HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND 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

February 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.

1.5 Problem Definition and Problem Decomposition

This section provides a rigorous formulation of the news-driven stock prediction problem, decomposes it into tractable sub-problems, and establishes the theoretical and practical justification for the approach taken in this system.

1.5.1 Context and Motivation for News-Driven Prediction

Financial markets are information-driven systems where asset prices reflect the collective expectations of market participants based on available information. In efficient market theory, new information is rapidly incorporated into prices, creating opportunities for prediction when information asymmetries exist or when market participants process information with varying speeds and interpretations. News articles represent one of the primary channels through which fundamental information about companies, industries, and macroeconomic conditions reaches market participants. In the Vietnamese stock market, particularly for the VN30 index constituents, corporate announcements, governance changes, earnings releases, and regulatory events are frequently disseminated through financial news portals before being fully reflected in trading prices. This creates a potential window for prediction, provided that the news can be systematically processed and linked to subsequent price movements.

The challenge in news-driven stock prediction lies in the fundamental disconnect between the unstructured, heterogeneous nature of news text and the structured, quantitative requirements of predictive modeling. News articles vary widely in quality, relevance, sentiment polarity, and information content. A single article may mention multiple companies, discuss events with varying degrees of materiality, and contain both factual reporting and speculative commentary. Furthermore, the Vietnamese financial news ecosystem presents additional challenges: language-specific processing requirements, inconsistent naming conventions (with companies often referred to by full names rather than ticker symbols), and varying editorial standards across news sources. These characteristics make naive text-to-price prediction approaches unreliable, as they fail to account for the structured relationship between specific events and specific corporate entities.

The VN30 index is chosen as the study universe for several strategic reasons. First, VN30 constituents are large-cap stocks with high liquidity, ensuring that price data are reliable and that observed price movements reflect genuine market reactions rather than microstructure noise or illiquidity artifacts. Second, these companies receive substantially more news coverage than smaller firms, providing sufficient data for model training while maintaining a manageable scope for data collection and processing. Third, VN30 stocks are widely followed by institutional and retail investors, meaning that news impact is more likely to be systematic and interpretable compared to thinly traded stocks where idiosyncratic factors dominate. Fourth, the VN30 universe is stable enough over the study period (January 2023 to January 2026) to support consistent longitudinal analysis without excessive rebalancing effects that would complicate event-to-price attribution.

The choice of medium-term prediction horizons—specifically 15 trading days and 40 trading days—reflects a deliberate focus on capturing sustained market reactions rather than intraday or overnight volatility. Short-term reactions (1-5 days) are dominated by noise, order flow dynamics, and high-frequency trading effects that are difficult to predict from news alone. Conversely, very long horizons (>60 days) introduce substantial confounding from macroeconomic shifts, earnings cycles, and unrelated events that obscure the signal from any single news item. The 15-day horizon captures the immediate-to-short-term market digestion of news, testing whether the market reacts within roughly three weeks of an event. The 40-day horizon extends this to approximately two months, capturing both delayed reactions and secondary effects such as analyst revisions, follow-on coverage, or regulatory responses. By evaluating both horizons, the system can assess whether news signals are primarily short-lived or whether they carry sustained predictive power, which has direct implications for trading strategy design.

1.5.2 Formal Problem Definition

We now provide a formal mathematical definition of the prediction task. Let $\mathcal{U} = \{s_1, s_2, \ldots, s_N\}$ denote the universe of N stock symbols (in this case, the 30 constituents of the VN30 index). Let $\mathcal{T} = [t_{\text{start}}, t_{\text{end}}]$ denote the temporal window of interest, where $t_{\text{start}} = 2023\text{-}01\text{-}01$ and $t_{\text{end}} = 2026\text{-}01\text{-}31$ in this study.

Raw Inputs:

1. **News Articles:** A collection $\mathcal{A} = \{a_1, a_2, \ldots, a_M\}$ of news articles, where each article a_i is characterized by:
    - title_i: article title (text)
    - summary_i: article summary or lead paragraph (text)
    - body_i: full article text (text)
    - url_i: unique identifier and source URL (string)
    - publish_date_i: publication timestamp (datetime)
    - source_i: news source identifier (e.g., Cafef, Vietstock, VnExpress)
2. **Price Series:** For each symbol $s \in \mathcal{U}$ and each trading day d in the trading calendar $\mathcal{C} \subseteq \mathcal{T}$, we observe:
    - $P_{\text{open}}(s, d)$: opening price
    - $P_{\text{close}}(s, d)$: closing price
    - $P_{\text{high}}(s, d)$: intraday high
    - $P_{\text{low}}(s, d)$: intraday low

- $V(s, d)$: trading volume

3. **Trading Calendar:** A set $\mathcal{C}$ of valid trading days, accounting for weekends, public holidays, and market closures. This is essential for computing horizon returns correctly, as calendar days and trading days diverge.

4. **Corporate Actions:** A set $\mathcal{X}$ of corporate action events, where each $x_j \in \mathcal{X}$ includes:

   - $\text{symbol}_j \in \mathcal{U}$: affected stock
   - action_type_j: type of action (dividend, rights issue, stock split, etc.)
   - record_date_j: date of record for eligibility
   - ex_date_j: ex-rights date when stock trades without entitlement
   - effective_date_j: date when action takes effect

Intermediate Representations:

The raw inputs are transformed into structured intermediate representations through a series of agents:

1. **Events:** Each article a_i is processed to extract a set of structured events $E(a_i) = \{e_{i,1}, e_{i,2}, \ldots, e_{i,k_i}\}$, where each event $e_{i,j}$ is a tuple:

   $$e_{i,j} = (\text{event\_id}, \text{symbol}, \text{event\_type}, \text{sentiment}, \text{impact\_hint}, \text{evidence})$$

   Here, event_id is a deterministic hash ensuring idempotency across pipeline runs, symbol $\in \mathcal{U}$ identifies the affected stock, event_type $\in \{\text{earnings}, \text{governance}, \text{macro}, \text{legal}, \text{M\&A}, \text{rumor}, \text{corp\_action}, \text{other}\}$ categorizes the information, sentiment $\in \mathbb{R}$ captures polarity, impact_hint $\in \mathbb{R}$ estimates materiality, and evidence is a JSON structure linking back to the source article.

2. **Market Reactions:** For each event e with symbol s and publication date t_{pub}, we compute the market reaction over a horizon $h \in \{15, 40\}$ trading days:

   - $t_0(e)$: the first trading day on or after t_{pub} (i.e., $t_0 = \min\{d \in \mathcal{C} : d \geq t_{\text{pub}}\}$)
   - $t_h(e) = t_0 + h$ trading days (not calendar days)
   - $r_h(e) = \frac{P_{\text{close}}(s,t_h) - P_{\text{close}}(s,t_0)}{P_{\text{close}}(s,t_0)}$: the h-day horizon return
   - $y(e) = \mathbb{I}(r_h(e) > 0)$: binary label indicating upward movement

3. **Feature Vectors:** For each event e with a valid market reaction, a feature vector $\mathbf{x}(e) \in \mathbb{R}^D$ is constructed by combining:

- News-derived features: event type indicators, sentiment, impact hint, text statistics, keyword counts, and hashed text representations
- Price-context features: pre-event returns ($r_{5d}^{\text{pre}}, r_{20d}^{\text{pre}}$), volatility ($\sigma_{20d}^{\text{pre}}$), momentum indicators (RSI, moving average ratios), and opening gaps
- Corporate action features: proximity to record/ex-dates, counts of recent actions, and action type indicators
- Interaction terms: e.g., sentiment $\times$ corporate action proximity

Predictive Target:

The goal is to learn a function $f : \mathbb{R}^D \to [0, 1]$ that maps feature vectors to predicted probabilities of upward price movement:

$$\hat{p}(e) = f(\mathbf{x}(e)) \approx \mathbb{P}(y(e) = 1 \mid \mathbf{x}(e))$$

In practice, we train two model families:

1. **Logistic Regression (LR):** $f_{\text{LR}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$, where $\sigma(z) = 1/(1 + e^{-z})$ is the logistic sigmoid and $\mathbf{w}, b$ are learned parameters.
2. **Multi-Layer Perceptron (MLP):** $f_{\text{MLP}}(\mathbf{x}) = \sigma(\mathbf{W}_4 \cdot \text{ReLU}(\mathbf{W}_3 \cdot \text{ReLU}(\mathbf{W}_2 \cdot \text{ReLU}(\mathbf{W}_1 \mathbf{x}))))$, where $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4$ are learned weight matrices with dimensions 96-48-24-1 respectively, and $\text{ReLU}(z) = \max(0, z)$.

Relationship to Trading Decisions:

The predicted probability $\hat{p}(e)$ is used to make binary trading decisions via a threshold rule:

$$\text{Signal}(e) = \begin{cases} 1 & \text{if } \hat{p}(e) \geq \tau \\ 0 & \text{otherwise} \end{cases}$$

where τ (e.g., $\tau = 0.55$) is a tunable threshold. A signal of 1 indicates a recommendation to take a long position in symbol s at time t_0 and hold for h days, expecting positive return $r_h > 0$. The choice of threshold determines the trade-off between precision (hit rate among executed trades) and recall (coverage of profitable opportunities). By using a threshold above 0.5, the system effectively filters low-confidence signals, focusing on high-conviction predictions where the model is most certain.

1.5.3 Problem Decomposition into Sub-Problems

The overall task of predicting horizon returns from news is complex and involves multiple failure modes. We decompose it into six distinct sub-problems, each of which is addressed by a specific component of the system. This decomposition follows the principle of separation of concerns, ensuring that each stage is modular, testable, and interpretable.

Sub-problem 1: News Acquisition and Coverage Constraints

Challenge: Financial news sources publish articles asynchronously, with varying coverage patterns across symbols and time periods. Some sources maintain comprehensive XML sitemaps, while others require active crawling of category pages. News portals may experience downtime, change URL structures, or introduce rate limits, leading to gaps in coverage. Furthermore, not all news is relevant: many articles discuss macroeconomic trends, sector-level commentary, or international markets without mentioning specific VN30 symbols.

