HANOI UNIVERSITY OF SCIENCE AND TECHONOLOGY

SCHOOL OF INFORMATION COMMUNICATION TECHNOLOGY

Graduation Research 2 Project Report:

The use of Lightweight Agentic AI in walk-forward trend prediction for VN30 stock symbols

---

Supervised by:

Assoc. Prof. Cao Tuan Dung

Student in charge:

Vo Thanh Vinh - 20226074

Hanoi - Vietnam

Jan 2026

Abstract

This report presents an agent-based financial AI pipeline for VN30 trend prediction using news and price data. The system includes news crawlers (Cafef, Vietstock, VnExpress), a price collector from FireAnt, a corporate action collector from VSD, PostgreSQL storage, and a four-agent processing chain: event extraction (Agent 1), market reaction labeling by horizon (Agent 2), joint feature engineering (Agent 3), and model training/prediction (Agent 4). The pipeline is orchestrated by the main entry scripts `run_all.py` and `run_smoke.py`, with the agent chain defined in `src/pipelines/run_agents.py`.

The experimental data window is January 2023 to January 2026. Within this window, Agent 1 processed 148,182 news articles and extracted 67,728 structured events. Two horizons are evaluated: 15 days and 40 days. For the 40-day horizon, the training/validation set size is about 64,990 samples; Logistic Regression achieves AUC 0.532 and accuracy 0.558, while the MLP 96-48-24-1 achieves AUC 0.565 and accuracy 0.570. In backtesting, Logistic Regression at horizon 40 yields `mean_ret_h` = 0.0873 and `hit_rate` = 0.648, while the MLP yields `mean_ret_h` = 0.0803 and `hit_rate` = 0.668. For horizon 15, the dataset is larger (about 130,953 samples), AUC ranges from 0.495 to 0.538, and the best `hit_rate` is 0.696 (LR) with `mean_ret_h` = 0.0645.

Key contributions include: (1) a reproducible end-to-end pipeline from data collection to backtesting; (2) joint feature engineering that integrates news text signals with price context; (3) a comparison of linear vs. nonlinear models across two horizons; and (4) a critical analysis of limitations with concrete future directions such as alpha labeling, regime conditioning, and cycle features.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivation

The VN30 index is highly sensitive to corporate news, macroeconomic updates, and governance events. Combining news with price history can provide early signals, but it requires a robust pipeline to handle large, noisy, and heterogeneous data. The objective of this project is to build a full end-to-end system that can (1) collect and unify news, price, and corporate action data; (2) convert news into structured events; (3) label market reactions by horizon; (4) construct multi-source features; and (5) train predictive models with backtesting. These steps are implemented in the `stock-agent-lab` codebase.

1.2 Objectives and Scope

The scope focuses on VN30 constituents (listed in `src/collectors/vn30_universe.py`) and two prediction horizons: 15 days and 40 days. The system collects news from three sources via a custom crawler in `src/collectors/news_sitemap_backfill.py`, price data from FireAnt (`src/collectors/fireant_price_collector.py`), and corporate action data from VSD (`src/collectors/corp_actions_collector.py`).

The main orchestration entrypoint is `stock-agent-lab/run_all.py`, which supports full-range execution and smoke testing. A lightweight entrypoint for fast testing is `stock-agent-lab/run_smoke.py`. The report describes the exact behavior of these scripts and their pipeline modules in `src/pipelines/*`.

1.3 Contributions

The specific contributions include:

- A batch pipeline with coverage measurement and gap-filling (see `run_all.py` and `src/utils/coverage.py`).

- A rule-based event extraction agent that links articles to symbols and event types with rich reporting (see `src/agents/agent1_news_event.py`).
- Market reaction labeling by horizon with volatility and drawdown statistics (see `src/agents/agent2_market_react.py`).
- Joint feature engineering across news, price, and corporate actions using text hashing and technical signals (see `src/agents/agent3_features_joint.py`, `src/utils/features.py`).
- A comparison of linear and nonlinear models with a consistent back-test protocol (see `src/agents/agent4_train_predict_joint.py` and `src/agents/agent4_train_predict_nn.py`).

1.4 Report Organization

Chapter 2 reviews related work. Chapter 3 explains the system architecture, including storage and collectors. Chapter 4 details the experimental design, labeling, and feature engineering. Chapter 5 presents results for horizons 15 and 40, including LR vs. MLP comparisons. Chapter 6 discusses interpretation, limitations, and threats to validity. Chapter 7 concludes and lists future work.

1.4.1 Problem Definition in Operational Terms

This project frames news-driven prediction as a sequence labeling and forecasting task. Each news article is first mapped to one or more VN30 symbols using aliases in `src/collectors/vn30_universe.py`. The article is then converted into a structured event record with an event type, sentiment score, and impact hint (Agent 1). The core prediction target is the horizon return (`ret_h`) measured from the first trading day after publication (t0) to t0+H. Agent 2 creates the binary label `label_up` (1 if `ret_h>0`) and provides auxiliary statistics for analysis and risk context.

1.4.2 Why an Agent Chain

The pipeline is separated into agents to enforce clear responsibilities and to make each step auditable. For example, Agent 1 focuses only on text-to-event transformation and does not depend on price data. Agent 2 focuses on aligning events to prices and constructing labels. Agent 3 creates joint features that combine text, price context, and corporate actions. Agent 4 performs model training and evaluation. This separation improves reproducibility and allows targeted debugging (e.g., if market reactions drop due to missing prices, the issue is isolated to Agent 2 or the price collector).

1.4.3 Data Window and Scope Justification

The experimental window is January 2023 to January 2026, which balances data volume with market regime relevance. VN30 is used as a stable universe of liquid, large-cap stocks, reducing thin-liquidity artifacts. The 15-day and 40-day horizons are chosen to represent short-term and medium-term reactions, which can diverge due to delayed market digestion or earnings cycles.

Chapter 2

Related Work

2.1 News-Driven Prediction

News-based market prediction assumes new information shifts investor expectations and is reflected in prices within a limited time window. In practice, news is noisy and heterogeneous, with many articles only weakly related to the target firm. Therefore, a practical system must filter and structure news into events that link articles to symbols and event categories. In this pipeline, Agent 1 performs this role using keyword rules, symbol aliases, and event-type classification.

