International Journal on Science and Technology (IJSAT)

E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Experiential Reinforcement Learning for Stock Trading: An LLM-Based Agent Comparison of Mistral and Qwen Without Gradient Updates

Sumedha Arya

Aryasumedha06@gmail.com

Abstract

Large Language Models (LLMs) have proved themselves stronger in decision-making tasks. However, they often struggle in financial markets where information related to profits and losses are delayed and uncertain. In this study, we adapt the Experiential Reinforcement Learning (ERL) framework for single-asset stock trading using Dow Jones Industrial Average (DJIA) data and financial news from 2015 to 2020. Unlike traditional reinforcement learning (RL), our method does not update model weights; instead, learning happens through structured self-reflection, FAISS-based memory storage, and reusing successful trades as few-shot examples. We implement the ERL cycle—first decision, simulated outcome, reflection, improved second decision, and selective memory storage—using Mistral-7B-Instruct-v0.3 and Qwen2.5-7B-Instruct in a custom trading simulator with $10,000 initial capital. Results show that Mistral produced weak reward signals and ended with a −2.13% return, while Qwen stored more useful reflections and achieved around +2–3% return in partial runs, showing more stable improvement. Overall, the study highlights the importance of model capability and clean reward signals, and demonstrates that ERL can be an effective no-gradient alternative to traditional RL for delayed-reward financial trading tasks.

Keywords: Experiential Reinforcement Learning, Large Language Models, Stock Trading, Reflection, Prompt-based Learning, Zero-Gradient Learning

1. Introduction

Large Language Models (LLMs) are now widely used as agents that can take actions based on the environmental feedback, and improves themselves with time. In many real-world tasks, these agents work in complex scenario where rewards can be delayed and information received is incomplete [3,4,5]. This makes the system learning difficult. Reinforcement Learning (RL) is one of the commonly used techniques to train such agents. In RL, the model performs an action and receives a reward at the end. However, when rewards are sparse or delayed, the model does not clearly understand what went wrong or how to fix that problem. As a result, learning becomes unstable and inefficient [6,7]. This problem becomes even more serious while performing multi-step reasoning tasks, where small mistakes early in the process can cause larger errors later.

Humans solve similar problems differently. According to experiential learning theory [1], people learn through a cycle. This cycle comprises of 4 steps; as follows:

1. Experience: - Try something
2. Reflection: - Think about what happened
3. Understand: - What needs to change
4. Try again with improvements

Current RL methods for LLMs mostly rely only on final reward signals. Even in RL with Verifiable Rewards (RLVR), models improve through trial and error, but they must figure out corrections on their own from reward numbers. There is no explicit step where the model reflects and corrects itself during training [3,7].

To solve this issue, authors introduced an Experiential Reinforcement Learning (ERL) paradigm. In ERL, the model follows three main steps inside each learning step:

1. First Attempt: The model generates an initial answer.
2. Self-Reflection: After receiving feedback, the model explains what went wrong and how to improve.
3. Second Attempt: The model tries again using its reflection.

If the second attempt performs well, the improvement is saved (internalized) so that the model can produce better answers in the future—even without reflection during testing.

According to authors, their experimental demonstrations show that ERL improves learning efficiency and final performance as compared to standard RLVR across different tasks. This includes control and reasoning problems also.

2. Literature Review

This section reviews the literature based on use of RL in LLMs. It is divided into three parts covering RL in language models, reflection in learning and trial-and-error to experiential learning.

Reinforcement Learning for Language Models Recent studies apply RL to large language models that act as agents in complex environments [3,4,5]. These methods usually provide a reward after the full answer or completion of an action sequence, rather than providing them in incremental steps. Due to this reason, sparse and delayed rewards can be obtained which create multiple issues. Models struggle to understand which specific part of their behavior caused success or failure [6,7]. This makes learning slow and unstable, especially for reasoning tasks that require multiple steps.