Failure Modes:

- *Incomplete coverage:* Missing news for certain symbols or time periods biases the dataset toward heavily covered stocks.
- *Duplication:* The same article may appear in multiple feeds or be republished with minor edits, inflating event counts.
- *Source reliability:* News quality varies; some sources focus on rumors or low-credibility commentary.

Approach Taken:

The system uses a two-phase crawler implemented in `src/collectors/news_sitemap_backfill.py`:

1. **Discovery Phase:** Parse XML sitemaps from Cafef, Vietstock, and VnExpress to enumerate URLs with `lastmod` timestamps. Filter URLs by category slugs (e.g., `/chung-khoan/`, `/doanh-nghiep/`) to exclude irrelevant sections.
2. **Fetch Phase:** Retrieve full article HTML for each URL, extract title, summary, and body using source-specific CSS selectors, and store raw data in the `news_raw` table with `fetch_date` metadata.

Coverage is monitored via the `coverage_monthly` table, which aggregates counts by source, symbol, and month. If counts fall below thresholds, a retry mechanism in `run_all.py` triggers gap-filling passes. This ensures that temporary downtime does not permanently corrupt the dataset.

Why This Approach Is Appropriate:

The sitemap-based approach is efficient (avoiding repeated crawls of the same content), deterministic (the `lastmod` field provides natural deduplication keys), and scalable (sitemaps are designed for bulk discovery). By storing raw HTML in PostgreSQL with metadata, the system supports reprocessing if extraction logic changes, without re-fetching from external sources.

Sub-problem 2: Noise and Redundancy in Financial News

Challenge: Raw news articles contain substantial noise: advertisements, boilerplate disclaimers, related article links, and journalistic filler. Many articles are low-information updates (e.g., "Stock X closed slightly higher today") that do not warrant event extraction. Others are duplicates with minor variations (e.g., syndicated content republished across sources).

Failure Modes:

- *Event inflation:* Extracting spurious events from low-content articles dilutes signal quality.
- *Overfitting to source style:* Model learns source-specific writing patterns rather than genuine information content.
- *Label leakage:* If an article published after market close discusses intraday price movements, using that article creates look-ahead bias.

Approach Taken:

Agent 1 (`src/agents/agent1_news_event.py`) applies a multi-stage filtering and structuring process:

1. **Concatenate Content:** Merge title, summary, and body into a unified text blob to maximize signal.
2. **Symbol Detection:** Use a comprehensive alias map (`SYMBOL_ALIASES` in `src/collectors/vn30_universe.py`) to detect mentions of VN30 companies, accounting for full names, short names, and Vietnamese-language variants. For example, "Ngân hàng Vietcombank" maps to symbol VCB.
3. **Hard Filters:** Discard articles that mention no VN30 symbols, or that contain blacklist keywords indicating irrelevant content (e.g., "advertisement", "sponsored").
4. **Deduplication:** Generate a deterministic `event_id` by hashing (symbol, publish_date, url) to ensure that reruns do not create duplicate events.

Why This Approach Is Appropriate:

Rule-based filtering is transparent and deterministic, making it easy to diagnose and adjust. The alias map is domain-specific and curated for the Vietnamese market, addressing the unique challenge of inconsistent naming conventions. By structuring events at this stage, downstream agents (Agent 2, 3, 4) operate on clean, deduplicated data, improving both training efficiency and model interpretability.

Sub-problem 3: Event Extraction from Unstructured Text

Challenge: A single news article may describe multiple types of information: an earnings announcement may also mention a governance change, or a macro commentary may discuss its implications for specific sectors. Extracting structured events requires identifying which parts of the text correspond to which event types, and assigning appropriate metadata (sentiment, impact, evidence trail).

Failure Modes:

- *Misclassification:* Keyword-based rules may misfire (e.g., the word "profit" in a negative context is classified as earnings but with wrong sentiment).
- *Ambiguity:* Events that span multiple types (e.g., a dividend announcement within an earnings report) may be forced into a single category.
- *Missing events:* Novel or rare event types not covered by keyword lists are dropped.

Approach Taken:

Agent 1 uses a hybrid approach:

1. **Event Type Classification:** Scan the concatenated text against keyword dictionaries in `EVENT_PATTERNS` (e.g., earnings: ["profit", "revenue", "li nhun"], governance: ["CEO", "board", "b nhim"]). If no match is found, assign `event_type="other"`.
2. **Sentiment Extraction:** If the source provides a sentiment field, use it directly. Otherwise, apply a lightweight heuristic lexicon that counts positive words (e.g., "increase", "growth", "tăng") vs. negative words (e.g., "loss", "decline", "gim") and computes a net score.
3. **Impact Estimation:** Assign a base impact weight by event type (e.g., earnings events receive higher weight than rumors), then boost the score if the text contains numeric mentions (percentages, monetary amounts), reflecting that quantified results are typically more material.
4. **Evidence Linking:** Store the original article ID and URL in a JSON `evidence` field, enabling traceback for verification or manual review.

Why This Approach Is Appropriate:

This design balances coverage and precision. Keyword-based classification is simple and fast, suitable for processing 148,000+ articles. The sentiment heuristic avoids dependence on external NLP models (which may not be well-calibrated for Vietnamese financial text), though it accepts lower accuracy. The impact hint is a rough proxy for materiality, sufficient for initial prioritization; more sophisticated approaches (e.g., using event-study abnormal returns) could be explored in future work but would require labeled training data. The evidence linking ensures that every event can be audited, which is critical for diagnosing errors and refining extraction rules.

Sub-problem 4: Mapping Events to Trading Calendar and Reaction Windows

Challenge: News articles are published at arbitrary times—before market open, during trading hours, or after market close. To measure market reaction, we must align each event to a well-defined reference point t_0 in the trading calendar, then compute returns over a fixed horizon h trading days. This requires handling weekends, holidays, market closures, and cases where price data are missing.

Failure Modes:

- *Look-ahead bias:* If t_0 is set to the publication date and the article is published after market close, using the closing price on that day would introduce leakage.
- *Insufficient data:* If the price series for a symbol has gaps or ends before $t_0 + h$, the horizon return cannot be computed, requiring the event to be dropped.
- *Calendar confusion:* Using calendar days instead of trading days inflates horizons and misaligns returns.

Approach Taken:

Agent 2 (`src/agents/agent2_market_react.py`) implements a rigorous alignment protocol:

1. **Determine t_0:** For each event with `publish_date`, find the first trading day on or after `publish_date`. This is computed by querying the `prices` table for the minimum date $d \geq$ `publish_date` where the symbol has a valid close price.
2. **Fetch Price Series:** Retrieve the ordered sequence of closing prices $[P_0, P_1, \ldots, P_k]$ starting from t_0, where each index corresponds to a trading day (not calendar day).
3. **Compute Horizon Return:** If the series has at least $h + 1$ prices, compute:
$$r_h = \frac{P_h - P_0}{P_0}$$
Otherwise, mark the event as `insufficient_horizon` and drop it from training.
4. **Assign Binary Label:**
$$y = \begin{cases} 1 & \text{if } r_h > 0 \\ 0 & \text{if } r_h \leq 0 \end{cases}$$

5. **Compute Risk Context:** Calculate 5-day pre-event volatility (σ_{5d}^{pre}) and maximum drawdown over the horizon ($\text{DD}_h = \max_{i \in [0,h]} \frac{P_0 - P_i}{P_0}$) to provide additional context for interpreting return magnitudes.

Why This Approach Is Appropriate:

This design ensures that returns are tradeable: an investor receiving the news at publication time could realistically enter a position at t_0 and exit at t_h. By using the first trading day on or after publication, we avoid look-ahead bias while still capturing market reactions that occur on the publication day if the market is open. The explicit handling of missing data (via drop reasons) makes coverage issues transparent, allowing targeted improvements in price collection. The addition of volatility and drawdown statistics enriches the label representation, enabling future extensions where models predict risk-adjusted returns or drawdown constraints.

Sub-problem 5: Feature Construction Combining News and Price Context

Challenge: Effective prediction requires integrating heterogeneous information sources: text signals from news, price momentum and volatility from historical trading, and corporate action timing from structured databases. Each source has different scales, missing data patterns, and relevance windows, making naive concatenation ineffective.

Failure Modes:

- *Feature leakage:* Including post-event price data or returns computed using t_h in the feature vector creates look-ahead bias.
- *Dimensionality explosion:* Naively encoding text as TF-IDF or word embeddings can produce high-dimensional, sparse feature spaces that overfit.
- *Missing value propagation:* Corporate actions are sparse; if missingness is not handled, many samples are dropped or assigned arbitrary fill values.

Approach Taken:

Agent 3 (`src/agents/agent3_features_joint.py`) constructs a unified feature vector through the following pipeline:

1. **News Features:**
    - One-hot indicators for `event_type` (8 categories)
    - Sentiment score (scalar)
    - Impact hint (scalar)
    - Text length statistics (title length, body length)

- Keyword counts for domain-specific terms (e.g., "profit", "dividend", "investigation")
- Hashed text features using `HashingVectorizer(n_features=128)` applied to title + summary, providing a fixed-size, memory-efficient representation of lexical content

2. **Price-Context Features (Pre-Event Only):**
   - Short-term return: $r_{5d}^{\text{pre}} = \frac{P_{t_0-5} - P_{t_0-1}}{P_{t_0-1}}$
   - Medium-term return: $r_{20d}^{\text{pre}} = \frac{P_{t_0-20} - P_{t_0-1}}{P_{t_0-1}}$
   - Volatility: $\sigma_{20d}^{\text{pre}} = \text{std}(\{r_{t-20}, \ldots, r_{t-1}\})$
   - RSI(14): Relative Strength Index over 14 trading days prior to t_0
   - Moving average ratio: MA5/MA20 $-$ 1, where MA5 and MA20 are simple moving averages
   - Opening gap: $(P_{\text{open}}(t_0) - P_{\text{close}}(t_0 - 1))/P_{\text{close}}(t_0 - 1)$
   - Volume Z-score: $(V_{t_0} - \mu_{V,20d})/\sigma_{V,20d}$

   All price features use only data prior to t_0, strictly avoiding leakage.

3. **Corporate Action Features:**
   - Days since most recent record date
   - Days until next record date
   - Binary flags for record/ex-date proximity within 7, 14, and 30-day windows
   - Count of corporate actions in the last 180 days
   - One-hot indicators for most recent action type (dividend, rights issue, etc.)

