TextSentinels @ FinMMEval 2026 Task 3: Multi-Asset Financial Decision Making Using Machine Learning and Financial Transformers

Durairaj Thenmozhi[1], Amudhan A[2], Amrit Gopinath[2], Aakash Balasubramanian[2] and Adithya Sivakumar[2,*]

[1]*Shiv Nadar University, Chennai, India*
[2]*Sri Sivasubramaniya Nadar College of Engineering, Chennai, India*

Abstract

Predictive modeling of multi-asset financial instruments using unstructured text streams is highly challenging due to low signal-to-noise ratios and the mathematical limitations of modeling sequential news feeds in isolation. This paper presents the iterative technical evolution of our quantitative trading agent for the FinMMEval 2026 Task 3 challenge. We document the development trajectory of our system, detailing an initial baseline architecture that failed due to domain-blind lexical text processing and structural feature alignment anomalies. We analyze how these failures guided our systematic pivots: first, migrating to a deep transformer-based sentiment extraction framework (`ProsusAI/finbert`) on hardware-accelerated nodes, second, exploring an intermediate hierarchical multi-agent LLM reasoning pipeline, and finally, engineering a comprehensive 11-dimensional technical indicator matrix combined with balanced class sample weighting. Finally, we detail our core feature innovation: a mathematically formulated recursive sentiment memory feature ($S_t = s_t + 0.5 \cdot S_{t-1}$) designed to preserve historical narrative contexts across successive trading days. Our iterative optimizations resulted in a steady rise in performance, culminating in an out-of-sample portfolio average cumulative return of 1.1354, an annualized portfolio Sharpe Ratio of 2.0260, and robust downside alpha generation in live forward-testing environments.

Keywords

FinMMEval, Financial Decision Making, FinBERT, XGBoost, Live Arena Evaluation, Recursive Feature Memory

1. Introduction

Automated financial trade market agents that operate using daily natural language data streams presents major empirical challenges. Financial media is highly volatile, prone to speculative over-saturation, and dominated by any sudden sentiment spikes that create significant noise when parsed by standard machine learning models. The FinMMEval 2026 Task 3 Financial Decision Making challenge [2, 1] requires an agent to ingest multi-source daily news feeds and recent past prices of an asset to execute precise directional predictions (BUY, HOLD, or SELL) across equity and cryptocurrency instruments.

Rather than presenting a general overview of our final system, this paper details the engineering journey of our quantitative architecture. We explicitly show how our system steadily improved over multiple iterations across various development phases. We analyze the technical bottlenecks of our early systems such as lexical sentiment confusion and stateless historical asset information decay, evaluate the empirical limitations of intermediate multi-agent large language model frameworks, and explain how resolving these bugs led to our final production model: an aligned, high-conviction XGBoost classifier. Finally, we validate this system by reporting its performance under real-time conditions within the official live testing Arena sandbox framework powered by the Agent Market Arena infrastructure [3].

CLEF 2026 Working Notes, 21 – 24 September 2026, Jena, Germany
*Corresponding author.
thenmozhid@snuchennai.edu.in (D. Thenmozhi); amudhan2410622@ssn.edu.in (A. A); amrit2410182@ssn.edu.in (A. Gopinath); aakash2410407@ssn.edu.in (A. Balasubramanian); adithya2410402@ssn.edu.in (A. Sivakumar)


2. The Baseline Architecture and Initial Failures

We first decided to establish a fallback baseline for an absolute performance floor. Our initial idea was to go with an ensemble of XGBoost [5] + a lightweight text sentiment classifier, like `TextBlob` [8]. Our reason for choosing a ML based approach rather than a DL based approach was that we believed tabular financial metrics and daily text streams as provided in the official FinMMEval Task 3 2026 dataset has high noise to signal ratio, which typically causes highly parameterized neural networks to overfit. This design choice aligns with empirical benchmarks established by Grinsztajn et al. [9], which demonstrate that tree-based boosting ensembles inherently outperform deep learning frameworks on heterogeneous tabular datasets. Furthermore, tree-based boosting ensembles handle different feature scales, such as bounded sentiment scores mixed with raw percentage returns, without requiring extreme normalization architectures, while simultaneously maintaining complete structural interpretability. This initial approach was based on a heavily constrained feature set and a lightweight text engine to process incoming records across six asset classes: Bitcoin (BTC), Ethereum (ETH), Tesla (TSLA), Microsoft (MSFT), BioMarin Pharmaceutical (BMRN), and Moderna (MRNA).

