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 backtest 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 presents the prediction methodology, combining dataset coverage, pipeline orchestration, system architecture, feature collection, and a detailed agent-by-agent explanation. Chapter 4 merges experimental setup, measurements, and results, including horizon comparisons and backtest summaries. Chapter 5 discusses interpretation, limitations, and threats to validity. Chapter 6 concludes and outlines future work.

1.4.1 Problem Definition in Operational Terms

This project frames news-driven prediction as a structured labeling task with explicit market horizons. Each article is first linked to VN30 symbols using alias maps in `src/collectors/vn30_universe.py`, which is necessary because Vietnamese news often mentions company names instead of tickers. Once the target symbol is known, the article is converted into a structured event with an event type, sentiment signal, and a lightweight impact score. This event structure serves as a bridge between unstructured text and numerical modeling. The prediction target is the horizon return `ret_h`, measured from the first tradeable day after publication (t0) to t0+H. The label `label_up` is defined as 1 if `ret_h>0`, otherwise 0. This definition ensures that labels are aligned with a real trading window and avoids leakage from future prices. In practice, this framing makes the task interpretable: the model is asked to decide whether the market will rise over a defined horizon after a specific news event. It also allows the system to report detailed drop reasons when labels cannot be computed, which is important for evaluating data coverage and diagnosing gaps.

1.4.2 Why an Agent Chain

The agent chain is a deliberate design choice to keep each transformation simple, auditable, and reusable. Agent 1 only concerns itself with turning raw text into structured events; it has no dependency on price data, which keeps the extraction logic clean and debuggable. Agent 2 takes those events and aligns them to the price series to produce horizon labels, encapsulating all market-reaction logic in a single module. Agent 3 builds a unified feature set that merges text, price context, and corporate actions without performing any modeling. Finally, Agent 4 is dedicated to model training and evaluation. This separation makes the pipeline robust: if price data are missing, only Agent 2?s outputs are affected, while the upstream event extraction remains valid. It also encourages experimentation?new features or models can be tested without touching the collectors. In a research setting, this structure supports reproducibility and transparency, because each stage has its own report output and defined input/output tables. The agent chain mirrors how analysts work: interpret information, measure reaction, extract signals, and then decide on a trading action.

1.4.3 Data Window and Scope Justification

The experimental window spans January 2023 to January 2026, a period that includes multiple market conditions and provides enough events for model training without being so long that market structure shifts dominate the results. The VN30 universe is chosen to reduce liquidity issues, as these large-cap stocks are more frequently covered by the news and have reliable price histories. The two prediction horizons, 15 and 40 days, are selected to balance short-term reactions and medium-term trends. A shorter horizon tests whether the market reacts quickly to news, while the longer horizon captures delayed digestion of information and secondary effects. This dual-horizon design allows the study to compare how sensitive the models are to the timing of reactions and whether signals persist beyond the initial news impact. The scope is intentionally constrained to a well-defined universe and timeframe to ensure that conclusions are grounded in data coverage rather than inconsistent sources or missing observations.

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

Prediction Methodology

3.1 Prediction Methodology Choice

The pipeline adopts an event-driven prediction methodology: news articles are mapped to VN30 symbols, converted to structured events, and then aligned with subsequent price movements over a fixed horizon. This approach is chosen because it separates semantic signal extraction (from text) from market reaction measurement (from prices), allowing clear evaluation of each stage. It also enables a modular agent chain in which each agent performs a single, auditable responsibility. In practice, this methodology mirrors how analysts reason about markets: a discrete piece of information is published, a target company is affected, and the market reacts within a definable window. By explicitly encoding this sequence, the system can be evaluated in parts (extraction quality, labeling quality, model quality) rather than only as a black box.

A key benefit of the event-driven framing is the ability to define and compare horizons. Short horizons test whether the market reacts quickly, while longer horizons test whether effects persist or diffuse. The design also supports extensibility: additional signal sources (e.g., sector-level data or macro indicators) can be attached to the same event backbone. Finally, the methodology is computationally efficient: it uses deterministic labeling and simple data joins, making it feasible to run repeatedly and to scale to larger time windows without excessive recomputation.