Reflection in Learning Reflection is an important concept in human learning. Experiential learning theory explains that people improve by reflecting on past experiences and adjusting their future actions [1]. LLMs can also generate reflective reasoning at inference time. However, most RL training pipelines do not explicitly train models to reflect and revise during learning. Instead, they depend only on reward signals.

This increases complexity in training time rather than motivating models to learn from their past experiences.

From Trial-and-Error to Experiential Learning There is a progression in training methods. From traditional RL to advanced techniques, concepts have been modified to improve the performance of models. These are as follows:

- Supervised Fine-Tuning (SFT): Models copy correct examples but cannot adapt after deployment. It is based on reward signals only.
- RLVR: Models improve using reward signals due to feedback but rely on trial-and-error learning.
- ERL: Models explicitly reflect, revise, and internalize improvements during training.

Unlike standard RLVR, ERL adds a structured reflection step. This helps the model convert feedback into clear corrective reasoning instead of blindly exploring. Means, the model learns from reflection by converting it into behavioural analysis The improved behavior is then saved into the model through internalization, making learning more efficient and stable. Therefore, ERL is an advanced technique which is built on previous RL approaches [3,4,6,7] along with experiential learning theory [1], combining reward optimization with structured self-reflection.

3. Research Methodology

In this study, we apply ERL proposed by Shi et al. in finance domain. We adapt this framework to stock trading, where profit and loss signals are also delayed and uncertain. The LLMs such as Mistral and Qwen were used as decision-makers for policy π_θ. Based on the core idea of ERL, we performed:

The model makes a decision → simulates the outcome → reflects on it → improves the decision → stores useful lessons → gradually improves over time.

Importantly, no model weight updates are performed. Learning happens through structured reflection, memory retrieval, and few-shot internalization. Following steps were taken in this research methodology:

3.1.Dataset and Preprocessing

The dataset is sourced from Kaggle, which contains approximately 49,637 financial news headlines from 2003 to 2020. Daily closing prices of the Dow Jones Industrial Average are collected using the yfinance library from 2008 onward. The steps for data preparation are as follows:

1. Merge news and price data using the Date column (inner join).
2. Remove missing values.
3. Sort chronologically.
4. Remove days where closing price does not change (to ensure non-zero rewards).
5. Filter data from 2015 onward (modern regime and manageable computation).

After preprocessing, we obtain approximately 1,200–1,500 trading days. Each trading day is treated as an independent task x.

3.2.Trading Environment

We design a custom TradingEnv to simulate single-asset trading. For this, the initial setup we took is with a capital of 10,000 USD and an asset of Dow Jones Industrial Average.

3.3.State Representation

As an input to the LLM, each trading day provides first 300 characters of financial news, current DJIA price and current portfolio value.

3.4.Action Space

The model must output strict JSON as

$$\{"decision": "buy" \mid "sell" \mid "hold", "fraction": 0.0\text{–}1.0\}$$

were,

- Decision is the trading action
- Fraction is the proportion of capital or shares to use

The reward function is given as:

$$\text{Reward} = \frac{\text{Next Portfolio Value} - \text{Current Portfolio Value}}{\text{Current Portfolio Value}}$$

This represents next-day percentage return.

The key innovation in the ERL is simulate () function. An environment is comprised of it. This allows the model to try an action, compute next-day outcome without advancing the real environment. Thus, the daily process becomes:

1. First attempt
2. Reflection
3. Second improved attempt
4. Execute only the improved action

This strictly follows the ERL structure.

3.5.ERL Agent Design

The ERL framework is implemented in the ERLAgent class with models used as Mistral-7B-Instruct-v0.3 and Qwen2.5-7B-Instruct. The algorithm for ERL process is:

Step 1: First Attempt (y^1)

The model receives:

- Current state

- Previously internalized successful trades (few-shot examples)

It generates a trading action.

Step 2: Simulated Feedback

We compute:

- r^1: reward of first attempt
- f^1: textual feedback

Step 3: Self-Reflection (Δ)

The model is prompted with:

- Task description
- First action
- Feedback
- Reward
- Top-2 similar past reflections

Reflection memory uses:

- all-MiniLM-L6-v2 for embeddings
- FAISS for similarity retrieval

The model generates a short reflection describing:

- What was correct or incorrect
- How to improve the decision

Reflection generation uses deterministic decoding (temperature = 0.0).