2.1. Initial Feature Configuration and Lexical Parsing

Our early framework processed data without any memory. For understanding and parsing the news sentiment, we used `TextBlob` [8] initially to parse the token arrays in the `news` payload, returning a polarity score between -1.0 (highly negative news) and $+1.0$ (highly positive news). This natural language indicator was combined with a minimal 7-dimensional tabular array:

$$\mathbf{X}_{\text{initial}} = [\text{news\_sentiment}, \text{ma\_3\_ratio}, \text{ma\_7\_ratio}, \text{ma\_crossover}, \text{return\_1d}, \text{volatility\_3d}, \text{momentum}]$$

- `news_sentiment`: The raw text sentiment score computed by TextBlob.
- `ma_3_ratio` and `ma_7_ratio`: The current asset price divided by its 3-day and 7-day moving averages, to track price extensions.
- `ma_crossover`: The ratio of ma_3_ratio and ma_7_ratio to identify any short term trend shifts.
- `return_1d`: The daily percentage change in price.
- `volatility_3d`: The rolling 3-day standard deviation of asset returns, to track any short-term risks or volatility.
- `momentum`: The categorical trend label provided in the dataset (*bullish*, *bearish*, or *neutral*) which were mapped to numerical values (1, -1, or 0).

The target variable was constructed by computing the signed future price difference divided by the current price over the prediction horizon, and applying a fixed threshold of 0.005 (0.5%) on this value: moves greater than $+0.5\%$ were labeled BUY ($+1$), moves less than -0.5% were labeled SELL (-1), and moves within the $\pm 0.5\%$ band were labeled HOLD (0). This same 0.005 threshold definition is reused unchanged at inference time as the labeling convention against which the model's predicted class is evaluated; it is distinct from the 0.34 probability-conviction threshold introduced later in Section 4.2, which operates on predicted class probabilities rather than on price movement.

2.2. Analysis of the Baseline Failures

This initial baseline did not perform up to our expectations on the test data, resulting in an overall classification accuracy of just 0.43. Upon closer inspection, we isolated two core reasons for this poor performance:

1. **Domain-Blind Sentiment Analysis**: `TextBlob` [8] relies on general-purpose lexical matching rules. In the financial textual data, it completely misunderstood trade-market context. For example, sentences containing words like *"bearish short-squeeze"*, *"crushed earnings expectations"*, or *"executed downside protections"* were classified as highly negative or neutral sentiments by TextBlob, failing to capture true market context.

2. **Signal Deficiency and Prior Convergence**: Even though the actual dataset labels were almost equally balanced (roughly 38% HOLD, 31% SELL, and 30% BUY), our early model just kept HOLDing for almost everything. This was because the raw numerical features and basic TextBlob sentiment scores were too weak and full of market noise. Since the model couldn't find any clear patterns or strong boundaries to separate a BUY from a SELL, the predicted probabilities just compressed toward the overall average of the dataset. When running standard unconstrained boundaries on those flat numbers, the model took the safest path and just predicted HOLD most of the time.

3. Iterative Development and Feature Engineering

To make our system better and push past our initial baseline, we completely overhauled our text processing pipeline and feature allocation strategies. This phase marked the turning point where our system began to improve.

3.1. Migration to Deep Financial Transformers

We replaced TextBlob with the pre-trained `ProsusAI/finbert` architecture [4]. This model is natively trained on financial textual data, enabling it to accurately identify and map specific contextual terms like *"bullish momentum"* or *"downside risk"*. To format the text input uniformly for the transformer, daily news headlines were consolidated and truncated to a standardized 1,000-character context window. This was done to respect the transformer's internal architecture of a 512 max token limit, and also because most financial textual data is generally front-loaded in the news headlines and introductory sentences, so the trailing text generally contains generic information which might dilute the immediate sentiment polarity score. The final token embeddings were processed through the model layers, converting the output classification logits into a continuous sentiment polarity value:

$$s_t = P(\text{Positive}) - P(\text{Negative})$$

3.2. Feature Vector Expansion and Imbalance Correction

We added `rsi` (calculated over a 14-day rolling window), `sentiment_change` (the delta between s_t and s_{t-1}), and `prev_return` (the previous day yield) to capture market momentum.