3.2 Dataset Coverage and Scope

The experimental window is January 2023 to January 2026. VN30 is used as the universe because of liquidity and stability. News coverage is collected from Cafef, Vietstock, and VnExpress; prices from FireAnt; corporate actions from VSD. The coverage system computes monthly counts to detect gaps and guide selective retries. This window captures multiple market phases, including periods of volatility and recovery, which

helps evaluate whether the pipeline remains robust across different regimes.

Coverage is a central concern because missing price data can invalidate labels, and missing news reduces event diversity. The pipeline therefore computes monthly coverage statistics for each symbol and domain (news, prices, corp actions). These metrics are stored in the `coverage_monthly` table and used to trigger targeted retries rather than full recrawls. This design keeps the data pipeline both efficient and transparent. The result is a dataset that is large enough for modeling yet sufficiently controlled to support reproducible experiments.

Table 3.1: 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

3.3 Pipeline Orchestration

The main entry script is `run_all.py`. It loads configuration, runs the collectors, then executes the agent chain in `src/pipelines/run_agents.py`. The smoke mode is a fast check with a short date range and limited symbols. This orchestration design keeps the pipeline simple while still allowing reprocessing by horizon and date range.

The orchestration layer also enforces ordering constraints: events are extracted only after news is stored, reactions are computed only after prices are available, and features are built only after labels exist. This ordering prevents partial or inconsistent datasets. In addition, `run_all.py` includes a coverage retry mechanism, which attempts a single pass of gap filling after the initial run. This balances completeness with runtime efficiency and reduces the need for manual intervention.

Listing 3.1: Orchestration logic in `run_all.py`

```
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)
```

3.4 System Architecture

The system architecture integrates collectors, storage, and the agent chain. Figure 3.1 shows the end-to-end flow. Figure 3.2 details the two-phase news crawler. Figure 3.3 shows how data moves across tables.

From an implementation perspective, the collectors operate independently and store results in dedicated tables. Agents then read from these tables and write derived outputs. This layered design enables checkpointing: if a later stage fails, earlier data remain intact. It also encourages reusability, as the same data can be used to test different feature sets or models without re-crawling. Overall, the architecture emphasizes reproducibility, traceability, and modularity?three properties that are essential for research-grade pipelines.

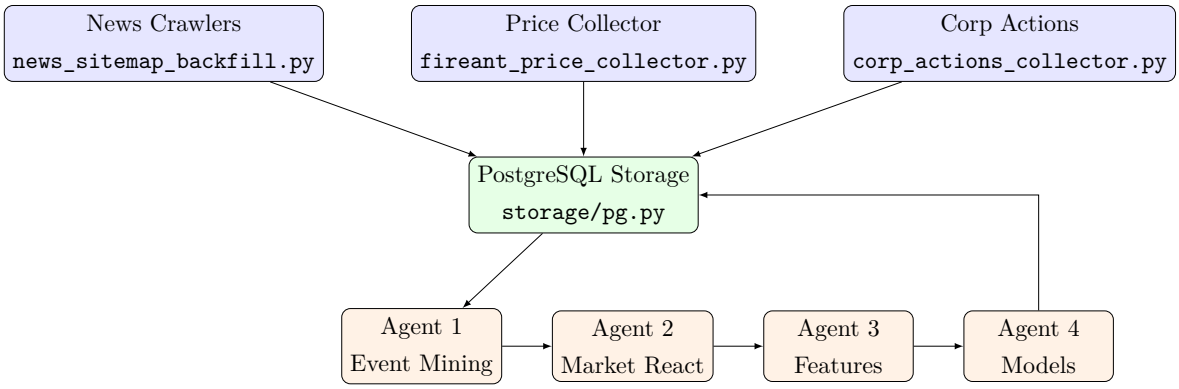

Figure 3.1: Overall VN30 pipeline architecture.