2.2 Event-Driven Strategies and Horizon Design

Event-driven strategies define a horizon to measure market reaction after an event. Short horizons reflect immediate reactions, while longer horizons capture delayed effects. Evaluating both 15-day and 40-day horizons tests signal stability across short- and medium-term windows. Agent 2 aligns t0 to the first trading day after publication, computes `ret_h` at the selected horizon, and assigns the up/down label.

2.3 Linear vs. Nonlinear Models

Logistic Regression is an interpretable baseline appropriate for near-linear relationships. However, news signals and price context often contain nonlinear interactions, so a Multi-Layer Perceptron (MLP) is used as a nonlinear baseline. Comparing LR and MLP helps reveal how much nonlinearity is required and guides future model upgrades.

2.4 Evaluation and Backtesting

Classification metrics (accuracy, AUC) quantify label prediction quality but do not directly measure trading utility. Therefore, the pipeline includes a simple backtest based on a probability threshold (p >= 0.55) and reports `n_trades`, `mean_ret_h`, and `hit_rate`. This dual evaluation separates predictive power from trading effectiveness.

2.4.1 Signal-to-Noise Considerations

News data often contain repeated or low-impact content. Simple keyword spotting can inflate recall but also creates noisy signals. The agent-based approach uses incremental filtering: initial relevance filtering at the crawler level, event classification at Agent 1, and reaction-based labeling at Agent 2. This pipeline reduces noise by requiring that an event maps to a valid trading horizon with a measurable return.

2.4.2 Feature Representation Trade-offs

Text features can be represented with TF-IDF, embeddings, or hashing. The pipeline chooses a hashing vectorizer with fixed dimension (128) to keep memory and runtime bounded. This is a pragmatic choice for large-scale crawling where vocabulary size can explode. The trade-off is lower interpretability, but downstream models benefit from a stable feature size that does not change between runs.

2.4.3 Evaluation Beyond Classification

Accuracy and AUC are reported, but the project explicitly emphasizes trading relevance through backtesting. The backtest uses a probability threshold to filter signals. This is aligned with typical event-driven trading workflows where only high-conviction signals are executed. The report uses both classification metrics and financial metrics to avoid over-optimizing for purely statistical measures.

Chapter 3

System Architecture

3.1 Architecture Overview

The pipeline follows a data collection → storage → multi-agent processing → modeling → prediction flow. The main entry script is `stock-agent-lab/run_all.py`, which calls `collect_news_default`, `collect_prices_default`, `collect_corp_actions_default`, and `run_agents` from `src/pipelines/*`. All data are stored in PostgreSQL, accessed via `PostgresStorage` in `src/storage/pg.py`.

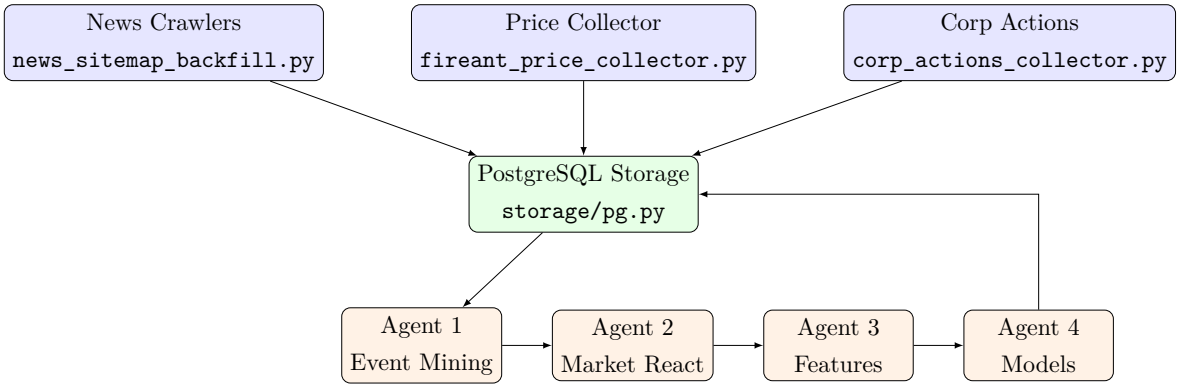

Figure 3.1: Overall VN30 pipeline architecture.

3.2 Pipeline Orchestration

`run_all.py` loads environment configuration through `load_settings()` in `src/config.py`, then executes the collection pipelines and the agent chain. The smoke mode limits the date range and symbol list for fast testing. The following snippet shows the core control flow in `run_all.py` (function `main`).

Listing 3.1: Orchestration logic in `run_all.py`

```
1 if args.smoke:
```

```
    end = date.today()
    start = end - timedelta(days=settings.smoke_days)
    collect_news_default(pg, settings, start, end, smoke_mode=True)
    collect_prices_default(pg, settings, start, end, smoke_mode=True)
    collect_corp_actions_default(pg, settings, start, end, smoke_mode=True)
    run_agents(pg, horizon_days=args.horizon)
else:
    start = datetime.strptime(args.start, "%Y-%m-%d").date()
    end = datetime.strptime(args.end, "%Y-%m-%d").date()
    collect_news_default(pg, settings, start, end, smoke_mode=False)
    collect_prices_default(pg, settings, start, end, smoke_mode=False)
    collect_corp_actions_default(pg, settings, start, end, smoke_mode=False)
    run_agents(pg, horizon_days=args.horizon)
    _run_coverage_and_retry(pg, settings, start, end)
```

`run_smoke.py` provides a reduced pipeline for quick checks. Both scripts finish with table counts (`table_counts`) and latest price date (`latest_price_date`) from `src/storage/pg.py`.

3.3 News Collection

The main crawler is `NewsSitemapBackfill` in `src/collectors/news_sitemap_backfill.py`. It runs in two phases: (1) URL discovery via sitemap or category crawling; (2) fetch and parse article content. Each source (Cafef, Vietstock, VnExpress) is configured in `SOURCE_CONFIG` with selectors for titles, dates, and content, plus allow/deny URL patterns. A hard filter can enforce relevance using symbol aliases and business keywords.

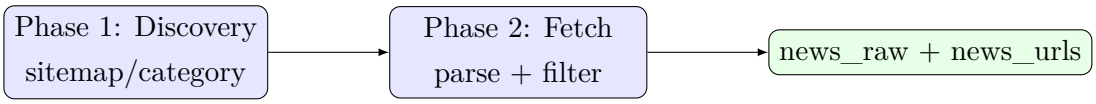

Figure 3.2: Two-phase news collection in `NewsSitemapBackfill`.