4. **Interaction Terms:**
   - sentiment $\times$ within_window_ex_7d: captures whether positive/negative news coincides with ex-dates
   - r_{5d}^{pre} $\times$ within_window_record_14d: captures momentum near corporate action events

Why This Approach Is Appropriate:

The feature design follows the principle of domain-informed engineering: each feature group reflects a specific hypothesis about what drives price movements (e.g., momentum persists, corporate actions create predictable patterns, sentiment matters). The use of hashing vectorizers avoids vocabulary growth and ensures consistent feature dimensionality across training runs. The strict separation of pre-event and post-event data prevents leakage. The interaction terms are simple (pairwise products) but theoretically motivated, allowing linear models to capture basic non-linear effects. Missing corporate action features are left as `None` at this stage and sanitized to zero in Agent 4, which is a conservative choice (missing = no signal) but avoids introducing artificial patterns.

Sub-problem 6: Model Training, Prediction, and Economic Evaluation

Challenge: The final sub-problem is to learn a mapping from feature vectors to binary labels (or probabilities) that generalizes to future data. This requires careful handling of data splits, class imbalance, regularization, and evaluation metrics. Furthermore, prediction quality must be assessed not only by classification accuracy but also by economic utility (i.e., whether predictions translate into profitable trades).

Failure Modes:

- *Temporal leakage:* Using random train/test splits allows the model to "see the future" during training.
- *Class imbalance:* If one label dominates (e.g., 70% up, 30% down), a naive model can achieve high accuracy by always predicting the majority class.
- *Overfitting:* High-capacity models (e.g., deep neural networks) can memorize training noise, especially when feature dimensions are high relative to sample size.
- *Evaluation mismatch:* Optimizing for AUC or accuracy does not guarantee profitable trading if the probability threshold is not tuned for financial objectives.

Approach Taken:

Agent 4 (implemented in `src/agents/agent4_train_predict_joint.py` for Logistic Regression and `src/agents/agent4_train_predict_nn.py` for MLP) addresses these challenges through a rigorous training protocol:

1. **Time-Based Split:** Sort all samples by t_0 in ascending order. Use the earliest 80% for training and the most recent 20% for validation. This mimics real-world deployment where models are trained on historical data and evaluated on future data.
2. **Feature Vectorization:** Apply `DictVectorizer` to convert feature dictionaries to sparse matrices. Missing values (None) are sanitized to 0.0 to ensure numerical stability.

3. **Class Weight Balancing:** For Logistic Regression, set `class_weight='balanced'` to automatically adjust loss weights inversely proportional to class frequencies. This mitigates the impact of label imbalance.

4. **Logistic Regression Training:**

$$\min_{\mathbf{w},b} \sum_{i=1}^{N_{\text{train}}} w_i \left[-y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i + b) - (1 - y_i) \log(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i + b))\right] + \lambda \|\mathbf{w}\|_2^2$$

   where w_i are class weights and λ is an L2 regularization coefficient. The model is trained using L-BFGS with a maximum of `LR_MAX_ITER` iterations.

5. **MLP Training:** The 96-48-24-1 architecture is trained using stochastic gradient descent with Adam optimizer, learning rate 10^{-3}, weight decay 10^{-4}, and dropout rate 0.3. Features are standardized using `StandardScaler`. Early stopping is applied based on validation loss to prevent overfitting.

6. **Prediction:** Both models output probabilities $\hat{p}_i \in [0, 1]$ for each validation sample. These probabilities are used for both classification metrics and backtesting.

7. **Evaluation Metrics:**

   - **Accuracy:** $\frac{1}{N_{\text{val}}} \sum_{i=1}^{N_{\text{val}}} \mathbb{I}(\hat{y}_i = y_i)$, where $\hat{y}_i = \mathbb{I}(\hat{p}_i \geq 0.5)$
   - **Precision:** $\frac{\text{TP}}{\text{TP+FP}}$
   - **Recall:** $\frac{\text{TP}}{\text{TP+FN}}$
   - **AUC:** Area under the ROC curve, measuring the model's ability to rank positive samples above negative samples

8. **Backtesting:** Apply a probability threshold $\tau = 0.55$ to filter signals. For each sample with $\hat{p}_i \geq \tau$, include the realized return $r_{h,i}$ in the backtest portfolio. Compute:

   - n_{trades}: number of signals executed
   - mean_ret_h $= \frac{1}{n_{\text{trades}}} \sum_{i:\hat{p}_i \geq \tau} r_{h,i}$
   - hit_rate $= \frac{1}{n_{\text{trades}}} \sum_{i:\hat{p}_i \geq \tau} \mathbb{I}(r_{h,i} > 0)$

Why This Approach Is Appropriate:

The time-based split is essential for financial prediction, as it prevents overfitting to future information. Class weighting addresses imbalance without discarding data. The choice of two model families (linear and nonlinear) allows us to assess whether the problem benefits from increased capacity or whether a simple interpretable baseline suffices. The dual evaluation framework (classification metrics + backtesting) ensures that we understand both predictive power and economic utility, recognizing that high AUC does not always translate to profitable trading. The fixed threshold ($\tau = 0.55$) is a simplification, but it provides a consistent basis for comparison; future work could explore threshold optimization or dynamic thresholding based on market regimes.

1.5.4 Definition of Events and Rationale for Event Selection

This subsection provides a detailed conceptual and operational definition of "events" in the context of this system, explains why an event-based abstraction is superior to article-level or sentence-level prediction, and justifies the specific design choices made in event extraction.

What Constitutes an Event

In this system, an **event** is defined as a structured representation of a discrete piece of information that links a specific news article to a specific VN30 stock symbol, characterized by:

- **Entity:** Which stock (symbol $s \in \mathcal{U}$) is affected?
- **Type:** What category of information is conveyed (earnings, governance, legal, M&A, etc.)?
- **Sentiment:** What is the directional tone (positive, negative, neutral)?
- **Materiality:** How impactful is the information likely to be (captured by `impact_hint`)?
- **Provenance:** Where did the information come from (URL, source, publish date)?

Formally, events are tuples in the `news_events` table, each with a deterministic `event_id` computed as:

$$\text{event\_id} = \text{hash}(\text{symbol}, \text{publish\_date}, \text{url})$$

This ensures that reprocessing the same article generates the same event, supporting idempotency.

Why a Single Article May Generate Multiple Events

A financial news article often discusses multiple companies or multiple types of information about the same company. For example, an article titled "Banking Sector Earnings Review" might mention VCB's quarterly profit, TCB's dividend policy, and ACB's governance changes. If we treated the entire article as a single unit, we would lose the specificity needed for stock-level prediction. Conversely, extracting events at the sentence level would fragment the signal and introduce excessive noise from transitional or contextual sentences that do not convey actionable information.

The event-based abstraction strikes a balance: each event corresponds to a (symbol, event_type) pair detected in the article. If an article mentions three symbols, it generates up to three events. If an article discusses both earnings and governance for the same symbol, it may generate two events with different types. This granularity ensures that:

1. Each event has a clear predictive target (the price of the associated symbol).
2. Event types can be compared and weighted differently (e.g., earnings events may be more predictive than rumors).
3. The system can filter or prioritize events based on type, sentiment, or impact without discarding entire articles.

Why Event-Based Abstraction Is Used Instead of Article-Level Prediction

There are several strong theoretical and practical reasons for using events rather than articles as the unit of analysis:

1. **Multi-Entity Coverage:** Articles frequently mention multiple companies. An article-level approach would require either (a) assigning a single "primary" symbol (which loses information about other mentioned companies), or (b) replicating the entire article across all symbols (which inflates the dataset with redundant text and creates correlated noise).
2. **Type-Specific Signal Strength:** Different event types have different predictive power. Earnings announcements with quantified results are typically more informative than vague rumors. By extracting typed events, the model can learn type-specific weights (in LR) or representations (in MLP), improving discrimination.
3. **Irrelevance Filtering:** Many articles are pure commentary or macro-level discussion with no stock-specific signal. Event extraction acts as a first-stage filter: articles that mention no VN30 symbols are immediately discarded, and articles that mention symbols only in passing (e.g., "the VN-Index, which includes VCB, rose 0.2%") can be flagged as low-impact through the `impact_hint` mechanism.
4. **Interpretability:** Events are human-interpretable units. An analyst can review the top-weighted event types from a trained model and understand which

information sources drive predictions. Article-level predictions obscure this interpretation by conflating multiple signals.

5. **Scalability:** The event extraction step (Agent 1) is computationally cheap (rule-based keyword matching). It reduces the downstream dataset size by 5-10x (148,182 articles $\rightarrow$ 67,728 events in this study), making feature engineering and model training more efficient.

Event Relevance to Price Cycles and Corporate Action Milestones

Financial markets operate on cycles: quarterly earnings, annual shareholder meetings, dividend record dates, and regulatory filing deadlines create predictable windows where information is expected. Events extracted from news are most useful when they align with or anticipate these cycles. For example:

- An earnings event published shortly before an official earnings release provides an early signal (analyst preview, leaked information).
- A governance event (CEO appointment) typically precedes or follows board meetings, which are scheduled events.
- Corporate action events (dividend announcements) are tied to record and ex-dates, which have mechanical price effects.

By structuring events and aligning them to corporate action dates (via Agent 3's corporate action features), the system can capture these regularities. For instance, if a news event discusses a dividend on day t and the ex-date is $t + 10$, the model can learn that such proximity increases the probability of short-term positive returns.

Filtering Irrelevant Articles: Category Slugs and Hard Filters

The news crawler applies preliminary filtering at the URL level using category slugs. Vietnamese financial news portals organize content into hierarchical categories (e.g., `/chung-khoan/`, `/doanh-nghiep/`, `/bat-dong-san/`). Only URLs containing finance-related slugs are fetched. This reduces the initial fetch volume by approximately 50%, excluding sports, entertainment, and international news sections.

After fetching, Agent 1 applies additional hard filters:

- **Symbol Requirement:** Articles that do not mention any VN30 symbol (after alias matching) are discarded.
- **Blacklist Keywords:** Articles containing terms like "qung cáo" (advertisement) or "sponsored content" are excluded, as they typically lack editorial information.

- **Length Threshold:** Very short articles (<100 characters) are often headlines or teasers without substantive content, and are dropped.