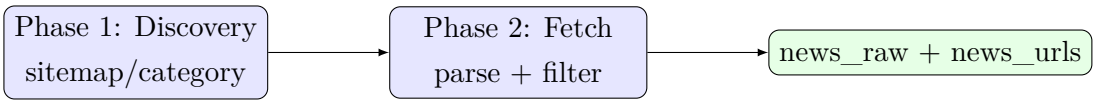

Figure 3.2: Two-phase news collection in `NewsSitemapBackfill`.

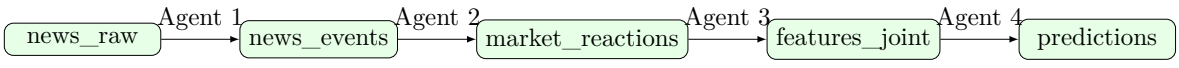

Figure 3.3: Core data flow across tables.

3.5 Feature Collection and Engineering

Feature collection is centralized in Agent 3, which merges news/event features, price context, and corporate action proximity. A HashingVectorizer creates compact text features (128 dimensions), while price features capture momentum, volatility, RSI, and gaps. Corporate action features encode calendar windows around record and ex-dates. Two interaction terms are added to model simple non-linearities.

This feature strategy balances expressiveness and stability. Hashing avoids vocabulary drift, so feature dimensions remain consistent across runs. Technical indicators are computed using pre-event prices to avoid leakage. Corporate action features allow the model to capture calendar-driven effects, such as dividend announcements or ex-rights impacts. The interaction features add limited nonlinearity without relying entirely on deep models. Together, these choices produce a feature set that is compact, interpretable, and suitable for both linear and nonlinear classifiers.

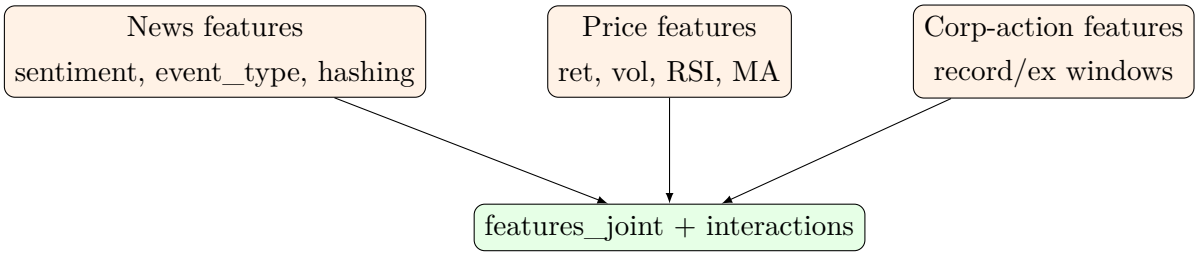

Figure 3.4: Feature construction flow in Agent 3.

3.6 Agent Chain (Inputs, Outputs, and Logic)

3.6.1 Agent 1: Event Extraction

Input: `news_raw` rows (title, summary, text, publish_date).

Output: `news_events` rows with event_id, symbol, event_type, sentiment, impact_hint, and evidence_json.

How it works: Agent 1 converts unstructured news into structured events. It first concatenates title, summary, and full text to form a single text blob, then detects VN30 symbols using `SYMBOL_ALIASES`. This alias map is critical because many articles mention company names instead of ticker codes, and a strict ticker-only match would miss a large fraction of events. Once symbols are detected, the agent assigns an event type by scanning keyword patterns in `EVENT_PATTERNS`, with a special rule for corporate actions using `CORP_ACTION_KEYWORDS`. Sentiment is taken from the source if available; otherwise a lightweight heuristic lexicon counts positive and negative words to produce a signed score. A separate `impact_hint` is computed by combining a base weight for the event type with a small boost for numeric mentions, reflecting the intuition that

quantified results (e.g., earnings figures) tend to carry higher impact. Each output event gets a stable hash-based `event_id` so that reruns are idempotent. The agent writes a report that summarizes coverage, top symbols, and event-type distribution, which is later used to verify data quality and detect source bias.