In addition to web crawling, `WorldNewsCollector` (`src/collectors/worldnews_collector.py`) can pull data via API when `WORLDNEWS_API_KEY` is set. It uses `monthly_ranges()` from `src/utils/time_ranges.py` and stores offsets in `collector_state`.

3.4 Price and Corporate Action Collection

Price data are collected from FireAnt via `FireAntPriceCollector` (`src/collectors/fireant_price_collector.py`). The collector supports XML/JSON responses, normalizes multiple field names, and stores rows in `prices`.

In full runs, `gap_fill_prices()` is invoked using missing trading ranges from `src/utils/coverage.py`.

Corporate actions are collected from VSD via `CorpActionsCollector` (`src/collectors/corp_actions_collector.py`). The collector crawls articles, extracts text, matches symbols via aliases, classifies action types using `ACTION_KEYWORDS`, and stores results in `corp_actions` with fields such as record_date, ex_date, effective_date, ratio, and cash_amount.

3.5 Storage and Schema

The schema is defined in `stock-agent-lab/db/init.sql`. Key tables include `news_raw`, `news_events`, `market_reactions`, `features_joint`, `models`, `predictions`, `prices`, and `corp_actions`. The `PostgresStorage` class in `src/storage/pg.py` provides upsert and query helpers for repeatable runs.

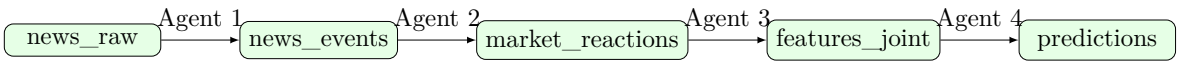

Figure 3.3: Core data flow across tables.

3.6 Agent Chain

3.6.1 Agent 1: Event Extraction

`Agent1NewsEvent` (`src/agents/agent1_news_event.py`) reads `news_raw`, merges content fields, detects symbols using `SYMBOL_ALIASES`, classifies event types with `EVENT_PATTERNS`, and writes `news_events`. It uses source sentiment if available, otherwise a heuristic lexicon, and writes detailed reports via `make_report_path()` and `write_report()` in `src/utils/reporting.py`.

3.6.2 Agent 2: Market Reaction Labeling

`Agent2MarketReact` (`src/agents/agent2_market_react.py`) aligns t0 to the first trading day after publish date, then computes ret_1d, ret_5d, and `ret_h` by horizon. Labels are set as `label_up` $= 1$ if `ret_h` > 0. It also computes 5-day volatility and max drawdown via `max_drawdown()` in `src/utils/features.py`.

3.6.3 Agent 3: Joint Feature Engineering

`Agent3FeaturesJoint` (`src/agents/agent3_features_joint.py`) combines news features (event type, sentiment, text hashing), price features (returns, volatility, RSI, MA

ratio, gap open), and corporate action features (record/ex-date windows). A HashingVectorizer with 128 dimensions is used for compact text representation.

3.6.4 Agent 4: Training and Prediction

`Agent4TrainPredictJoint` trains Logistic Regression and logs metrics/backtests in `models` and `predictions`. `Agent4TrainPredictNN` trains an MLP 96-48-24-1 using PyTorch when available, with sklearn MLPClassifier as fallback. Both agents split by time (80

3.6.5 Collector Robustness and Coverage

The news collector uses a two-phase approach to maximize coverage. Phase 1 discovers URLs using sitemap or category crawling. Phase 2 fetches articles, extracts content via source-specific selectors, and applies hard filters to ensure relevance. The coverage logic in `src/utils/coverage.py` tracks monthly counts for prices, news, and corporate actions, enabling gap detection and selective re-collection. This improves data completeness without fully rerunning the crawler.

3.6.6 Storage Design Rationale

PostgreSQL is used for its reliability and strong relational modeling. The schema is normalized by stage: raw data in `news_raw` and `prices`, structured events in `news_events`, labels in `market_reactions`, and engineered features in `features_joint`. This layered schema allows reprocessing at any stage without re-crawling, which is important for iterative model development.

3.6.7 Model Management and Reproducibility

Agent 4 stores models in the `artifacts/` directory and logs metadata to the `models` table. Each model stores training/validation ranges, feature dimensions, and metrics. This metadata is essential for reproducibility and enables later comparisons across horizons or model versions. The pipeline also writes detailed report files for each run, which are included in the report analysis.

3.6.8 Coverage Monitoring and Retry Logic

The coverage system in `src/utils/coverage.py` computes monthly counts for prices, news, and corporate actions and compares them against thresholds. In `run_all.py`, the function `_run_coverage_and_retry` logs gaps and performs a single retry pass: price gaps trigger `gap_fill_prices`, missing corporate actions trigger another crawl,

and missing news can trigger `collect_news_deepen_months` when `NEWS_DEEPEN=1`. mechanism makes the pipeline resilient to transient crawler failures or source downtime.

3.6.9 Collector State Management

The WorldNews collector uses the `collector_state` table to store offsets and completion flags for each symbol and month. This prevents redundant API calls and enables resumption. The same table is also used by the sitemap crawler to keep track of the last processed sitemap, reducing repeated discovery work. This design is a lightweight alternative to a full workflow scheduler and is sufficient for batch-oriented data collection.

3.6.10 Data Lineage Across Tables

Each stage keeps a clear lineage: `news_raw.id` is referenced in `news_events.evidence_json`, and `market_reactions.event_id` links labels back to events. Features in `features_joint` retain `event_id` and `symbol` so predictions can be traced to the originating article. This traceability is crucial for debugging and for explaining predictions in later analysis.

Chapter 4

Experimental Setup

4.1 Dataset and Coverage

Agent 1 produces `news_events` from `news_raw`. Report `stock-agent-lab/reports/agent1_report_20260131_172706.txt` indicates 148,182 news articles, 67,728 extracted events, and a 17.12

Table 4.1: News and event statistics (Agent 1).

Metric	Value	Report source
news_raw processed	148,182	agent1_report_20260131_172706
news_events upserted	67,728	agent1_report_20260131_172706
Article to event ratio	17.12%	agent1_report_20260131_172706

4.2 Market Reaction Labeling (Agent 2)

Agent 2 in `src/agents/agent2_market_react.py` generates market reactions by horizon. The `_resolve_t0_index` method aligns the first trading day after publish date, then `_calc_return` computes `ret_h` for labeling. The snippet below illustrates the horizon return calculation.