These filters collectively reduce noise while preserving relevant events. The drop statistics (148,182 articles → 67,728 events, an extraction rate of 45.7%) indicate that the filters are selective but not overly aggressive.

Reducing Noise: Multi-Ticker Handling and Allowlist/Denylist

A common source of noise is articles that mention many symbols in a generic context (e.g., market summary articles listing all VN30 constituents with their daily price changes). Such articles generate many events, but each event carries minimal symbol-specific information. To mitigate this:

- **Impact Hint Adjustment:** The `impact_hint` computation reduces the score for symbols mentioned only in passing (detected via occurrence count relative to article length).
- **Event Type Filtering:** Events with `event_type="other"` and low sentiment are deprioritized during model training (via lower weights or explicit filtering in Agent 4).

Future enhancements could include an explicit denylist of article URLs or sources known to produce low-quality signals, and an allowlist of high-credibility sources (e.g., official corporate announcements on company investor relations pages).

Code Excerpt: Event Extraction Logic

To illustrate the event extraction process concretely, consider the following simplified excerpt from Agent 1:

Listing 1.1: Simplified event extraction logic from Agent 1

```python
def extract_events(article: dict, symbol_aliases: dict) -> list[dict]:
    """
    Extract structured events from a raw news article.

    Args:
        article: Dict with keys 'title', 'summary', 'text', 'url', 'publish_date'
        symbol_aliases: Dict mapping symbol -> list of name variants

    Returns:
        List of event dicts, each with symbol, event_type, sentiment, impact_hint
    """
    # Step 1: Concatenate all text fields to maximize signal coverage
    full_text = f"{article['title']} {article['summary']} {article['text']}"
    lowered_text = full_text.lower()

    # Step 2: Detect mentioned VN30 symbols using alias matching
    detected_symbols = set()
    for symbol, aliases in symbol_aliases.items():
        for alias in aliases:
            if alias.lower() in lowered_text:
                detected_symbols.add(symbol)
                break # One match per symbol is sufficient

    # Step 3: If no symbols detected, return empty list (article is irrelevant)
    if not detected_symbols:
        return []

    # Step 4: Classify event type using keyword patterns
    event_type = classify_event_type(lowered_text)
    # classify_event_type scans for keywords from EVENT_PATTERNS dict
    # Returns 'earnings', 'governance', 'legal', 'mna', 'macro',
    # 'rumor', 'corp_action', or 'other' if no match

    # Step 5: Extract or estimate sentiment
    # If source provides sentiment field, use it; else apply heuristic
    sentiment = article.get('sentiment', estimate_sentiment(lowered_text))
    # estimate_sentiment counts pos/neg words, returns float in [-1, 1]

    # Step 6: Compute impact hint based on type and numeric mentions
    base_impact = EVENT_TYPE_WEIGHTS.get(event_type, 0.5)
    has_numbers = bool(re.search(r'\d+[.,]?\d*\s*%|\d+[.,]?\d*\s*(t|triu|nghn)',
        lowered_text))
    impact_hint = base_impact * (1.5 if has_numbers else 1.0)

    # Step 7: Generate one event per detected symbol
    events = []
    for symbol in detected_symbols:
        event_id = hashlib.sha256(
            f"{symbol}|{article['publish_date']}|{article['url']}".encode()
        ).hexdigest()[:16]

        events.append({
            'event_id': event_id,
            'symbol': symbol,
            'event_type': event_type,
            'sentiment': sentiment,
            'impact_hint': impact_hint,
            'publish_date': article['publish_date'],
            'evidence_json': json.dumps({
                'url': article['url'],
60                  'title': article['title']
61              })
62          })
63
64      return events
```

Line-by-Line Explanation:

- **Lines 13-14:** Concatenate title, summary, and body to ensure no information is missed. Convert to lowercase for case-insensitive matching.
- **Lines 17-22:** Iterate over all VN30 symbols and their aliases (e.g., VCB → ["Vietcombank", "BIDV", "Ngân hàng TMCP Ngoi thng Vit Nam"]). If any alias appears in the text, add the symbol to the detected set.
- **Lines 25-27:** Early exit if no symbols are detected, saving computation on irrelevant articles.
- **Line 30:** Call a helper function that scans the text against keyword dictionaries (`EVENT_PATTERNS`) to assign a type. This is deterministic and fast.
- **Line 36:** Retrieve sentiment from the source if available (some sources include a pre-computed sentiment score). If missing, call `estimate_sentiment`, which counts occurrences of positive and negative words from a predefined lexicon.
- **Lines 39-41:** Assign a base impact weight depending on event type (e.g., earnings = 1.0, rumor = 0.3). Boost by 50% if the text contains numeric mentions (percentages, monetary amounts), as these indicate quantified information.
- **Lines 44-58:** For each detected symbol, generate a unique event ID by hashing the (symbol, publish_date, url) tuple. This ensures idempotency: reprocessing the same article with the same parameters produces the same event ID, preventing duplicates.
- **Lines 47-57:** Construct the event dictionary with all required fields. The `evidence_json` field stores a backreference to the source article, enabling later verification or manual review.

This excerpt demonstrates the operational definition of event extraction: it is a deterministic, rule-based transformation that converts noisy, multi-entity news text into structured, symbol-specific events with metadata suitable for downstream modeling. The design prioritizes transparency, efficiency, and reproducibility, which are critical for a research-grade pipeline.

Chapter 2

Related Work and Theoretical Foundations

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.

2.5 Theoretical Foundations and Methodological Choices

This section establishes the theoretical underpinnings of the methodologies employed in the pipeline, connecting abstract concepts from machine learning, financial econometrics, and software engineering to the concrete design decisions made in each agent. The goal is to justify why specific techniques are appropriate for news-driven stock prediction and to articulate the strengths and limitations of each approach.

2.5.1 Agent-Based System Design and Separation of Concerns

The architecture of this pipeline is organized as a chain of four specialized agents, each responsible for a distinct transformation stage. This design follows the principle of **separation of concerns**, a foundational concept in software engineering that advocates for modular systems where each component has a well-defined, narrow responsibility. In the context of machine learning pipelines, separation of concerns offers several theoretical and practical advantages.

Theoretical Motivation for Modularity

From a computational perspective, complex data pipelines can be viewed as compositions of transformations:

$$\mathcal{D}_{\text{output}} = f_4 \circ f_3 \circ f_2 \circ f_1(\mathcal{D}_{\text{input}})$$

where each f_i represents an agent's transformation function, and $\mathcal{D}$ represents a dataset or intermediate representation. If these functions are tightly coupled (i.e., f_i depends on internal details of f_j for $j \neq i$), then changes to one component propagate unpredictably through the system, making debugging and iteration difficult. Conversely, if each f_i operates on a well-defined interface (input schema and output schema), the system exhibits **composability**: agents can be tested, validated, and improved independently.

In statistical learning theory, composability also relates to the concept of **staged estimation**. Many modern machine learning pipelines (e.g., those involving feature extraction followed by supervised learning) can be formalized as two-stage or multi-stage estimators, where each stage solves a sub-problem and the final estimator combines the results. By decoupling stages, we can apply different optimization techniques, regularization strategies, or data representations at each stage without interfering with others. For example, Agent 1 (event extraction) uses rule-based heuristics, while Agent 4 (model training) uses gradient-based optimization; these paradigms are incompatible in a monolithic system but coexist naturally in a modular one.

Agent Responsibilities and Interfaces

The four agents in this pipeline are designed with explicit interfaces and responsibilities:

1. **Agent 1 (Event Extraction):**
    - *Input:* `news_raw` table (unstructured text, metadata)
    - *Output:* `news_events` table (structured events with symbol, type, sentiment, impact)

- *Responsibility:* Convert unstructured news into structured, symbol-specific events
- *Independence:* Does not depend on price data or market reactions; purely text-based

2. **Agent 2 (Market Reaction Labeling):**
   - *Input:* `news_events` + `prices` table
   - *Output:* `market_reactions` table (event ID, returns, labels, risk metrics)
   - *Responsibility:* Align events to trading calendar, compute horizon returns, assign binary labels
   - *Independence:* Does not perform feature engineering or modeling; purely labeling logic

3. **Agent 3 (Feature Engineering):**
   - *Input:* `news_events` + `market_reactions` + `prices` + `corp_actions`
   - *Output:* `features_joint` table (event ID, feature dictionary)
   - *Responsibility:* Construct multi-source feature vectors combining news, price, and corporate action signals
   - *Independence:* Does not train models or make predictions; purely feature construction

4. **Agent 4 (Model Training and Prediction):**
   - *Input:* `features_joint` + `market_reactions` (joined)
   - *Output:* Trained models in `artifacts/`, metadata in `models`, predictions in `predictions`
   - *Responsibility:* Train classifiers, evaluate performance, generate predictions
   - *Independence:* Does not perform feature extraction; operates on pre-computed feature vectors

This decomposition ensures that if, for example, the price data source changes (e.g., switching from FireAnt to a different API), only Agent 2 needs to be updated. Similarly, if we wish to experiment with different feature representations (e.g., replacing HashingVectorizer with transformer embeddings), only Agent 3 needs modification. This reduces the risk of introducing bugs and accelerates experimentation.

Why Agent-Based Design Is Suitable for Financial Pipelines

Financial prediction pipelines face unique challenges that make modularity especially valuable:

- **Data Heterogeneity:** Financial systems integrate multiple data types (text, time-series, structured databases) from disparate sources. Each data type has different reliability, latency, and update frequencies. Separating data processing into specialized agents allows each agent to handle source-specific idiosyncrasies (e.g., Agent 1 handles news parsing quirks, Agent 2 handles trading calendar alignment).
- **Regulatory and Audit Requirements:** Financial models are often subject to regulatory scrutiny. Modular systems produce intermediate outputs at each stage (e.g., `news_events`, `market_reactions`) that can be audited independently. Regulators or risk managers can verify that event extraction is unbiased, that labeling logic is correct, and that features do not introduce leakage, without needing to understand the entire end-to-end system.
- **Non-Stationarity and Retraining:** Financial markets are non-stationary: relationships between features and returns change over time due to regime shifts, regulatory changes, and macroeconomic trends. In a modular pipeline, models can be retrained (Agent 4) without re-extracting events (Agent 1) or re-labeling reactions (Agent 2), saving computation and ensuring consistency.
- **Error Diagnosis and Coverage Monitoring:** Each agent produces its own report (e.g., Agent 1 reports symbol coverage, Agent 2 reports drop reasons). This allows pinpointing where data quality issues arise. If Agent 4 performance degrades, we can trace backward: Are features missing? Are labels noisy? Is event extraction failing? Monolithic systems obscure this causal chain.