3.6.2 Agent 2: Market Reaction Labeling

Input: `news_events` + `prices`.
Output: `market_reactions` with ret_1d, ret_5d, ret_h, vol_5d, dd_h, label_up.
How it works: Agent 2 measures how the market reacts to each event over a fixed horizon. It aligns each event to the first trading day on or after the publish date (t0), which is essential for avoiding look-ahead bias and ensuring returns are tradable. It then computes ret_1d, ret_5d, and ret_h as relative returns from t0 to the specified horizon. If the series does not have enough trading days, the event is dropped and the reason is counted in a detailed drop report. The binary label `label_up` is defined as 1 when ret_h is positive and 0 otherwise. To capture risk context, the agent also computes 5-day volatility and maximum drawdown over the horizon using utilities in `src/utils/features.py`. This provides a richer description of event impact beyond simple direction, and it enables later analysis of how signal strength correlates with volatility or drawdown. The resulting `market_reactions` table serves as the ground truth for supervised learning in Agent 4.

Listing 3.2: 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
```

3.6.3 Agent 3: Joint Feature Engineering

Input: `news_events`, `market_reactions`, `prices`, `corp_actions`.
Output: `features_joint` rows with `feature_json`.
How it works: Agent 3 performs joint feature engineering to bridge text signals and price context. Event features include one-hot indicators for event types, sentiment, impact hint, and text length statistics. It also uses a `HashingVectorizer` (128 dimensions) to encode keyword-like signals from the title and summary without maintaining a growing vocabulary. Price-context features are computed strictly using pre-event data: short- and medium-term returns (5d, 20d), volatility over 20 days, RSI(14), moving

average ratio (MA5/MA20-1), and the gap between the open at t0 and the previous close. Corporate action features measure proximity to record and ex-dates within 7/14/30-day windows and encode the most recent action type. Two interaction terms are added to capture simple non-linear effects between sentiment and corporate action windows, and between momentum and ex-date proximity. Missing values are left as `None` at this stage and later sanitized to zero in Agent 4. This design keeps feature extraction explicit and reproducible, while allowing both linear and nonlinear models to exploit cross-domain information.

3.6.4 Agent 4: Training and Prediction

Input: `features_joint` joined with `market_reactions`.
Output: Models stored in `artifacts/`, metadata in `models`, and predictions in `predictions`.
How it works: Agent 4 trains two model families: a Logistic Regression baseline and a multi-layer perceptron (MLP). The dataset is split chronologically (80% train, 20% validation) to simulate forward prediction. Features are vectorized with `DictVectorizer`; missing values are sanitized to 0.0 for numerical stability. The Logistic Regression model uses `class_weight=balanced` to mitigate label imbalance and `LR_MAX_ITER` for convergence control. This baseline is intentionally simple and interpretable: it produces calibrated probabilities that are easy to threshold and compare across horizons.

For the MLP, Agent 4 uses a 96-48-24-1 architecture with ReLU activations and a sigmoid output. When PyTorch is available, it applies dropout and early stopping based on validation AUC or loss, which helps prevent overfitting. If PyTorch is not available, it falls back to sklearn's `MLPClassifier` with early stopping and L2 regularization. Features are standardized with `StandardScaler`; if the feature space is large, optional SVD reduces dimensionality. After training, both models output probabilities for the validation window. These probabilities are evaluated with accuracy, precision, recall, and AUC, then fed into a simple backtest: only predictions with `proba_up >= 0.55` are executed, and `n_trades`, `mean_ret_h`, and `hit_rate` are computed. Models and metadata are stored in the database for reproducibility and comparison across horizons.
Model implementation excerpts:

Listing 3.3: Logistic Regression training in `agent4_train_predict_joint.py`

```
model = LogisticRegression(max_iter=self.max_iter, class_weight=class_weight)
model.fit(X_train, y_train)
proba = model.predict_proba(X_val)[:, 1]
preds = (proba >= 0.5).astype(int)
metrics = self._evaluate(y_val, preds, proba)
backtest = self._backtest(rows[split_idx:], proba)
```

Listing 3.4: PyTorch MLP architecture in `agent4_train_predict_nn.py`