Listing 4.1: Horizon return calculation in `agent2_market_react.py`

```
@staticmethod
def _calc_return(closes: list[float], horizon: int) -> float | None:
    if len(closes) <= horizon:
        return None
    if closes[0] in (None, 0):
        return None
    return closes[horizon] / closes[0] - 1
```

Two Agent 2 reports correspond to horizons 15 and 40. The longer horizon has fewer samples and more `insufficient_horizon` drops.

Table 4.2: Market reaction counts and drop reasons (Agent 2).

Horizon	market_reactions	missing_publish_date	missing_t0	insufficient_closes	insufficient_horizon
40 days	64,946	198	25	105	2,454
15 days	65,963	198	25	105	1,437

Table 4.3: Label balance by horizon (Agent 2).

Horizon	Up	Down
40 days	35,262	29,684
15 days	45,834	20,129

4.3 Feature Engineering (Agent 3)

Agent 3 combines news, price, and corporate action features. Key feature groups include: (1) event type and sentiment; (2) text hashing features; (3) price returns, volatility, RSI, MA ratio, and gap open; (4) windows around record/ex dates. Figure 4.1 summarizes the feature flow.

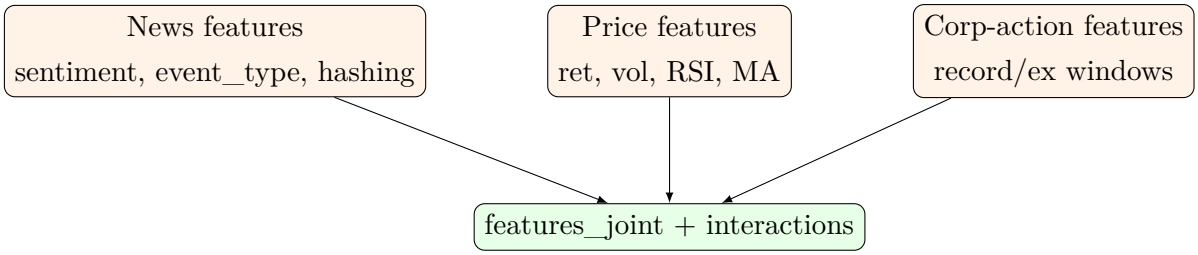

Figure 4.1: Feature construction flow in Agent 3.

Two Agent 3 reports show missingness across feature groups. Corporate action features are largely missing, reflecting sparse matches between events and corporate action dates.

Table 4.4: Feature missingness summary (Agent 3).

Horizon	features_produced	missing_news	missing_price	missing_corp
40 days	64,990	0	22,685	64,990
15 days	130,953	0	48,142	130,953

4.4 Model Training and Evaluation (Agent 4)

Agent 4 performs a chronological split: the earliest 80

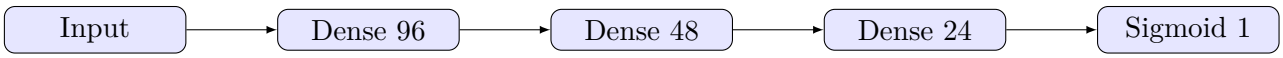

Figure 4.2: MLP 96-48-24-1 architecture in Agent 4 NN.

In `agent4_train_predict_nn.py`, the model is stored together with scaler, vectorizer, and (optionally) SVD for dimensionality reduction. The metrics are recorded in report files and stored in the `models` table.

4.4.1 Label Construction Details

Agent 2 aligns each event to the first available trading date (t0) after publish time. This decision ensures that the return horizon reflects actual tradeable windows. Returns are computed as `close[t0+H] / close[t0] - 1`, and the label is `1` if the return is positive. This definition avoids look-ahead bias and aligns with common event-study conventions.

4.4.2 Feature Engineering Rationale

Agent 3 creates three feature groups: (1) news/event features, (2) price context features, and (3) corporate action proximity features. News features encode event types and sentiment; price features capture momentum, volatility, and gaps; corporate action features capture calendar effects around record and ex-dates. Two interaction features are explicitly added to model simple non-linear effects without relying entirely on the MLP.

4.4.3 Training Protocol

The training protocol uses a chronological split (80/20) to reflect real-world forecasting. The logistic regression uses `class_weight=balanced` to mitigate label imbalance. The MLP uses a 96-48-24-1 architecture with dropout and early stopping when using PyTorch. This design balances model capacity with risk of overfitting and keeps inference lightweight.

4.4.4 Backtest Implementation

The backtest selects predictions with `proba_up >= 0.55`. For each selected trade, the corresponding `ret_h` is collected. The output includes number of trades, mean return,

and hit rate (fraction of positive returns). This backtest is intentionally simple to focus on signal quality rather than execution mechanics.

4.4.5 Handling Missingness

Agent 3 reports substantial missingness in price context (e.g., `ret_20d_pre`, `vol_20d_pre`) when there is insufficient pre-event price history. Instead of dropping these samples, missing values are imputed to zero in `Agent4TrainPredictJoint._sanitize_features`. This keeps the dataset size large and avoids selection bias, but it can reduce signal strength if missingness correlates with market conditions.

4.4.6 Feature Scaling and Dimensionality

The MLP path uses a `StandardScaler` and optionally `TruncatedSVD` when the sparse feature space exceeds 20,000 dimensions. In current runs the feature dimension is 178, so SVD is typically skipped. Scaling is still applied to stabilize optimization. This design keeps the training pipeline robust to feature growth if the hashing space or keyword sets are expanded in future experiments.

4.4.7 Validation Protocol Limitations

The validation set is strictly the most recent 20% of samples. This is appropriate for forecasting, but the split is still a single cut. A more robust protocol could use rolling or expanding windows to estimate stability over time. These alternatives are discussed as future work because they require more computation and more consistent data coverage.

Chapter 5

Results

5.1 Results Summary by Horizon

Based on the reports in `stock-agent-lab/reports/`, we summarize results for horizons 15 and 40. The mapping of reports to horizons is inferred from sample counts and the higher `insufficient_horizon` in the longer horizon. The tables below report LR and MLP metrics and backtest results.

Table 5.1: Logistic Regression results by horizon (Agent 4).