2.5.2 Event-Based Learning in Finance

Event-driven strategies are a well-established paradigm in quantitative finance, rooted in the theory that discrete, identifiable events (earnings announcements, M&A, regulatory filings) carry information that is not immediately reflected in prices. This subsection explains the theoretical basis for event-based learning and connects it to the pipeline design.

Information Content and Market Microstructure

In classical efficient market theory (Fama, 1970), asset prices reflect all available information, and new information is incorporated instantaneously. However, empirical research in market microstructure has documented systematic delays and patterns in how news is processed:

- **Price Discovery Delay:** Not all market participants observe or interpret news simultaneously. Retail investors may react hours or days after institutional investors.
- **Attention Constraints:** Investors have limited attention and cannot process all news in real time. Salient or unexpected events receive faster attention than routine updates.
- **Liquidity Effects:** In less liquid markets (including many emerging markets), large news-driven trades can take multiple sessions to execute, prolonging price adjustment.

For the VN30 universe, which includes the largest and most liquid Vietnamese stocks, we expect relatively efficient pricing compared to smaller stocks. Nonetheless, the 15- and 40-day prediction horizons are designed to capture both immediate reactions and delayed effects. The hypothesis is that news events create an informational advantage that decays over time: the signal is strongest in the first few days but may persist for weeks as the market digests implications.

Event Windows and Reaction Measurement

An **event study** is a standard econometric method for measuring the impact of events on stock returns. The typical event study framework involves:

1. Defining an event date t_{event} (e.g., earnings announcement date).
2. Defining an estimation window (e.g., $[-120, -20]$ days before the event) to model "normal" returns.
3. Defining an event window (e.g., $[-1, +5]$ days around the event) to measure abnormal returns.
4. Computing cumulative abnormal returns (CAR) as the sum of differences between actual and expected returns.

The pipeline's approach in Agent 2 is a simplified event study: instead of modeling expected returns, we directly measure the realized return from t_0 to t_h and assign a binary label based on the sign. This is a pragmatic choice motivated by:

- **Simplicity:** Binary labels are easier to model than continuous abnormal returns, and they directly correspond to a trading decision (long or no position).

- **Robustness:** Estimating "expected" returns requires a benchmark model (e.g., CAPM, Fama-French), which introduces additional assumptions and potential misspecification. Raw returns avoid this.
- **Interpretability:** A label of 1 means "the stock went up over the horizon", which is immediately understandable to traders and analysts.

Advantages of Event-Based Abstraction Over Naive Text-to-Price Prediction

Naive approaches to news-driven prediction often train models to map raw text (or bag-of-words features) directly to price movements. This has several drawbacks:

- **Symbol Ambiguity:** A single article may mention multiple companies, but the target label is specific to one symbol. Naive models cannot disambiguate which part of the text relates to which symbol.
- **Type Conflation:** Different event types (earnings vs. rumors) have different signal strengths, but bag-of-words models treat all text equally.
- **Temporal Misalignment:** Articles are published at irregular times, but labels are defined relative to trading days. Naive models that ignore the trading calendar introduce noise.

By extracting events as structured tuples (symbol, type, sentiment, publish date) and aligning them to the trading calendar in Agent 2, the pipeline avoids these pitfalls. Events serve as a **semantic bridge** between unstructured text and structured time-series, enabling more precise modeling.

2.5.3 Logistic Regression: Theory and Justification

Logistic Regression (LR) is the baseline model used in Agent 4. Despite its simplicity, LR is widely used in financial prediction due to its interpretability, calibration properties, and computational efficiency.

Mathematical Formulation

Let $\mathbf{x} \in \mathbb{R}^D$ denote a feature vector and $y \in \{0, 1\}$ denote a binary label. Logistic Regression models the conditional probability of the positive class as:

$$p(y = 1 \mid \mathbf{x}; \mathbf{w}, b) = \sigma(\mathbf{w}^\top \mathbf{x} + b) = \frac{1}{1 + \exp(-(\mathbf{w}^\top \mathbf{x} + b))}$$

where $\sigma(z)$ is the logistic sigmoid function, $\mathbf{w} \in \mathbb{R}^D$ is a weight vector, and $b \in \mathbb{R}$ is a bias term. The sigmoid function maps the linear combination $z = \mathbf{w}^\top \mathbf{x} + b$ (called the **log-odds** or **logit**) to a probability in $[0, 1]$.

The log-odds can be interpreted as:

$$\log \frac{p(y = 1 \mid \mathbf{x})}{p(y = 0 \mid \mathbf{x})} = \mathbf{w}^\top \mathbf{x} + b$$

This shows that Logistic Regression assumes a **linear decision boundary** in the feature space: the log-odds is a linear function of the features.

Training Objective: Cross-Entropy Loss

LR is trained by minimizing the binary cross-entropy loss (also called log-loss or negative log-likelihood):

$$\mathcal{L}(\mathbf{w}, b) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

where $p_i = \sigma(\mathbf{w}^\top \mathbf{x}_i + b)$ is the predicted probability for sample i. This loss is convex in $\mathbf{w}$ and b, which guarantees that gradient-based optimization (e.g., L-BFGS, gradient descent) converges to a global minimum.

To prevent overfitting, L2 regularization is typically added:

$$\mathcal{L}_{\text{reg}}(\mathbf{w}, b) = \mathcal{L}(\mathbf{w}, b) + \lambda \|\mathbf{w}\|_2^2$$

where λ is a regularization strength parameter. The regularization term penalizes large weights, encouraging the model to prefer simpler, more generalizable solutions.

Class Imbalance Handling

In this pipeline, the label distribution varies by horizon. For example, the 40-day horizon has approximately 54% up and 46% down labels, while the 15-day horizon is more imbalanced (70% up, 30% down). Naive training on imbalanced data leads to models that are biased toward the majority class.

To mitigate this, the pipeline uses `class_weight='balanced'` in scikit-learn's LogisticRegression, which sets:

$$w_{\text{class}} = \frac{N}{2 \times N_{\text{class}}}$$

where N is the total number of samples and N_{class} is the number of samples in each class. This reweights the loss function:

$$\mathcal{L}_{\text{balanced}}(\mathbf{w}, b) = -\frac{1}{N}\sum_{i=1}^{N} w_{y_i}\left[y_i \log p_i + (1-y_i)\log(1-p_i)\right]$$

where w_{y_i} is the class weight for label y_i. This effectively upweights the minority class, forcing the model to pay more attention to rare labels.

Interpretation of Coefficients

One of the key advantages of LR is interpretability. The weight w_j for feature j can be interpreted as the change in log-odds per unit change in x_j, holding all other features constant:

$$\frac{\partial}{\partial x_j}\log\frac{p(y=1 \mid \mathbf{x})}{p(y=0 \mid \mathbf{x})} = w_j$$

Positive weights indicate features that increase the probability of upward movement, while negative weights indicate features that decrease it. For example:

- If $w_{\text{sentiment}} > 0$, higher sentiment scores are associated with higher probability of upward returns.
- If $w_{\text{event\_legal}} < 0$, legal events (lawsuits, investigations) are associated with lower probability of upward returns.

This interpretability is valuable for validation: domain experts can review the learned weights and assess whether they align with financial intuition.

Why LR Is Used as a Baseline

LR serves as a **strong baseline** for several reasons:

1. **Efficiency:** Training is fast even on large datasets (tens of thousands of samples, hundreds of features).
2. **Stability:** LR is less prone to overfitting than high-capacity models, especially when regularized.
3. **Calibration:** Logistic Regression produces well-calibrated probabilities (i.e., predicted probabilities closely match empirical frequencies), which is important for backtesting with probability thresholds.
4. **Simplicity:** LR has few hyperparameters (mainly regularization strength λ), reducing tuning effort.

Theoretical Limitations of LR in Non-Linear Settings

Despite its strengths, LR has a fundamental limitation: it assumes a linear relationship between features and log-odds. If the true relationship is non-linear or involves interactions between features, LR will underfit. For example:

- If sentiment is only predictive when combined with high volatility (an interaction effect), LR cannot capture this unless an explicit interaction term sentiment $\times$ volatility is added to the feature set.
- If the relationship between momentum and returns is non-monotonic (e.g., very high or very low momentum both signal reversals), LR cannot model this without polynomial features.

In this pipeline, Agent 3 includes a small number of hand-crafted interaction terms (e.g., sentiment $\times$ corporate action proximity), which partially address this limitation. However, for more complex non-linearities, a more flexible model is needed, motivating the use of the MLP.

2.5.4 Multi-Layer Perceptron (MLP): Theory and Justification

The Multi-Layer Perceptron (MLP), also known as a feedforward neural network, is a non-linear classifier used in Agent 4 to test whether the prediction task benefits from increased model capacity.

Architecture and Forward Pass

The MLP used in this pipeline has the following architecture:

- **Input layer:** D features (typically 150-250 after vectorization)
- **Hidden layer 1:** 96 neurons with ReLU activation
- **Hidden layer 2:** 48 neurons with ReLU activation
- **Hidden layer 3:** 24 neurons with ReLU activation
- **Output layer:** 1 neuron with sigmoid activation

Mathematically, the forward pass is:

$$\mathbf{h}^{(1)} = \mathrm{ReLU}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \tag{2.1}$$
$$\mathbf{h}^{(2)} = \mathrm{ReLU}(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \tag{2.2}$$
$$\mathbf{h}^{(3)} = \mathrm{ReLU}(\mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)}) \tag{2.3}$$
$$\hat{p} = \sigma(\mathbf{W}^{(4)}\mathbf{h}^{(3)} + \mathbf{b}^{(4)}) \tag{2.4}$$

where $\text{ReLU}(z) = \max(0, z)$ is the Rectified Linear Unit activation, $\mathbf{W}^{(i)}$ are weight matrices, and $\mathbf{b}^{(i)}$ are bias vectors.