```
model = nn.Sequential(
    nn.Linear(X_train.shape[1], 96),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(96, 48),
    nn.ReLU(),
    nn.Linear(48, 24),
    nn.ReLU(),
    nn.Linear(24, 1),
    nn.Sigmoid(),
).to(device)
```

3.7 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`. This mechanism makes the pipeline resilient to transient crawler failures or source downtime.

3.8 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.

3.9 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.

Chapter 4

Experimental Setup, Measurements and Results

4.1 Experimental Setup

The experimental setup uses two horizons (15 and 40 days) and a time-based split to simulate forward prediction. Features are generated by Agent 3 and labels by Agent 2. The evaluation uses accuracy, precision, recall, and AUC, supplemented by a simple backtest.

4.1.1 Label Balance and Data Filtering

The longer horizon has more `insufficient_horizon` drops and fewer total reactions. Label balance differs by horizon, which motivates using `class_weight=balanced` in LR.

Table 4.1: 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 4.2: Label balance by horizon (Agent 2).

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

4.1.2 Training Protocol

Agent 4 performs a chronological split: the earliest 80% of samples are used for training and the most recent 20% for validation. The Logistic Regression baseline uses `class_weight=balanced` and `LR_MAX_ITER`. The MLP uses a 96-48-24-1 architecture. Both output probabilities used in backtest thresholding.

The chronological split mimics deployment: models are trained on past data and evaluated on future data. This prevents leakage that can occur with random shuffling. The LR baseline is deliberately simple, providing a stable comparison point across runs and horizons. The MLP adds capacity to capture interactions among text features and price context. Both models share the same feature pipeline so that performance differences reflect model capacity rather than feature availability. The evaluation metrics (accuracy, precision, recall, AUC) are complemented by backtesting to ensure the predictive gains translate to trading-relevant outcomes. the drop-reason report is not only a diagnostic but also a proxy for coverage quality: large `insufficient_horizon` counts indicate missing long-range prices, which can be improved with deeper price collection or a longer historical range.

4.1.3 Training Protocol

Agent 4 performs a chronological split: the earliest 80% of samples are used for training and the most recent 20% for validation. The Logistic Regression baseline uses `class_weight=balanced` and `LR_MAX_ITER`. The MLP uses a 96-48-24-1 architecture. Both output probabilities used in backtest thresholding.

4.2 Measurements and Results

Results are summarized by horizon and model. The MLP improves AUC/accuracy, while LR sometimes yields higher mean_ret_h.

Table 4.3: 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 4.4: 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 4.5: 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

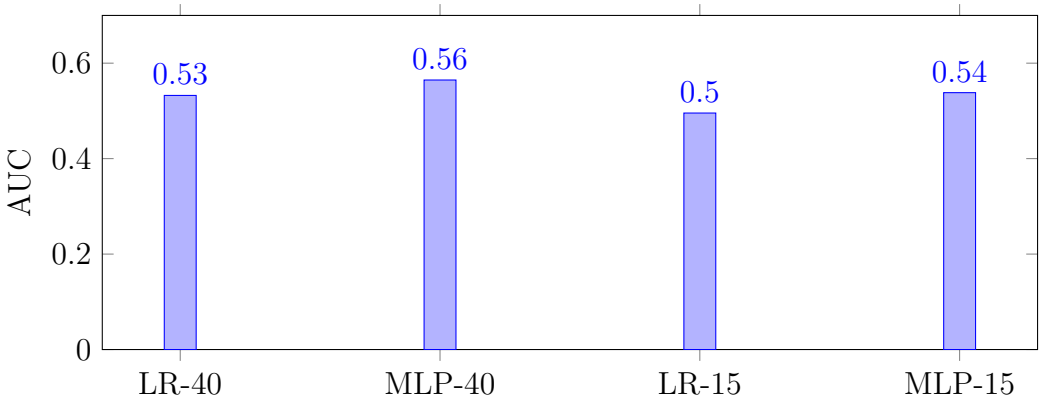

Figure 4.1: AUC comparison by horizon and model.

Table 4.6: 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

Chapter 5

Discussion

5.1 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.

5.2 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.

5.3 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.

5.3.1 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.

This phenomenon is common in financial prediction: a model can have low overall discriminative power but still produce a small subset of profitable signals. When using a probability threshold, the model effectively focuses on extreme cases where the signal is strongest, which inflates hit rate and mean return within that subset. However, this also reduces trade count, and the risk is that such results may not generalize if the threshold is optimized on the validation set. A robust approach is to test multiple thresholds across rolling windows and to report sensitivity bands rather than a single point estimate.

5.3.2 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.

Coverage imbalance implies that certain symbols dominate the training signal. This can skew model behavior toward highly mentioned stocks, especially if those symbols also have stronger or more consistent price trends. One mitigation is to enforce per-symbol sampling or to introduce symbol-specific normalization. Another approach is to explicitly model symbol embeddings or sector-level features so the model can distinguish between symbol-specific dynamics and broader market effects.

5.3.3 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.

Robustness checks should include alternative splits (rolling or expanding), outlier handling for extreme returns, and sensitivity to missing corporate action data. It is also useful to compare performance when excluding macro or governance events to see if certain event types dominate predictive performance. These checks help clarify whether the pipeline learns a generalizable signal or a narrow artifact tied to a specific data source or time period.

5.3.4 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.

5.3.5 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.

5.3.6 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 6

Conclusion and Future Work

6.1 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.

6.2 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.

6.2.1 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.

The agent-based pipeline provides a clear chain from raw data to predictions, which simplifies auditing and allows targeted improvements. The separation of responsibilities also makes it easier to extend or replace individual components (for example, swapping the text hashing with a transformer-based encoder) without rebuilding the entire system. The results highlight both the promise and the limitations of news-driven prediction in noisy markets.

6.2.2 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.

A practical roadmap should prioritize improvements that increase signal quality without dramatically increasing computational cost. Examples include better entity linking, richer sentiment modeling, and regime-aware features. On the modeling side, calibration and cost-sensitive learning could align predictions more closely with trading objectives. Finally, improving data freshness and reducing crawler duplication would make the pipeline more viable for near-real-time analysis.

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;
```