Horizon	Accuracy	Precision	Recall	AUC
40 days	0.5584	0.6356	0.6744	0.5324
15 days	0.4006	0.6663	0.1248	0.4955

Table 5.2: MLP 96-48-24-1 results by horizon (Agent 4 NN).

Horizon	Accuracy	Precision	Recall	AUC
40 days	0.5698	0.6603	0.6375	0.5647
15 days	0.5595	0.6550	0.6520	0.5382

Table 5.3: Backtest by horizon and model (Agent 4).

Horizon	Model	n_trades	mean_ret_h	hit_rate
40 days	LR	4,474	0.0873	0.6480
40 days	MLP	6,862	0.0803	0.6677
15 days	LR	786	0.0645	0.6959
15 days	MLP	14,233	0.0595	0.6595

5.2 Horizon 15 vs. 40

Horizon 40 shows lower but more stable AUC across LR and MLP. Horizon 15 produces more trades for MLP but with lower `mean_ret_h`. This suggests shorter horizons generate more signals but with smaller average payoff, while longer horizons yield fewer signals with larger `mean_ret_h`.

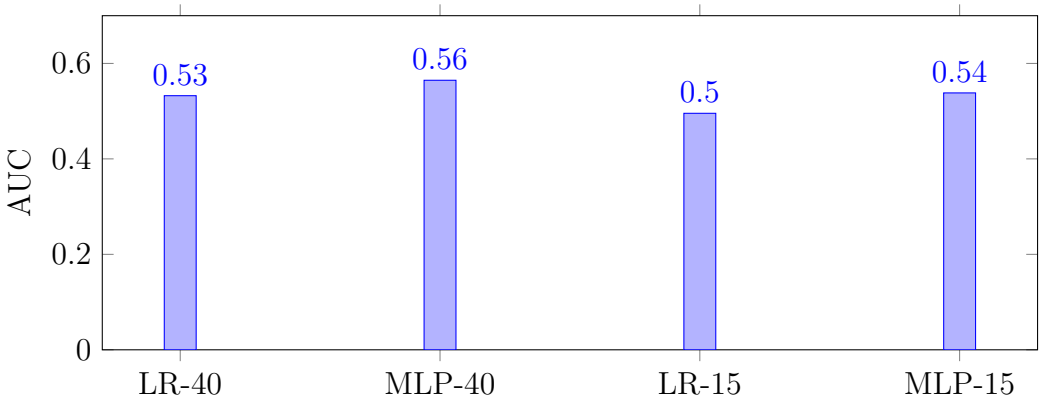

Figure 5.1: AUC comparison by horizon and model.

5.3 Market Reaction Statistics

Agent 2 reports `ret_h` summaries by horizon. For horizon 40, mean `ret_h` is 0.0284 with std 0.1210. For horizon 15, mean `ret_h` is 0.0383 with std 0.0751. Shorter horizons show lower dispersion in returns.

Table 5.4: `ret_h` summary from Agent 2.

Horizon	mean	std	min	max	median
40 days	0.0284	0.1209	-0.3902	1.1063	0.0180
15 days	0.0383	0.0751	-0.2880	0.6386	0.0330

5.4 LR vs. MLP

At horizon 40, MLP improves AUC and accuracy over LR, while LR slightly exceeds MLP on `mean_ret_h`. At horizon 15, MLP keeps higher accuracy, but LR achieves the best `hit_rate`. This indicates that nonlinear models better capture structure in the data, but classification metrics do not always align with trading utility.

5.5 Note on Horizon Mapping

The report filenames do not explicitly encode horizon. The mapping above is inferred from sample counts (longer horizon →fewer samples) and from `insufficient_horizon` in Agent 2 reports. Also note that `Agent3FeaturesJoint` does not filter by horizon in its SQL query, so if `market_reactions` contains multiple horizons, the features table may mix horizons. This is discussed further in Threats to Validity.

5.5.1 Metric Interpretation

AUC values around 0.5–0.56 indicate weak separability, which is expected in noisy financial data. The MLP improves AUC and accuracy in most settings, suggesting that non-linear combinations of news and price context are informative. However, the LR model sometimes yields higher mean_ret_h, showing that the most profitable signals may not align with the highest AUC.

5.5.2 Horizon Effects

The 15-day horizon produces more signals (higher `n_trades`) but a smaller mean return, while the 40-day horizon produces fewer signals with larger mean return. This is consistent with the idea that shorter horizons capture immediate reactions but may be noisier, whereas longer horizons capture more durable trends but occur less frequently.

5.5.3 Error Patterns and Practical Takeaways

The drop reasons in Agent 2 show that most losses come from insufficient horizon data. This implies that data coverage, rather than modeling, is a limiting factor. Improving price history completeness or extending data windows could increase usable samples and potentially improve model stability.

5.5.4 Precision–Recall Behavior

The LR model shows higher precision at horizon 15 but low recall, indicating it is conservative in predicting positives. The MLP provides a better balance, suggesting nonlinear features help recover more positive cases without dramatically sacrificing precision. This behavior aligns with the backtest results where MLP produces more trades but slightly lower mean return per trade.

5.5.5 Trade Frequency vs. Return Trade-off

A key practical trade-off is between `n_trades` and `mean_ret_h`. Higher trade frequency can dilute average returns and increase transaction costs, while fewer trades can miss opportunities. The horizon 40 results illustrate this trade-off: fewer signals, but higher mean return. The horizon 15 results illustrate the opposite.

5.5.6 Effect of Class Imbalance

Label balance differs by horizon. Horizon 15 has a higher fraction of positive labels (up moves), which can inflate naive accuracy if not handled carefully. The use of `class_weight=balanced` in LR partially mitigates this, but further calibration (e.g., threshold tuning) could improve the trade-off between hit rate and trade count.

Chapter 6

Discussion

6.1 Interpretation

The AUC values (0.50–0.56) indicate a challenging prediction task with weak signals. However, backtests show positive `mean_ret_h` and relatively high `hit_rate` (0.64–0.70), suggesting the model identifies a subset of trades with useful signal when applying a probability threshold. The discrepancy between low AUC and positive backtest can arise from skewed return distributions or from the model capturing a few profitable regions in feature space.

MLP improves AUC and accuracy over LR across both horizons, implying nonlinearity in the underlying data. Yet LR sometimes produces higher `mean_ret_h`, indicating that the classification objective does not always align with financial utility. This motivates optimization toward trading-aware metrics rather than pure classification loss.