Step 4: Second Attempt (y²)

The model re-decides using:

- Same market state
- Reflection Δ

This produces improved action y^2.

Step 5: Memory Storage

If:

$$r^2 > \tau \text{ where} \tau = 0.0005$$

Then:

- Store reflection in FAISS memory
- Store (state, action) as internalized example

Step 6: Internalization

Successful examples are added as few-shot demonstrations.

Over time:

- First attempts improve
- Reflection becomes less necessary
- Behavior improvement persists

No gradient updates are performed; learning occurs through prompting and memory.

3.6.Baselines

To isolate the effect of ERL, we compare with:

1. Standard Prompting (No ERL)
   Single decision per day, no reflection, no memory.
2. Reflection Only
   Reflection generated but not stored or reused.
3. Memory Only
   Reflections stored but not used for same-day improvement.

These settings mirror the ablation studies.

3.7.Evaluation Metrics

Primary Metric

- Cumulative portfolio return (%)

Reflection Effectiveness

- Frequency of $r^2 > r^1$

Learning Dynamics

- Number of stored reflections
- Number of internalized examples

Secondary Metrics

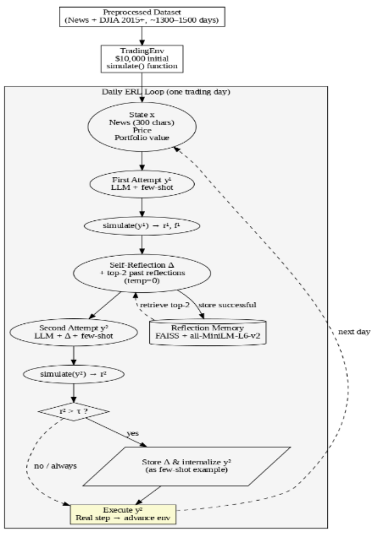
- Sharpe ratio
- Maximum drawdown
- Win rate (positive-return days)

Multiple runs are conducted with different random seeds. Results are averaged and reported with standard deviation.

3.8.Implementation Details

The implementation was done on Kaggle with an NVIDIA Tesla T4 GPU (16GB VRAM) to ensure faster computation and efficient model execution. Several libraries were used, including transformers for working with LLMs, sentence-transformers for generating text embeddings, faiss-cpu for fast similarity search, and yfinance for downloading stock market data. Key hyperparameters selected for this task are: τ (tau); set to 0.0005 as a small decision threshold, top_k; set to 2 to retrieve the two most relevant examples, and 4–5 few-shot examples were provided to guide the model's reasoning. The temperature was set to 0.7 during the decision phase to allow variations in trading actions, and 0.0 during the reflection phase to ensure deterministic and consistent evaluation. The maximum token length was limited to 80 tokens for action generation and 120 tokens for reflection. The ERL architecture for financial stock trading is shown in the figure 1.

Figure 1: ERL Architecture for Financial Stock Trading



4. Results Analysis

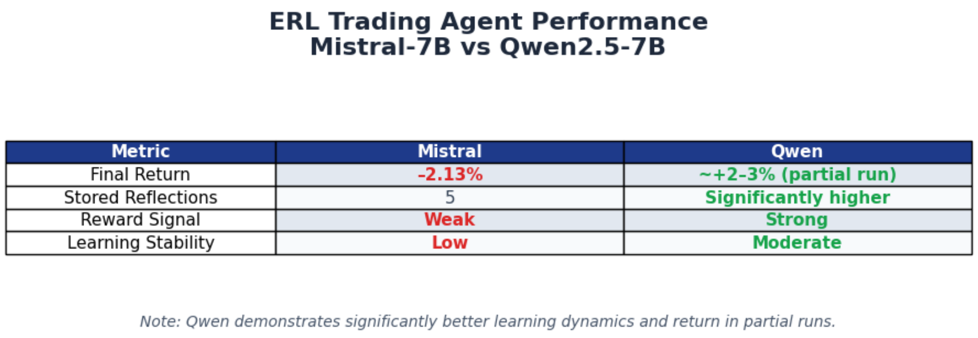
This section represents the analysis of results for ERL based experimentation on financial stock dataset.