Non-Linearity and Universal Approximation

The key property of MLPs is their ability to model non-linear functions. The ReLU activation introduces non-linearity: without it, stacking multiple linear layers would collapse to a single linear transformation. With ReLU activations, an MLP can approximate any continuous function on a compact domain, given sufficient neurons (universal approximation theorem).

In the context of financial prediction, this means the MLP can learn:

- **Feature interactions:** For example, the effect of sentiment may depend on volatility, or the predictive power of momentum may vary by event type. The hidden layers can learn these interactions automatically.
- **Non-monotonic relationships:** If high and low values of a feature both signal negative returns (a U-shaped relationship), the MLP can model this, whereas LR cannot without polynomial features.

Regularization via Dropout

The MLP uses **dropout** as a regularization technique. Dropout randomly sets a fraction of neurons to zero during each training iteration, preventing co-adaptation and reducing overfitting. Formally, during training, each hidden neuron $h_j^{(i)}$ is set to zero with probability p (typically $p = 0.3$ in this pipeline):

$$\tilde{h}_j^{(i)} = \begin{cases} 0 & \text{with probability } p \\ \frac{h_j^{(i)}}{1-p} & \text{with probability } 1-p \end{cases}$$

The scaling factor $\frac{1}{1-p}$ ensures that the expected value of $\tilde{h}_j^{(i)}$ equals $h_j^{(i)}$, so the model behaves consistently at test time when dropout is disabled.

Dropout is particularly effective in high-dimensional feature spaces (as in this pipeline) where overfitting is a risk.

Optimization and Training Dynamics

The MLP is trained using the Adam optimizer, which adapts learning rates for each parameter based on first and second moment estimates of gradients:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where g_t is the gradient, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ are momentum parameters, $\alpha = 10^{-3}$ is the learning rate, and $\epsilon = 10^{-8}$ is a small constant for numerical stability.

The loss function is binary cross-entropy (same as LR), but with an additional L2 weight decay term $\lambda \sum_i \|\mathbf{W}^{(i)}\|_F^2$ to regularize the weights.

Early stopping is applied: if the validation loss does not improve for a fixed number of epochs (e.g., 10), training halts to prevent overfitting.

Role of Feature Standardization

Before training the MLP, features are standardized using `StandardScaler`:

$$\tilde{x}_j = \frac{x_j - \mu_j}{\sigma_j}$$

where μ_j and σ_j are the mean and standard deviation of feature j in the training set. Standardization is important for MLPs because:

- **Gradient stability:** Features with large scales (e.g., price in thousands of VND) can dominate gradients, slowing convergence. Standardization ensures all features contribute equally.
- **Activation balance:** If inputs have very different scales, some neurons may saturate (ReLU outputs all zeros or all large values), reducing model expressiveness.

Why MLP Is Appropriate for Financial Data

Financial data often exhibit complex, non-linear relationships:

- **Regime-dependent effects:** The impact of news may differ in bull vs. bear markets, or high vs. low volatility regimes. MLPs can learn these conditional effects.
- **Cross-domain interactions:** News sentiment may interact with technical indicators (e.g., momentum, RSI) in non-obvious ways. MLPs can discover these interactions from data.
- **Noise resilience:** With proper regularization (dropout, weight decay), MLPs can filter noise and focus on robust patterns.

2.5.5 Financial Time-Series Considerations

Financial data have unique properties that distinguish them from standard machine learning datasets. This subsection discusses these properties and how the pipeline addresses them.

Return Definitions: Simple vs. Log Returns

There are two common definitions of returns:

1. **Simple (arithmetic) return:** $r_t = \frac{P_t - P_{t-1}}{P_{t-1}}$
2. **Log return:** $\tilde{r}_t = \log \frac{P_t}{P_{t-1}} = \log(1 + r_t)$

This pipeline uses simple returns because they are more intuitive for trading: a simple return of 0.05 means a 5% gain, which directly translates to profit. Log returns have theoretical advantages (time-additivity, normality assumption), but for short-to-medium horizons (15-40 days), the difference is negligible.

Volatility and Non-Stationarity

Financial returns exhibit **volatility clustering**: periods of high volatility tend to cluster together, and periods of low volatility tend to cluster together. This violates the assumption of constant variance (homoskedasticity) made by many classical models.

To capture this, Agent 3 includes pre-event volatility features:

$$\sigma_{20d}^{\text{pre}} = \sqrt{\frac{1}{20} \sum_{i=1}^{20} (r_{t-i} - \bar{r})^2}$$

This allows the model to condition on the volatility regime: high pre-event volatility may indicate higher uncertainty and lower predictability.

Regime Shifts

Financial markets undergo **regime shifts**: structural changes in the relationship between features and returns due to macroeconomic events (e.g., policy changes, crises). For example, during a credit crisis, leverage ratios become highly predictive; during normal times, they may be irrelevant.

The pipeline's time-based train/test split partially addresses this: the validation set represents a different time period than the training set, testing the model's ability to generalize across regimes. However, if the validation period contains a major regime shift, performance may degrade. Future work could include explicit regime indicators (e.g., VIX-like volatility indices) as features.

Why Time-Based Splits Are Mandatory

In standard machine learning, random train/test splits are common. However, in financial prediction, this creates **temporal leakage**: the model can "see the future" during training. For example, if sample i from January 2025 is in the training set and sample j from December 2024 is in the test set, the model has access to information from a later time period when predicting an earlier one.

To prevent this, the pipeline uses a strict chronological split:

$$\text{Train} = \{(i, \mathbf{x}_i, y_i) : t_0(i) \leq t_{\text{split}}\}, \quad \text{Val} = \{(i, \mathbf{x}_i, y_i) : t_0(i) > t_{\text{split}}\}$$

where t_{split} is the 80th percentile of event dates. This ensures that the model is evaluated on future data, simulating real-world deployment.

2.5.6 Connection Between Theory and Pipeline Implementation

This subsection maps each theoretical concept to its concrete implementation in the pipeline, ensuring traceability.

Table 2.1: Mapping of theoretical concepts to pipeline components.

Theoretical Concept	**Implementation**	**Agent/Module**
Event-driven learning	Structured event extraction from news	Agent 1
Separation of concerns	Modular agent chain with defined interfaces	All agents
Time-based data splitting	Chronological 80/20 split by t_0	Agent 4
Class imbalance handling	`class_weight='balanced'`	Agent 4 (LR)
Logistic Regression	Binary classification with sigmoid output	Agent 4
Multi-Layer Perceptron	96-48-24-1 architecture with ReLU	Agent 4
Dropout regularization	$p = 0.3$ dropout in hidden layers	Agent 4 (MLP)
Feature standardization	`StandardScaler` preprocessing	Agent 4 (MLP)
Volatility clustering	Pre-event volatility features ($\sigma_{20d}^{\text{pre}}$)	Agent 3
Trading calendar alignment	Map publish date to first trading day t_0	Agent 2
Horizon return measurement	$r_h = (P_{t_h} - P_{t_0})/P_{t_0}$	Agent 2
Binary labeling	$y = \mathbb{I}(r_h > 0)$	Agent 2
Hashed text features	`HashingVectorizer(n_features=128)`	Agent 3

Justification of Method Appropriateness at Each Stage

1. **Agent 1:** Rule-based event extraction is appropriate because:
   - The problem is well-structured: detecting company names and keywords.
   - Training data for supervised NER (Named Entity Recognition) in Vietnamese financial text is scarce.
   - Rule-based systems are transparent and easy to audit, which is critical for financial compliance.
2. **Agent 2:** Deterministic labeling based on realized returns is appropriate because:

- It avoids model assumptions (no need to specify a return-generating process).
- Binary labels are robust to outliers (a small number of extreme returns do not distort the label distribution).
- The approach is simple and reproducible, facilitating comparison with future work.

3. **Agent 3:** Hybrid feature engineering (hashing + domain features) is appropriate because:

    - Hashing provides scalability (fixed feature size, no vocabulary growth).
    - Domain features (RSI, MA ratios) incorporate prior knowledge from technical analysis.
    - Interaction terms allow simple non-linearities without complex feature learning.

4. **Agent 4:** Logistic Regression and MLP together provide a balanced evaluation:

    - LR tests whether a linear baseline is sufficient (often the case in noisy financial data).
    - MLP tests whether non-linearity improves performance without overfitting.
    - Both models share the same feature pipeline, ensuring fair comparison.

This theoretical grounding ensures that the pipeline is not an ad-hoc collection of techniques, but rather a principled system where each component is justified by domain knowledge, statistical theory, or software engineering best practices.

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.

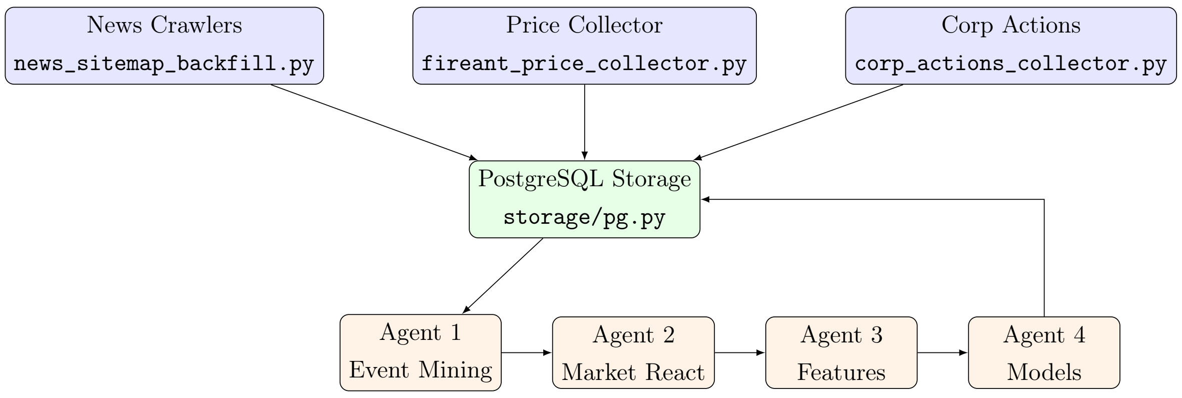

Figure 3.1: Overall VN30 pipeline architecture.