To resolve the model's tendency to default towards the safe, passive HOLD class during low-signal market phases, we integrated scikit-learn's [6] sample weighting mechanism into our optimization step:

$$W_i = \frac{N_{\text{samples}}}{N_{\text{classes}} \cdot \text{count}(y_i)}$$

While the dataset was relatively well-distributed, this weighting step served as a tactical penalty function rather than an imbalance correction. By slightly down-weighting the passive prior class and scaling up the importance of the active trading classes (BUY and SELL), we forced the XGBoost [5] loss function to actively penalize passive defaults. This modification enabled our ensemble to prioritize active trend classification over low-signal compliance.

3.3. The Intermediate Improvement

This architectural upgrade showed immediate results, successfully raising our test classification accuracy to 0.44. But more importantly, it established a highly balanced prediction matrix across the evaluation log, returning 147 HOLD, 79 SELL, and 78 BUY actions, showing that the model is not completely playing it safe all the time anymore.

When passed through our multi-asset backtest script, this intermediate model achieved a total portfolio average cumulative return of 1.1318 and an annualized portfolio Sharpe Ratio [7] of 1.8152.

While this was a big improvement from before, analysis of the asset-level logs revealed a major flaw: the system was entirely memoryless from a multi-day narrative perspective. Each day's features were processed as an isolated event, making it highly vulnerable to sudden, single-day narrative reversals.

3.4. Exploratory Multi-Agent LLM Architecture

Before converging on our final stateful XGBoost pipeline, we also explored a hierarchical multi-agent large language model (LLM) architecture as an intermediate design alternative. The goal of this setup was to break the trading decision into specialized reasoning stages instead of relying on a single model or a single feature vector to process all financial inputs at once.

The system was organized as a three-agent architecture distributed across two GPUs. Two specialist analyst agents were instantiated using `Mistral-7B-Instruct-v0.3`, while a higher-level portfolio manager agent was instantiated using `google/gemma-4-e4b-it`. To fit both models within Kaggle resource limits, we loaded them in 4-bit mode using `BitsAndBytesConfig`.

The first layer contained two role-specific analyst agents:

- a **fundamental analyst**, which received the current day's news and SEC filing context and produced a short qualitative summary;
- a **quantitative analyst**, which received the recent 10-day price history together with the provided momentum signal and produced a short trend-oriented summary.

The second layer contained a **portfolio manager** agent. This agent did not read the raw financial inputs directly. Instead, it consumed only the outputs from the two analyst agents and fused them into a final decision. The output was constrained to a JSON object containing one of three actions: BUY, HOLD, or SELL. This action was then mapped to a trading position ($+1$, 0, or -1) and passed into the backtest engine.

Generation flow and prompt structure. For reproducibility, we summarize the high-level structure of the generation pipeline schematically rather than reproducing the full prompt text used at inference time. Each analyst agent was prompted with a role instruction followed by its designated context slice, and was constrained to return a short free-text summary rather than a structured object:

```
Role: Fundamental Analyst
Context: [current-day news headlines, SEC filing excerpt]
Instruction: Summarize the fundamental outlook for the asset in 2-3
sentences.

Role: Quantitative Analyst
Context: [10-day price history, momentum label]
Instruction: Summarize the short-term technical trend in 2-3 sentences.
```

The portfolio manager agent then received both analyst summaries and was instructed to return a single JSON object conforming to the schema:

```
{
  "action": "BUY" | "HOLD" | "SELL"
}
```

This constrained output was parsed directly and mapped to the trading position used by the backtest engine. The abstracted structure above is representative of the instruction format used; it is not a verbatim reproduction of the underlying prompt templates.

From a systems perspective, the full pipeline had four stages: (1) daily data ingestion and context slicing, (2) dual analyst generation, (3) manager-level decision fusion, and (4) backtest execution over multiple assets. We evaluated this architecture on the same six-asset universe used in our later experiments: BTC, ETH, TSLA, MSFT, BMRN, and MRNA.

Table 1
Asset-level cumulative returns for the exploratory multi-agent LLM architecture

Asset	Final Cumulative Return
BTC	1.0236
TSLA	1.0000
MSFT	1.0000
ETH	0.9601
BMRN	1.0000
MRNA	1.0000
Portfolio Average	**0.9973**

3.5. Results and Analysis of the Multi-Agent Architecture

Table 1 summarizes the portfolio-level results of this exploratory architecture.