In the first experiment, we used an LLM, Mistral-7B, as a trading agent who ended with a small loss of about 2.13%. Most trading days had almost zero reward, which means the agent often chose to hold or the price changes were too small to make a difference. Because of this weak reward signal, the reflection system did not learn much, and only five useful experiences were stored in memory over 80 days. As a result, the agent struggled to adapt to market changes and could not recover well from larger losses.

In the second experiment, we used Qwen2.5-7B LLM as a trading agent. The results got improved after cleaning the data. Removing flat-price days created stronger and clearer reward signals, which helped the agent learn more effectively. The reflection mechanism worked better, more experiences were stored, and the portfolio showed gradual growth of around 2–3% during the early phase. Although the agent still faced volatility and lacked strong risk management, the overall learning process was more stable and effective as compared to the previous experiment. This shows that the ERL framework works, but performance depends heavily on data quality and model capability.

The quantitative summary for the experimentation is given in the table 1.

Table 1: Quantitative Summary of ERL Experimentation



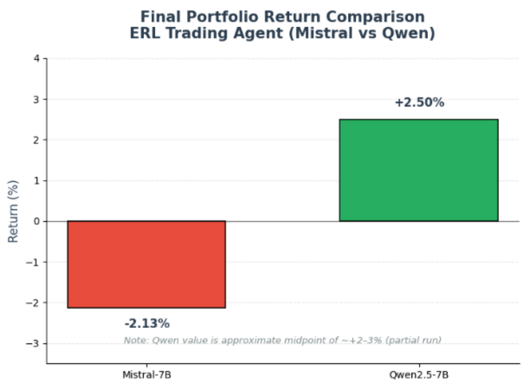
ERL Trading Agent Performance
Mistral-7B vs Qwen2.5-7B

Metric	Mistral	Qwen
Final Return	-2.13%	~+2–3% (partial run)
Stored Reflections	5	Significantly higher
Reward Signal	Weak	Strong
Learning Stability	Low	Moderate

Note: Qwen demonstrates significantly better learning dynamics and return in partial runs.

The final return comparison between Mistral and Qwen models is shown in the figure 2.

Figure 2: Final Return Comparison: Mistral vs Qwen



5. Conclusion

In conclusion, based on the comparative analysis of results, it is clear that Qwen outperformed Mistral in this ERL-based trading setup. Mistral ended with a negative return (–2.13%), stored very few reflections, and showed weak learning stability due to poor reward signals. In contrast, Qwen achieved a positive return (around +2–3% in the partial run), demonstrated stronger reward signals, and showed more stable learning behavior.

Overall, the results suggest that model capability and data quality significantly impact performance for ERL framework. Qwen was better at using reflection memory and adapting to market changes, while Mistral struggled to learn effectively. This confirms that the ERL framework works, but its success depends heavily on the strength of the underlying language model and the clarity of the reward signal.

References

1. Kolb D.A., “Experiential Learning: Experience as the Source of Learning and Development”, FT Press, 2014, 2nd ed.
2. OpenAI, “GPT-5 System Card”, arXiv:2601.03267v1 [cs.CL], December 2025.
3. Yang A., et al., “Qwen3 Technical Report”, arXiv:2505.09388, 2025.
4. Song L., Dai Y., Prabhu V., et al., “Coact-1: Computer-Using Agents with Coding as Actions”, arXiv:2508.03923, 2025.
5. Bai Y., et al., “Kimi K2: Open Agentic Intelligence”, arXiv:2507.20534, 2026.
6. Zhang et al., “Agent Learning via Early Experience”, arXiv:2510.08558, 2025.
7. Shi T., et al., “Experiential Reinforcement Learning”, arXiv preprint, 2026.