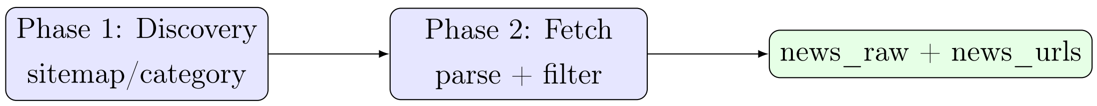

Figure 3.2: Two-phase news collection in `NewsSitemapBackfill`.

$$\begin{tikzcd}\text{news\_raw} \arrow[r, "\text{Agent 1}"] & \text{news\_events} \arrow[r, "\text{Agent 2}"] & \text{market\_reactions} \arrow[r, "\text{Agent 3}"] & \text{features\_joint} \arrow[r, "\text{Agent 4}"] & \text{predictions}\end{tikzcd}$$

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.

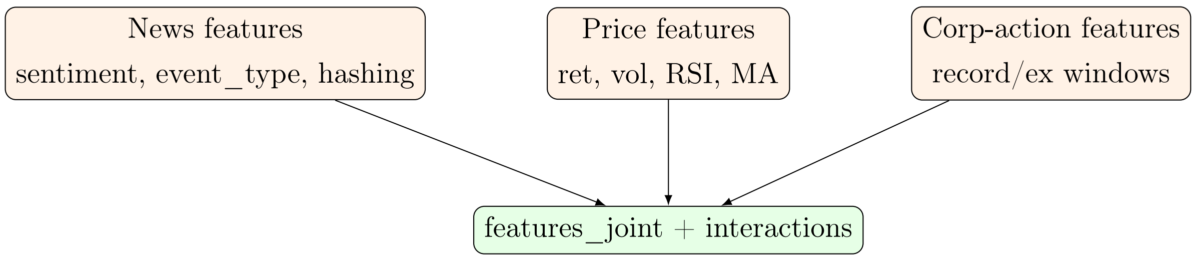

Figure 3.4: Feature construction flow in Agent 3.

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 Evaluation Metrics and Experimental Protocol

This section provides rigorous mathematical definitions of all evaluation metrics used in the pipeline, explains the experimental procedure in detail, and establishes the connection between statistical evaluation and economic utility.

4.1.1 Classification Metrics

The pipeline uses standard binary classification metrics to assess predictive performance on the validation set. Let N_{val} denote the number of validation samples, $y_i \in \{0,1\}$ the true label for sample i, $\hat{p}_i \in [0,1]$ the predicted probability of the positive class, and $\hat{y}_i = \mathbb{I}(\hat{p}_i \geq \tau_{\text{class}})$ the predicted class label using a classification threshold τ_{class} (typically 0.5).

Accuracy

Accuracy measures the proportion of correct predictions:

$$\text{Accuracy} = \frac{1}{N_{\text{val}}} \sum_{i=1}^{N_{\text{val}}} \mathbb{I}(\hat{y}_i = y_i) = \frac{\text{TP} + \text{TN}}{N_{\text{val}}}$$

where TP (true positives) is the count of samples where $\hat{y}_i = 1$ and $y_i = 1$, and TN (true negatives) is the count where $\hat{y}_i = 0$ and $y_i = 0$.

Interpretation: Accuracy provides an overall sense of model performance but can be misleading in the presence of class imbalance. For example, if 70% of labels are positive, a naive model that always predicts positive achieves 70% accuracy without learning anything useful.

Precision

Precision measures the fraction of predicted positives that are actually positive:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

where FP (false positives) is the count of samples where $\hat{y}_i = 1$ but $y_i = 0$.

Interpretation: High precision means that when the model predicts an upward movement, it is usually correct. This is important for trading strategies where false signals incur transaction costs.

Recall (Sensitivity)

Recall measures the fraction of actual positives that are correctly identified:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

where FN (false negatives) is the count of samples where $\hat{y}_i = 0$ but $y_i = 1$.

Interpretation: High recall means the model captures most of the profitable opportunities. However, high recall often comes at the cost of lower precision (i.e., more false positives).

Confusion Matrix

The confusion matrix provides a complete breakdown of prediction outcomes:

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

From the confusion matrix, we can derive:

- **True Negative Rate (Specificity):** $\frac{\text{TN}}{\text{TN+FP}}$ — the fraction of actual negatives correctly identified.
- **False Positive Rate:** $\frac{\text{FP}}{\text{TN+FP}} = 1 - \text{Specificity}$ — the fraction of actual negatives incorrectly classified as positive.
- **False Negative Rate:** $\frac{\text{FN}}{\text{TP+FN}} = 1 - \text{Recall}$ — the fraction of actual positives incorrectly classified as negative.

ROC-AUC (Area Under the Receiver Operating Characteristic Curve)

The ROC curve is a graphical representation of the trade-off between the true positive rate (recall) and the false positive rate as the classification threshold τ_{class} varies from 0 to 1. Formally, for a given threshold τ:

$$\text{TPR}(\tau) = \frac{\sum_{i=1}^{N_{\text{val}}} \mathbb{I}(\hat{p}_i \geq \tau, y_i = 1)}{\sum_{i=1}^{N_{\text{val}}} \mathbb{I}(y_i = 1)}$$

$$\text{FPR}(\tau) = \frac{\sum_{i=1}^{N_{\text{val}}} \mathbb{I}(\hat{p}_i \geq \tau, y_i = 0)}{\sum_{i=1}^{N_{\text{val}}} \mathbb{I}(y_i = 0)}$$

The ROC curve is the set of points $\{(\text{FPR}(\tau), \text{TPR}(\tau)) : \tau \in [0, 1]\}$. The Area Under the Curve (AUC) is:

$$\text{AUC} = \int_0^1 \text{TPR}(\text{FPR}^{-1}(x))\, dx$$

In practice, AUC is computed using the trapezoidal rule over discrete threshold values.

Interpretation: AUC represents the probability that a randomly chosen positive sample has a higher predicted probability than a randomly chosen negative sample. An AUC of 0.5 indicates random performance (no discrimination), while an AUC of 1.0 indicates perfect separation. In financial prediction, AUC values between 0.55 and 0.65 are typical for news-driven models, reflecting the inherent noise and non-stationarity of markets.

Why AUC Is Valuable: Unlike accuracy, AUC is threshold-independent and robust to class imbalance. It evaluates the model's ability to rank samples correctly, which is crucial for strategies that select top-N signals rather than classifying all samples.

4.1.2 Economic Evaluation: Trading Metrics Based on Model Outputs

While classification metrics measure predictive accuracy, they do not directly quantify economic utility. A model with 55% accuracy could be highly profitable if its errors are small and its correct predictions capture large price movements. Conversely, a model with 70% accuracy might be unprofitable if transaction costs exceed gains. To address this, the pipeline includes a simple backtest that simulates trading decisions based on model outputs.

Probability Threshold and Signal Generation

The backtest applies a probability threshold τ_{trade} (distinct from the classification threshold τ_{class}) to generate trading signals. Specifically, for each validation sample i, a signal is generated if:

$$\hat{p}_i \geq \tau_{\text{trade}}$$

In this pipeline, $\tau_{\text{trade}} = 0.55$ is used to filter out low-confidence predictions and focus on high-conviction signals. This threshold is chosen heuristically: values below 0.5 would include negative predictions, while values much above 0.55 would drastically reduce trade count. Future work could optimize τ_{trade} via cross-validation or rolling-window evaluation, but for comparability across horizons and models, a fixed threshold is used.

Number of Trades

The number of trades executed in the backtest is simply the count of samples where the probability exceeds the threshold:

$$n_{\text{trades}} = \sum_{i=1}^{N_{\text{val}}} \mathbb{I}(\hat{p}_i \geq \tau_{\text{trade}})$$

Interpretation: A higher n_{trades} indicates that the model generates more signals, which can increase opportunities but also increases transaction costs and exposure. A lower n_{trades} indicates selectivity, which may improve signal quality but reduces diversification.

Mean Return per Trade

For each sample i where a trade is executed (i.e., $\hat{p}_i \geq \tau_{\text{trade}}$), the realized return over the horizon is $r_{h,i}$ (the actual return from t_0 to t_h, computed in Agent 2). The mean return per trade is:

$$\text{mean\_ret\_h} = \frac{1}{n_{\text{trades}}} \sum_{i:\hat{p}_i \geq \tau_{\text{trade}}} r_{h,i}$$

Interpretation: This metric measures the average profitability of executed trades. A positive mean_ret_h indicates that, on average, the model's signals lead to profitable outcomes. For example, mean_ret_h = 0.0873 (as observed for the LR model at horizon 40) means that each executed trade yields an average return of 8.73% over 40 trading days, which corresponds to an annualized return of approximately $0.0873 \times (252/40) \approx 55\%$ (assuming 252 trading days per year), though this ignores compounding and transaction costs.

Important Caveat: mean_ret_h does not account for:

- **Transaction costs:** Brokerage fees, bid-ask spreads, and slippage reduce realized returns.
- **Opportunity cost:** Capital is locked for h days, limiting the ability to exploit other opportunities.
- **Risk:** High mean_ret_h with high volatility may be less desirable than moderate mean_ret_h with low volatility.

Hit Rate

The hit rate measures the proportion of executed trades that result in positive returns:

$$\text{hit\_rate} = \frac{1}{n_{\text{trades}}} \sum_{i:\hat{p}_i \geq \tau_{\text{trade}}} \mathbb{I}(r_{h,i} > 0)$$

Interpretation: A hit rate of 0.648 (as observed for LR at horizon 40) means that 64.8% of trades are profitable. This is a stronger metric than accuracy because it directly measures trading success, not just label prediction. A hit rate above 0.5 indicates positive expectancy (assuming symmetric position sizes and ignoring costs), while a hit rate above 0.6 is generally considered strong for equity strategies.

Relationship to Precision and Recall: Hit rate is similar to precision but measured on realized returns rather than predicted labels. If $r_{h,i} > 0$ aligns well with $y_i = 1$, then hit rate $\approx$ precision. However, if some samples have $\hat{p}_i \geq \tau_{\text{trade}}$ but $y_i = 0$ (false positives) yet still yield positive returns due to market noise, hit rate can exceed precision.