A.15 Agent 4 Model Implementations

Below are concise code excerpts illustrating the Logistic Regression and MLP implementations used in Agent 4.

Listing A.14: Logistic Regression training in `agent4_train_predict_joint.py`

```
model = LogisticRegression(max_iter=self.max_iter, class_weight=class_weight)
model.fit(X_train, y_train)
proba = model.predict_proba(X_val)[:, 1]
preds = (proba >= 0.5).astype(int)
metrics = self._evaluate(y_val, preds, proba)
```

```
backtest = self._backtest(rows[split_idx:], proba)
```

Listing A.15: PyTorch MLP architecture in `agent4_train_predict_nn.py`

```
model = nn.Sequential(
    nn.Linear(X_train.shape[1], 96),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(96, 48),
    nn.ReLU(),
    nn.Linear(48, 24),
    nn.ReLU(),
    nn.Linear(24, 1),
    nn.Sigmoid(),
).to(device)
```

Listing A.16: MLP training loop (excerpt)

```
optimizer = optim.Adam(model.parameters(), lr=1e-3, weight_decay=1e-4)
criterion = nn.BCELoss()
for epoch in range(epochs):
    model.train()
    indices = torch.randperm(X_train_t.size(0))
    for start in range(0, X_train_t.size(0), batch_size):
        batch_idx = indices[start : start + batch_size]
        batch_x = X_train_t[batch_idx]
        batch_y = y_train_t[batch_idx]
        optimizer.zero_grad()
        outputs = model(batch_x)
        loss = criterion(outputs, batch_y)
        loss.backward()
        optimizer.step()
```