps`	8	Number of gradient accumulation steps
`minibatch_size`	32	Mini-batch size for fine-tuning
`max_episode_steps`	65	Maximum number of steps per episode