6.2 Limitations and Lessons

- **Sparse corporate actions**: Agent 3 reports very high missingness in corp-action features, which reduces their contribution and shifts the model toward news and short-term price context.
- **News quality and parsing**: Source heterogeneity makes extraction noisy, causing drops in Agent 2 for missing t0 or insufficient horizon.
- **Simplified backtest**: The strategy uses a fixed probability threshold and ignores transaction costs, liquidity, and slippage, which may inflate performance.

6.3 Threats to Validity

- **Horizon ambiguity in Agent 3**: In `src/agents/agent3_features_joint.py`, the SQL query does not filter by horizon. If `market_reactions` contains multiple horizons, `features_joint` may mix horizons, affecting horizon-specific analysis and training unless `TRAIN_HORIZON_DAYS` is set consistently.
- **Temporal leakage risk**: Although the split is chronological, the backtest uses the validation window; if the threshold were tuned on validation, the results could be optimistic relative to true out-of-sample performance.
- **Coverage imbalance**: News coverage varies by symbol (see Agent 1 top symbols), which can bias the model toward heavily covered stocks.
- **Rule-based extraction bias**: Agent 1 relies on keyword and alias rules, so events that lack those tokens may be missed, causing type and symbol bias.

6.3.1 Why Backtest Can Look Better Than AUC

The backtest uses a high-probability threshold, effectively sampling only the most confident predictions. This can yield a higher hit rate even when overall AUC is modest. It also means that the model is used more as a filter than as a full classifier. This is a realistic scenario for event-driven trading where only top-ranked signals are executed.

6.3.2 Data Quality and Symbol Coverage

The top-10 symbol distribution in Agent 1 indicates uneven news coverage. This can bias the model toward frequently mentioned stocks, potentially reducing generalization. A possible mitigation is per-symbol normalization or stratified sampling during training.

6.3.3 Robustness Checks

Future work should include alternative splits (rolling windows), sensitivity to probability thresholds, and evaluation with transaction costs. These checks can validate whether the signals persist under realistic trading constraints.

6.3.4 Model Interpretability vs. Flexibility

LR provides a transparent linear decision boundary and is easier to inspect for feature contributions. MLP improves performance but sacrifices interpretability. For production use, it may be useful to maintain both models: LR for explanation and MLP for signal generation, especially if the goal is to present rationale to analysts.

6.3.5 Potential Leakage Risks

While the pipeline uses time-based splits, leakage can still occur if feature construction inadvertently includes post-event information. The current design avoids this by using only pre-event prices in Agent 3 (e.g., `pre_closes = closes[:-1]`). This is a positive design choice, but future feature additions should maintain this principle.

6.3.6 Sensitivity to Thresholds

The backtest relies on a fixed probability threshold (0.55). This choice affects both trade count and hit rate. A systematic threshold sweep could reveal more efficient operating points, but it would also risk overfitting to the validation set. A better approach is to evaluate threshold stability across rolling windows.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This report detailed an agent-based financial AI pipeline from data collection through structured events, market reaction labeling, joint feature engineering, and model training with backtesting. The system is fully implemented in `stock-agent-lab`, with `run_all.py` and `run_smoke.py` as entry scripts and pipeline modules under `src/pipelines/*`. Results show that the MLP generally improves AUC and accuracy over the LR baseline, while LR can yield higher `mean_ret_h` in some settings.

7.2 Future Work

- **Alpha labeling**: Replace the binary `ret_h`>0 label with excess returns over VNIndex or quantile-based labels.
- **Regime conditioning**: Add features for market regime (trend/range, volatility regime) to capture context.
- **Cycle features**: Explore Fourier/cycle features or seasonal indicators to capture periodic effects.
- **Pattern detection**: Add technical pattern features (e.g., cup-handle, breakout) or sequence models.
- **Incremental crawling**: Optimize collectors to reduce duplication and improve data freshness.

7.2.1 Summary of Findings

The pipeline demonstrates that a lightweight agent chain can transform noisy news streams into structured events, align them to market reactions, and train predictive models with meaningful backtest performance. The approach is modular and can be extended with minimal disruption, which is important for research workflows.

7.2.2 Roadmap for Extensions

A practical next step is to integrate regime-aware features and to test quantile-based labels that distinguish strong signals from weak signals. Another direction is to add cross-asset or sector-level signals to capture spillover effects between VN30 constituents.

Appendix A

Appendices

A.1 Environment Configuration

Key environment variables are loaded in `src/config.py` and used across pipelines. The table below lists the main parameters.

Table A.1: Key environment variables in the pipeline.

Variable	Purpose
DATABASE_URL	PostgreSQL connection
WORLDNEWS_API_KEY	WorldNews API key (optional)
WORLDNEWS_SOURCE_COUNTRY	Source country (default vn)
WORLDNEWS_LANGUAGE	Language (default vi)
WORLDNEWS_PAGE_SIZE	API page size
FIREANT_BASE_URL	Price data endpoint
HTTP_SLEEP_SECS	Delay between requests
HTTP_TIMEOUT_SECS	Request timeout
HTTP_MAX_RETRIES	Max retries
SMOKE_SYMBOLS	Symbols for smoke-test
SMOKE_DAYS	Days for smoke-test
NEWS_MODE	category/sitemap
NEWS_DEEPEN	Enable deepen mode
NEWS_DEEPEN_PER_SYMBOL	Extra URLs per symbol
LR_MAX_ITER	Max iterations for LR
LR_CLASS_WEIGHT	class_weight for LR (balanced)
TRAIN_HORIZON_DAYS	Horizon filter for Agent 4 training

A.2 VN30 Universe

The VN30 list is defined in `src/collectors/vn30_universe.py`.

Table A.2: VN30 symbols used in the pipeline.