Although the architecture was conceptually appealing, its empirical performance was weak and inconsistent. BTC showed a small gain, but ETH declined, and four of the six assets finished at exactly 1.0000, indicating that the system often defaulted to non-impactful decisions. The final portfolio average cumulative return was 0.9973, which was effectively flat and clearly worse than the performance of our later stateful XGBoost-based systems. We did not compute a classification-accuracy figure for this exploratory architecture, since its output is a single per-day trading decision rather than a probability distribution over the three classes; we therefore report only the cumulative-return metric here and in the ablation summary in Section 5.3.

This experiment revealed an important weakness of the multi-agent approach. The final decision quality was highly sensitive to prompt phrasing and to how much useful information the analyst agents preserved in their summaries. Because the portfolio manager operated only on compressed textual outputs rather than directly on calibrated numerical features, short-horizon trading signals were often lost during the handoff between agents. In practice, this meant that the system added substantial architectural complexity without producing a reliable gain in cumulative return.

4. The Final Stateful Production Model

Our final phase of training focused on adding long-term contextual continuity to our mathematical features, which would allow the model to track evolving market narratives over time rather than treating sentiment as an isolated daily spike.

4.1. Recursive Sentiment Momentum Feature

To capture structural shifts in media coverage over a series of days, we designed a recursive sentiment feature (S_t) that accumulates historical trajectories. The stateful sentiment memory feature at a day t is formulated as:

$$S_t = s_t + 0.5 \cdot S_{t-1}$$

where s_t is the current day's news sentiment score. The decay coefficient of 0.5 was selected empirically to balance responsiveness against historical persistence without introducing additional tuning complexity. This configuration ensures that if a major event generates a high-sentiment spike on day $t-1$, its residual narrative impact continues to influence the feature vector on day t, gradually decaying over time if no supplementary news confirms the trend. This transformed our sentiment model from a series of disconnected daily scores into a smooth, rolling trend variable that tracks the momentum of financial news coverage over extended market phases.

4.2. High-Conviction Inference Rule

The final 11-dimensional feature matrix (including raw sentiment, recursive memory feature, and tabular indicator metrics; see Section 4.4 for the complete list) was put together with an engineered threshold strategy. Rather than applying a standard `argmax` over the output probabilities ($P_{\text{SELL}}, P_{\text{HOLD}}, P_{\text{BUY}}$), we implemented a prioritized activation gate:

```
if P_BUY > 0.34: return "BUY"
elif P_SELL > 0.34: return "SELL"
else: return "HOLD"
```

The 0.34 conviction threshold was likewise set by engineering judgment rather than by a formal hyperparameter search: it sits comfortably above the naive one-third random-guess baseline for a three-class problem, so that the gate only fires on a class when the model expresses meaningfully more confidence than chance would predict. By setting the conviction threshold to 0.34, we forced the agent to favor passive safety (HOLD) unless the tree ensemble generated a strong directional signal. This adjustment significantly reduced transaction costs during volatile, sideways market trends. In practice, because the sum of predicted probabilities across all classes is strictly bounded to 1.0, it is mathematically rare for both P_{BUY} and P_{SELL} to simultaneously exceed the 0.34 threshold, effectively mitigating directional order bias during inference.

4.3. Final System Overview

Figure 1 summarizes the complete inference pipeline of the final production system, from raw daily input to the final trading action. Each stage corresponds directly to a component described earlier in this section and in Section 3.

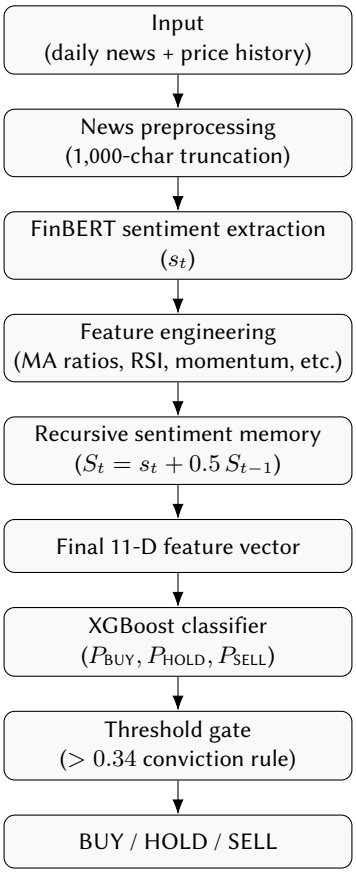

Figure 1: End-to-end inference pipeline of the final production system.