ACB	BCM	BID	BVH	CTG
FPT	GAS	GVR	HDB	HPG
MBB	MSN	MWG	PLX	POW
SAB	SSI	STB	TCB	TPB
VCB	VHM	VIB	VIC	VJC
VND	VNM	VPB	VRE	VPI

A.3 Additional Code Snippets

Listing A.1: Rule-based event classification in `agent1_news_event.py`

```python
def _classify_event(self, text: str) -> str:
    lowered = text.lower()
    if any(keyword in lowered for keyword in CORP_ACTION_KEYWORDS):
        return "corp_action"
    for event_type, keywords in EVENT_PATTERNS.items():
        if any(keyword in lowered for keyword in keywords):
            return event_type
    return "other"
```

Listing A.2: Backtest logic in `agent4_train_predict_joint.py`

```python
@staticmethod
def _backtest(rows: list[dict[str, Any]], proba: np.ndarray) -> dict[str, Any]:
    returns = []
    hits = []
    for row, p in zip(rows, proba):
        if p >= 0.55:
            ret = row.get("ret_h")
            if ret is not None:
                returns.append(ret)
                hits.append(1 if ret > 0 else 0)
    if not returns:
        return {"n_trades": 0}
    return {
        "n_trades": len(returns),
        "mean_ret_h": float(np.mean(returns)),
        "hit_rate": float(np.mean(hits)),
    }
```

A.4 Example SQL Queries

The queries below are derived from `db/init.sql` and agent logic to illustrate verification steps in PostgreSQL.

Listing A.3: Monthly coverage query

```
SELECT symbol, to_char(date_trunc('month', date), 'YYYY-MM') AS ym, COUNT(*) AS cnt
FROM prices
WHERE date >= %(start)s AND date <= %(end)s
GROUP BY symbol, ym;
```

Listing A.4: Market reactions by horizon

```
SELECT reaction_id, event_id, symbol, t0, horizon_days, ret_h, label_up
FROM market_reactions
WHERE horizon_days = %(horizon)s
ORDER BY t0 ASC;
```

A.5 Event Type Taxonomy (Agent 1)

Table A.3 summarizes the event categories and representative keywords from `EVENT_PATTERNS` in `src/agents/agent1_news_event.py`.

Table A.3: Event categories and representative keywords.

Category	Example keywords (not exhaustive)
earnings	profit, earnings, revenue, loi nhuan, doanh thu
macro	interest rate, inflation, lai suat, ty gia
governance	board, ceo, appointment, bo nhiem
legal	lawsuit, regulator, investigation, khoi to, phat
mna	acquisition, merger, mua lai, sap nhap
rumor	rumor, tin don
corp_action	chot quyen, gdkhq, co tuc, esop
other	fallback class when no keyword matches

A.6 Corporate Action Types (Collector)

Table A.4 lists corporate action types defined in `ACTION_KEYWORDS` within `src/collectors/corp_actions_collector.py`.

Table A.4: Corporate action types and example triggers.

Action type	Example triggers
dividend_cash	co tuc, tien mat, chi tra co tuc
dividend_stock	co tuc bang co phieu, co phieu thuong
rights_issue	quyen mua, phat hanh them, chao ban
bonus_issue	thuong co phieu, phat hanh thuong
esop	esop, co phieu esop
split	chia tach
consolidation	gop co phieu
additional_listing	niem yet bo sung, giao dich bo sung
record_date	chot quyen, ngay dang ky cuoi cung
ex_rights	gdkhq, ex-date
other	fallback when no trigger matches

A.7 Feature Catalog (Agent 3)

This catalog summarizes the feature groups created in `src/agents/agent3_features_joint.py`. Hashing features `h_0`–`h_127` are produced by `HashingVectorizer(n_features=128)`. The list below focuses on explicit engineered features and interactions.

Table A.5: Explicit features in `features_joint`.

Feature	Description
`event_earnings`	One-hot flag for earnings event.
`event_governance`	One-hot flag for governance event.
`event_mna`	One-hot flag for M&A event.
`event_legal`	One-hot flag for legal event.
`event_macro`	One-hot flag for macro event.
`event_rumor`	One-hot flag for rumor event.
`event_corp_action`	One-hot flag for corp action event.
`event_other`	One-hot flag for other events.
`sentiment`	Sentiment score from source or heuristic.
`impact_hint`	Heuristic impact score based on event type and numbers.
`title_len`	Length of title text.
`text_len`	Length of full article text.
`kw_profit`	Keyword count for profit.

Feature	Description
kw_revenue	Keyword count for revenue.
kw_dividend	Keyword count for dividend.
kw_investigation	Keyword count for investigation.
kw_interest_rate	Keyword count for interest rate.
kw_inflation	Keyword count for inflation.
kw_lai_suat	Keyword count for lai suat.
kw_doanh_thu	Keyword count for doanh thu.
ret_5d_pre	Pre-event return over 5 trading days.
ret_20d_pre	Pre-event return over 20 trading days.
vol_20d_pre	Pre-event volatility over 20 trading days.
volume_z_20d	Z-score of volume versus 20-day history.
ma_ratio_5_20	Moving-average ratio (MA5 / MA20 - 1).
rsi_14_pre	RSI(14) using pre-event prices.
gap_open_pre	Gap between open(t0) and previous close.
days_since_prev_record	Days since last record date.
days_to_next_record	Days to next record date.
within_window_record_7	Record date within 7 days window.
within_window_record_14	Record date within 14 days window.
within_window_record_30	Record date within 30 days window.
within_window_ex_7	Ex-date within 7 days window.
within_window_ex_14	Ex-date within 14 days window.
within_window_ex_30	Ex-date within 30 days window.
count_corp_actions_last_180d	Count of recent corp actions in last 180 days.
last_action_dividend_cash	One-hot of most recent corp action type.
last_action_dividend_stock	One-hot of most recent corp action type.
last_action_rights_issue	One-hot of most recent corp action type.
last_action_bonus_issue	One-hot of most recent corp action type.
last_action_esop	One-hot of most recent corp action type.
last_action_split	One-hot of most recent corp action type.
last_action_consolidation	One-hot of most recent corp action type.
last_action_additional_listing	One-hot of most recent corp action type.
last_action_record_date	One-hot of most recent corp action type.
last_action_ex_rights	One-hot of most recent corp action type.
last_action_other	One-hot of most recent corp action type.
sentiment_x_record_14	Interaction: sentiment record_14 flag.
pre_momentum_20d_x_ex_14	Interaction: 20d momentum ex_14 flag.

A.8 Collector Source Configuration

Table A.6 summarizes key sources configured in `SOURCE_CONFIG` inside `src/collectors/news_sitemap_backfill.py`.

Table A.6: News sources and collection modes.

Source	Base domain	Modes
cafef	https://cafef.vn	sitemap or category crawl
vietstock	https://vietstock.vn	sitemap or category crawl
vnexpress	https://vnexpress.net	sitemap or category crawl

A.9 Run Commands (Examples)

The following commands mirror the intended usage of the entry scripts. They are illustrative; actual dates and horizons are controlled by runtime configuration.