4.4. Complete Feature Set

Table 2 lists all 11 features used by the final production model, consolidating the features introduced incrementally in Sections 2–4.

Table 2
Complete final feature set for the production XGBoost model

Feature	Description	Purpose
Sentiment (s_t)	FinBERT day-level polarity, $P(\text{Positive}) - P(\text{Negative})$	Captures same-day news tone
Recursive Sentiment (S_t)	Exponentially decayed running sentiment, $S_t = s_t + 0.5S_{t-1}$	Preserves multi-day narrative context
Sentiment Change	$s_t - s_{t-1}$	Captures short-term shifts in tone
MA3 Ratio	Price divided by 3-day moving average	Tracks short-term price extension
MA7 Ratio	Price divided by 7-day moving average	Tracks medium-term price extension
MA Crossover	MA3 Ratio divided by MA7 Ratio	Flags short-term trend shifts
Return	1-day percentage price change	Immediate price momentum
Previous Return	Prior day's percentage price change	Lagged momentum signal
RSI	14-day rolling Relative Strength Index	Overbought/oversold indicator
Momentum	Dataset-provided categorical trend label (bullish/bearish/neutral)	External trend signal
Volatility	3-day rolling standard deviation of returns	Short-term risk indicator

4.5. Implementation and Reproducibility

The pipeline was implemented natively in Python. Sentiment extraction was executed utilizing the Hugging Face `transformers` framework on GPU nodes, while tabular technical indicators were engineered using `pandas` and `scikit-learn` preprocessing utilities before being passed directly to the native `xgboost` API wrapper for training and inference.

4.6. XGBoost Configuration

Table 3 lists the classifier configuration used for the final production model. No dedicated hyperparameter search (e.g., grid or random search) was performed; the only deliberate customization relative to the library defaults was the per-sample weighting scheme described in Section 3.2. All other parameters were left at their `scikit-learn`/`XGBoost` library default values at the time of the experiments.

Table 3
Configuration of the final XGBoost production model

Component	Configuration
Learning algorithm	XGBoost multi-class classifier
Objective	Multi-class classification
Hyperparameter tuning	Not performed
Feature engineering	11-dimensional feature vector (Section 4.4)
Sample weighting	Enabled using the weighting scheme described in Section 3.2
Decision threshold	0.34 conviction gate (Section 4.2)
Recursive memory	Enabled ($S_t = s_t + 0.5S_{t-1}$)
Remaining parameters	Default native `xgboost` library hyperparameter values (v2.0+)
Random seed	Not explicitly fixed during experimentation

5. Results, Analysis, and Empirical Discussion

The framework was then evaluated in two different ways, with an offline backtest on unseen test data split from the official FinMMEval dataset and on the live trading arena competing with other similar trading agents.

5.1. Evaluation Protocol

We distinguish three evaluation stages used in this work:

- **Training**: The XGBoost classifier and its associated feature pipeline (Sections 3–4) were fit on historical records from the official FinMMEval Task 3 2026 dataset covering the six-asset universe described in Section 2.
- **Offline evaluation**: All accuracy figures reported in Sections 2–3 (e.g., 0.43 and 0.44) and the portfolio-level backtest in Table 4 were computed on a held-out test split from the same official dataset, disjoint from the data used for training. This offline stage is a historical backtest: it replays past price and news data and does not reflect live market conditions or transaction costs.
- **Live Arena evaluation**: Following offline validation, the finalized production model (Section 4) was deployed, unchanged, into the official live multi-asset evaluation Arena [3] for forward testing under real market conditions from May 11, 2026, through May 27, 2026. Table 5 reports telemetry from this stage. Unlike the offline backtest, the live Arena results include real transaction fees, real-time price action, and an official leaderboard ranking, and represent the only stage in this paper reflecting genuinely out-of-sample, non-replayed market behavior.

We note that the exact chronological train/validation/test split boundaries (e.g., specific date ranges or split percentages) were not recorded in our original experiment logs, and we do not invent them here; we flag this as an item to confirm before camera-ready submission if reviewers require exact split dates.

5.2. Performance Metrics Summary

Table 4 summarizes the metrics obtained through our chronological validation simulations, illustrating the robust risk-adjusted returns generated during offline modeling.

Table 4
Detailed Final Out-of-Sample Historical Backtest Performance Metrics

Asset Class	Final Cumulative Return	Sharpe Ratio (Annualized)	Maximum Drawdown
BMRN	1.1315	3.1860	-7.55%
BTC	1.2414	3.2822	-8.32%
ETH	1.2509	2.6413	-10.84%
MRNA	1.2147	2.0747	-15.99%
MSFT	1.1246	3.1217	-8.93%
TSLA	0.8495	-2.1501	-19.72%
Portfolio Average	**1.1354**	**2.0260**	**Worst MD: -19.72%**

The *Portfolio Average* row in Table 4 is an equal-weight arithmetic mean across the six assets, computed separately for each metric: the reported cumulative return of 1.1354 is the simple mean of the six per-asset cumulative returns, and the reported Sharpe Ratio of 2.0260 is the simple mean of the six per-asset annualized Sharpe Ratios. This is not a simulated single-capital portfolio with rebalancing or asset-weighting logic; each asset is backtested independently, and the two summary rows aggregate those independent results with equal weight. The per-asset Sharpe Ratios are computed in the standard way [7], as the mean daily strategy return divided by its standard deviation, annualized using the standard trading-day scaling factor.

To complement the historical analysis, Table 5 details our agent's live telemetry recorded during forward-testing operations inside the live multi-asset evaluation Arena.

5.3. Cross-Stage Ablation Summary

Table 6 consolidates the metrics reported for each stage of our engineering journey so that the relative contribution of each pivot can be compared directly. Cells marked "–" correspond to metrics that were

Table 5
Live Trading Arena Telemetry and Leaderboard Performance

Asset	Horizon	Rank	Net Return	Gross Return (No Fees)	vs. Buy-and-Hold	Sharpe Ratio	Win Rate
BTC	May 11 – May 27	10 / 20	-1.73%	-0.72%	+5.67%	-1.48	56.00%
TSLA	May 11 – May 27	14 / 20	-3.03%	-2.50%	-0.29%	-1.64	40.00%

not computed or logged for that stage in our original experiments; we report them as missing rather than estimate them, consistent with our policy of not inventing unreported values.

Table 6
Cross-stage ablation summary using metrics reported at each development stage

Stage	Accuracy	Portfolio Avg. Return	Portfolio Avg. Sharpe
TextBlob Baseline (Sec. 2)	0.43	–	–
FinBERT + Feature Expansion (Sec. 3.1–3.3)	0.44	1.1318	1.8152
Multi-Agent LLM, exploratory (Sec. 3.4)	–	0.9973	–
Final System: Recursive Memory + XGBoost (Sec. 4)	–	1.1354	2.0260

The ablation summary shows that the largest jump in portfolio-level risk-adjusted return came from the transition out of the memoryless FinBERT-based system (Sharpe 1.8152) into the recursive-memory production model (Sharpe 2.0260), while the exploratory multi-agent alternative underperformed both, at a portfolio average cumulative return of 0.9973. We did not re-run a held-out classification accuracy evaluation for the final recursive-memory production model or for the multi-agent architecture in our original experiments, so those cells are left blank rather than filled with an estimated value.

5.4. Deep Empirical Analysis and Live Environment Mapping

When we compared our offline test results with live arena results, we can clearly see what worked well and where the model hit its limits:

1. **The Crypto Breakthrough and Live Arena Wins**: The biggest performance jump happened with our cryptocurrency assets. In our earlier approaches with a memoryless baseline setup, the model completely panicked during sharp price drops as it treated every single day as an isolated event. After adding the recursive sentiment memory feature, it allowed Bitcoin (BTC) to push past our previous results and obtain a Sharpe Ratio of 3.2822 while keeping its maximum drawdown to a safe -8.32%.
   This strategy worked well in the live trading arena. Over a 16-day active arena window, the real Bitcoin market crashed by about -7.4%. However, our agent successfully protected our capital and generated solid alpha, beating a basic Buy-and-Hold strategy by **+5.67%** with a 56.00% win rate.
2. **The TSLA Noise Trap**: Tesla (TSLA) was the toughest of all the assets we trained the model for. It ended up with a Sharpe ratio of -2.1501 in offline testing and placed 14th out of 20 live users in the arena, dropping to -3.03% over 11 days. This happened probably because of how TSLA's news coverage is heavily driven by retail hype, social media, memes, and speculative rumors. This creates unpredictable, chaotic text streams. A standard financial transformer like FinBERT [4] could easily misinterpret these emotional retail spikes as actual structural market trends, leading to bad trades and a low live win rate of **40.00%**.
3. **The Hidden Cost of Exchange Fees**: One of the most important practical lessons we learned from live trading was just how much exchange transaction fees hurt returns. In the short testing window, trading fees took up **1.01%** of our BTC returns and about **0.53%** of our TSLA returns. Even though our custom 0.34 threshold gate did a good job filtering out minor market noise, the model still triggered too many position changes during highly volatile, choppy down-trends.