Listing A.5: Example full run

```
python run_all.py --start 2023-01-01 --end 2026-01-31 --horizon 40
```

Listing A.6: Example smoke run

```
python run_smoke.py
```

A.10 Database Schema Reference

This section summarizes the PostgreSQL schema defined in `stock-agent-lab/db/init.sql`. The table focuses on field names and their roles in the pipeline.

Table A.7: Schema reference (selected fields).

Table	Column	Type	Notes
news_raw	id	BIGINT	Primary key from source.
news_raw	source	TEXT	News source name.
news_raw	publish_date	TIMESTAMPTZ	Article publish time.

Table	Column	Type	Notes
news_raw	title	TEXT	Article title.
news_raw	summary	TEXT	Short summary.
news_raw	text	TEXT	Full content.
news_raw	sentiment	DOUBLE	Optional sentiment score.
news_urls	url	TEXT	URL key.
news_urls	source	TEXT	Source name.
news_urls	status	TEXT	new/fetched/skipped.
news_urls	lastmod	DATE	Sitemap lastmod.
news_events	event_id	TEXT	Primary key.
news_events	symbol	VARCHAR	VN30 symbol.
news_events	event_type	VARCHAR	Event category.
news_events	sentiment	DOUBLE	Sentiment score.
news_events	impact_hint	DOUBLE	Heuristic impact.
market_reactions	reaction_id	TEXT	Primary key.
market_reactions	event_id	TEXT	FK to news_events.
market_reactions	t0	DATE	First trading day after event.
market_reactions	horizon_days	INTEGER	Horizon in days.
market_reactions	ret_h	DOUBLE	Horizon return.
market_reactions	label_up	INTEGER	Binary label.
features_joint	event_id	TEXT	Primary key, FK to news_events.
features_joint	horizon_days	INTEGER	Horizon for feature row.
features_joint	feature_json	JSONB	Feature map.
prices	symbol	VARCHAR	VN30 symbol.
prices	date	DATE	Trading day.
prices	open	DOUBLE	Open price.
prices	close	DOUBLE	Close price.
prices	volume	DOUBLE	Volume.
corp_actions	action_id	TEXT	Primary key.
corp_actions	symbol	VARCHAR	VN30 symbol.
corp_actions	action_type	TEXT	Corp action type.
corp_actions	record_date	DATE	Record date.
corp_actions	ex_date	DATE	Ex-rights date.
corp_actions	effective_date	DATE	Effective date.

Table	Column	Type	Notes
models	model_id	TEXT	Primary key.
models	meta_json	JSONB	Metrics/backtest metadata.
predictions	pred_id	TEXT	Primary key.
predictions	model_id	TEXT	FK to models.
predictions	proba_up	DOUBLE	Predicted probability.
collector_state	source	TEXT	Collector name.
collector_state	cursor	TEXT	State cursor key.
collector_state	cursor_value	TEXT	Cursor value.
coverage_monthly	key	TEXT	Composite key.
coverage_monthly	domain	TEXT	Domain (news/prices/corp).
coverage_monthly	count	INTEGER	Monthly count.

A.11 Agent Pseudocode

The following pseudocode summarizes the logic for each agent at a high level.

Listing A.7: Agent 1 pseudocode

```
for article in news_raw:
    text = title + summary + text
    symbols = detect_symbols(text)
    if not symbols:
        continue
    event_type = classify_event(text)
    sentiment = source_sentiment or heuristic_sentiment(text)
    impact_hint = estimate_impact(event_type, text)
    for symbol in symbols:
        event_id = hash(symbol, publish_date, url)
        upsert news_events
```

Listing A.8: Agent 2 pseudocode

```
for event in news_events:
    t0 = first_trading_day_after_publish(event)
    if t0 missing:
        continue
    ret_h = return(close[t0+H], close[t0])
    if ret_h missing:
        continue
    label_up = 1 if ret_h > 0 else 0
```

```
    compute vol_5d, dd_h
    upsert market_reactions
```

Listing A.9: Agent 3 pseudocode

```
for event in news_events with reaction:
    news_features = build_news_features(event)
    price_features = build_price_features(symbol, t0)
    corp_features = build_corp_action_features(symbol, t0)
    feature_json = merge + interactions
    upsert features_joint
```

Listing A.10: Agent 4 pseudocode

```
rows = join(features_joint, market_reactions)
rows = filter_by_horizon_if_set
split = time_based_split(rows, 80/20)
train model (LR or MLP)
compute metrics and backtest
store model + predictions
```

A.12 Additional Metric Tables

The following tables provide compact cross-horizon comparisons for quick reference.

Table A.8: Accuracy and AUC by model and horizon.

Model/Horizon	Accuracy	AUC	Notes
LR (40)	0.5584	0.5324	Baseline linear model
MLP (40)	0.5698	0.5647	Nonlinear MLP
LR (15)	0.4006	0.4955	Lower accuracy, high precision
MLP (15)	0.5595	0.5382	Better balance

Table A.9: Backtest summary by model and horizon.

Model/Horizon	n_trades	mean_ret_h	hit_rate
LR (40)	4,474	0.0873	0.6480
MLP (40)	6,862	0.0803	0.6677
LR (15)	786	0.0645	0.6959
MLP (15)	14,233	0.0595	0.6595

A.13 Collector Configuration Examples

These examples illustrate how crawler behavior can be adjusted with environment variables.

Listing A.11: Example environment overrides

```
NEWS_MODE=category
NEWS_DEEPEN=1
NEWS_DEEPEN_PER_SYMBOL=30
MAX_URLS_PER_SOURCE=3000
HTTP_TIMEOUT_SECS=30
HTTP_MAX_RETRIES=5
LR_MAX_ITER=500
LR_CLASS_WEIGHT=balanced
TRAIN_HORIZON_DAYS=40
```

A.14 Coverage Diagnostic Queries

Listing A.12: Coverage below threshold query

```
SELECT domain, symbol, year_month, count
FROM coverage_monthly
WHERE count < 5
ORDER BY domain, symbol, year_month;
```

Listing A.13: Monthly news counts by symbol

```
SELECT symbol, to_char(date_trunc('month', publish_date), 'YYYY-MM') AS ym, COUNT(*)
     AS cnt
FROM news_events
GROUP BY symbol, ym
ORDER BY ym ASC;
```