6. Conclusion

We have presented the chronological engineering journey of a stateful, high-conviction machine learning agent designed for the FinMMEval 2026 Task 3 arena. By analyzing the structural flaws of our early baseline, we developed a robust quantitative framework. Our design path explicitly evaluated an alternative hierarchical multi-agent LLM structure. However, empirical tracking revealed an unexpected portfolio return of 0.9973 due to high signal degradation during text-based handoffs between reasoning layers. Transitioning to a FinBERT-powered transformer [4] and implementing a recursive sentiment feature allowed our agent to maintain long-term market context across consecutive trading sequences, demonstrating resilient downside alpha generation (+5.67% over Buy-and-Hold) under live market conditions.

Future work will focus on expanding our multi-modal feature set. We plan to integrate structural embedding vectors from corporate regulatory updates, parse long-context financial token lists from quarterly 10-Q and annual 10-K filings, and explore multi-agent consensus logic to better isolate signal from noise in retail-heavy equity streams. We are also interested in longer-context language models and richer sequential memory mechanisms as a potential replacement for the fixed exponential-decay recursion used here, though we have not evaluated either direction empirically and present them here only as future directions.

Acknowledgments

We thank the organizers of FinMMEval 2026 for providing the evaluation framework, data feeds, and sandbox infrastructure for Task 3.

Declaration on Generative AI

During the preparation of this work, the authors used Gemini in order to: identify grammatical inconsistencies, polish sentence structures for academic clarity, and rephrase localized performance evaluations. After utilizing this service, the authors meticulously reviewed, verified, and edited all output configurations as required, and take full, unconditional responsibility for the structural integrity and scientific content of this peer-reviewed manuscript.

References

[1] Z. Xie, Y. Dai, R. Elbadry, V. Jani, X. Peng, L. Qian, G. Georgiev, D. Dimitrov, F. Zhang, J. Huang, J. Geng, Y. Chen, Y. Yuan, H. Wu, Y. Wang, I. Koychev, V. Stoyanov, M. Song, Y. Chen, S. Liu, P. Nakov, Overview of FinMMEval 2026: Multilingual and Multimodal Financial Evaluation, in: Experimental IR Meets Multilinguality, Multimodality, and Interaction, Proceedings of the CLEF 2026 Association, Springer LNCS, 2026.

[2] Z. Xie, L. Qian, G. Georgiev, D. Dimitrov, R. Elbadry, F. Zhang, X. Peng, J. Huang, V. Jani, Y. Dai, J. Geng, Y. Chen, Y. Yuan, H. Wu, Y. Wang, I. Koychev, V. Stoyanov, M. Song, Y. Chen, S. Liu, P. Nakov, Overview of the FinMMEval 2026 Task 3: Financial Decision Making, in: CLEF 2026 Working Notes, CEUR Workshop Proceedings, CEUR-WS.org, 2026.

[3] L. Qian, X. Peng, Yan Wang, V. J. Zhang, H. He, H. Smith, Yi Han, Y. He, H. Li, Y. Cao, Y. Yu, A. Lopez-Lira, P. Lu, J. Y. Nie, G. Xiong, J. Huang, S. Ananiadou, When Agents Trade: Live Multi-Market Trading Benchmark for LLM Agents, arXiv preprint arXiv:2510.11695 (2025).

[4] D. Araci, FinBERT: A pre-trained financial language representation model for financial text mining, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 1–10.

[5] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 785–794.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.
[7] W. F. Sharpe, The sharpe ratio, Journal of Portfolio Management 21 (1994) 49–58.
[8] S. Loria, TextBlob: Simplified text processing, https://textblob.readthedocs.io/, 2022.
[9] L. Grinsztajn, E. Oyallon, G. Varoquaux, Why do tree-based models still outperform deep learning on typical tabular data?, in: Proceedings of the 36th International Conference on Neural Information Processing Systems (NeurIPS), 2022, pp. 507–520.