Risk-Adjusted Metrics (Future Extension)

While not currently implemented, future versions of the pipeline could include:

- **Sharpe Ratio:** $\frac{\text{mean\_ret\_h}}{\text{std\_ret\_h}}$, where std_ret_h is the standard deviation of returns among executed trades. This adjusts for volatility.
- **Maximum Drawdown:** The largest peak-to-trough decline in cumulative returns, measuring downside risk.
- **Win/Loss Ratio:** $\frac{\text{mean return of winning trades}}{\text{mean return of losing trades}}$, indicating whether wins are larger than losses on average.

4.1.3 Experimental Procedure and Evaluation Protocol

This subsection describes the step-by-step experimental procedure, ensuring reproducibility and transparency.

Data Preparation and Filtering

1. **Data Collection:** The pipeline collects news, prices, and corporate actions for the VN30 universe over the period January 1, 2023 to January 31, 2026 (approximately 3 years).
2. **Event Extraction (Agent 1):** Process all articles in `news_raw` to generate structured events in `news_events`. This stage is deterministic and can be rerun without re-fetching data.
3. **Market Reaction Labeling (Agent 2):** For each event, compute t_0 and horizon returns. Two horizons are evaluated independently: $h = 15$ days and $h = 40$ days. Events without sufficient price data are dropped with a logged reason (e.g., `insufficient_horizon`). The resulting `market_reactions` table contains separate rows for each horizon.
4. **Feature Engineering (Agent 3):** For each event-horizon pair with a valid reaction, construct the feature vector $\mathbf{x}(e)$ by merging news features, price-context features (computed using only pre-event data), and corporate action features. Store in `features_joint`.
5. **Horizon Filtering:** If the environment variable `TRAIN_HORIZON_DAYS` is set (e.g., to 15 or 40), Agent 4 filters the dataset to include only samples matching that horizon. This ensures that models are trained and evaluated on a single horizon, preventing cross-horizon contamination.

Time-Based Train/Validation Split

To simulate realistic forward prediction, the pipeline uses a strict chronological split:

1. **Sort Samples:** Order all samples by t_0 (the first trading day after event publication) in ascending order.
2. **Compute Split Index:**
$$i_{\text{split}} = \lfloor 0.8 \times N \rfloor$$
where N is the total number of samples for the given horizon.
3. **Assign Sets:**
   - Training set: $\{(i, \mathbf{x}_i, y_i) : i \leq i_{\text{split}}\}$ (earliest 80%)
   - Validation set: $\{(i, \mathbf{x}_i, y_i) : i > i_{\text{split}}\}$ (most recent 20%)

Rationale: This split ensures that the model is trained on past data and evaluated on future data, mimicking deployment conditions. Random splits would allow the model to learn from future events, which is unrealistic and inflates performance estimates.

Feature Vectorization and Sanitization

1. **DictVectorizer:** Convert feature dictionaries (stored as JSONB in `features_joint`) to sparse numerical matrices $\mathbf{X}_{\text{train}} \in \mathbb{R}^{N_{\text{train}} \times D}$ and $\mathbf{X}_{\text{val}} \in \mathbb{R}^{N_{\text{val}} \times D}$. The vectorizer learns the feature vocabulary from the training set and applies it consistently to the validation set.
2. **Missing Value Sanitization:** Replace `None` or `NaN` values with 0.0. This is a conservative choice: missing features contribute zero to the linear combination, effectively indicating "no information". More sophisticated imputation (e.g., mean/median fill) was not used to avoid introducing bias.

Model Training: Logistic Regression

1. **Initialize Model:**

```
model = LogisticRegression(
    max_iter=LR_MAX_ITER, # e.g., 500
    class_weight='balanced', # handle imbalance
    solver='lbfgs', # L-BFGS optimizer
    random_state=42 # reproducibility
)
```

2. **Fit Model:** Train on $(\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}})$ by minimizing the weighted cross-entropy loss.
3. **Predict Probabilities:**

$$\hat{\mathbf{p}}_{\text{val}} = \text{model.predict\_proba}(\mathbf{X}_{\text{val}})[:, 1]$$

   Extract the probability of the positive class (upward movement).

4. **Compute Classification Labels:** $\hat{y}_i = \mathbb{I}(\hat{p}_i \geq 0.5)$ for metric computation.

Model Training: Multi-Layer Perceptron

1. **Feature Standardization:**

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.toarray())
X_val_scaled = scaler.transform(X_val.toarray())
```

The scaler computes mean and standard deviation from the training set and applies the same transformation to the validation set.

2. **Define Architecture:**

```
model = nn.Sequential(
    nn.Linear(D, 96), # Input -> Hidden 1
    nn.ReLU(),
    nn.Dropout(0.3), # Regularization
    nn.Linear(96, 48), # Hidden 1 -> Hidden 2
    nn.ReLU(),
    nn.Linear(48, 24), # Hidden 2 -> Hidden 3
    nn.ReLU(),
    nn.Linear(24, 1), # Hidden 3 -> Output
    nn.Sigmoid() # Probability output
)
```

3. **Training Loop:**

   - **Optimizer:** Adam with learning rate $\alpha = 10^{-3}$ and weight decay $\lambda = 10^{-4}$ (L2 regularization).
   - **Loss Function:** Binary cross-entropy (BCELoss).
   - **Batch Size:** 256 samples per mini-batch.
   - **Epochs:** Train for up to 100 epochs with early stopping if validation loss does not improve for 10 consecutive epochs.

4. **Predict Probabilities:** Forward pass on $\mathbf{X}_{\text{val}}$ to obtain $\hat{\mathbf{p}}_{\text{val}}$.

Metric Computation

For both models, compute the following on the validation set:

1. **Classification Metrics:**

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    roc_auc_score

accuracy = accuracy_score(y_val, y_pred)
precision = precision_score(y_val, y_pred)
recall = recall_score(y_val, y_pred)
auc = roc_auc_score(y_val, proba_val)
```

2. **Backtesting:**

```python
def backtest(rows, proba, threshold=0.55):
    returns = []
    hits = []
    for row, p in zip(rows, proba):
        if p >= threshold:
            ret = row['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))
    }
```

Comparison Across Horizons and Models

The pipeline trains four model-horizon combinations:

1. Logistic Regression, $h = 15$ days
2. Logistic Regression, $h = 40$ days
3. MLP 96-48-24-1, $h = 15$ days
4. MLP 96-48-24-1, $h = 40$ days

For each combination, the pipeline:

- Trains on the earliest 80% of events for that horizon.
- Evaluates on the most recent 20%.
- Stores metrics in the `models` table with metadata (horizon, model type, feature count, sample sizes).
- Stores predictions in the `predictions` table for further analysis.

4.1.4 Why Two Horizons Are Evaluated

The choice to evaluate both 15-day and 40-day horizons addresses the following research questions:

1. **Signal Persistence:** Do news signals have predictive power only in the short term (15 days), or do they persist over medium-term horizons (40 days)?

2. **Noise vs. Signal Trade-off:** Shorter horizons may capture immediate market reactions but are also more susceptible to microstructure noise (e.g., intraday volatility, order flow effects). Longer horizons average out noise but may introduce confounding from unrelated events.

3. **Trading Strategy Implications:** A 15-day holding period is suitable for active traders, while a 40-day horizon is more aligned with swing trading or position trading. By evaluating both, the pipeline provides insights for different investor profiles.

4. **Model Sensitivity:** Some models may perform well on short-term predictions (where immediate sentiment dominates) but poorly on long-term predictions (where fundamentals matter more). Comparing performance across horizons reveals model strengths and weaknesses.

4.1.5 Probability Threshold Selection and Sensitivity

The backtest uses a fixed threshold $\tau_{\text{trade}} = 0.55$. This choice is motivated by:

- **Focus on High-Confidence Signals:** A threshold above 0.5 filters out marginal predictions, focusing on samples where the model is relatively confident of upward movement.
- **Consistency Across Comparisons:** Using the same threshold for all models and horizons ensures fair comparison.
- **Practical Relevance:** In real trading, not all signals are acted upon; traders often require a minimum confidence level (e.g., $p \geq 0.6$) to justify transaction costs.

Sensitivity Analysis (Not Currently Implemented): Future work should explore how backtest metrics vary with τ_{trade}. For example, plotting hit_rate(τ) and mean_ret_h(τ) for $\tau \in [0.5, 0.9]$ would reveal:

- Whether there is an optimal threshold that maximizes risk-adjusted returns.
- How trade count decreases as τ increases (higher thresholds $\rightarrow$ fewer trades).
- Whether the relationship is stable across horizons and models.

4.1.6 Backtesting Step-by-Step Execution

The backtesting procedure can be formalized as a sequence of steps:

1. **Input:** Validation set $\{(i, \mathbf{x}_i, y_i, r_{h,i}, t_{0,i}) : i \in \mathcal{I}_{\text{val}}\}$ and predicted probabilities $\{\hat{p}_i : i \in \mathcal{I}_{\text{val}}\}$.

2. **Signal Generation:** For each i, generate signal $s_i = \mathbb{I}(\hat{p}_i \geq \tau_{\text{trade}})$.

3. **Filter Executed Trades:** Define the set of executed trades:

$$\mathcal{T} = \{i \in \mathcal{I}_{\text{val}} : s_i = 1 \text{ and } r_{h,i} \text{ is not null}\}$$

4. **Compute Trade Count:** $n_{\text{trades}} = |\mathcal{T}|$.

5. **Compute Mean Return:**

$$\text{mean\_ret\_h} = \frac{1}{n_{\text{trades}}} \sum_{i \in \mathcal{T}} r_{h,i}$$

6. **Compute Hit Rate:**

$$\text{hit\_rate} = \frac{1}{n_{\text{trades}}} \sum_{i \in \mathcal{T}} \mathbb{I}(r_{h,i} > 0)$$

7. **Output:** Store $\{n_{\text{trades}}, \text{mean\_ret\_h}, \text{hit\_rate}\}$ in the model metadata table.

Important Note: The backtest does not model position sizing (all trades are assumed equal weight), portfolio-level effects (correlations between trades), or transaction costs. These simplifications make the backtest optimistic but transparent. Future enhancements could include:

- **Transaction Costs:** Subtract a fixed percentage (e.g., 0.2%) from each return to account for brokerage and slippage.
- **Position Sizing:** Weight trades by predicted probability or by inverse volatility.
- **Portfolio Constraints:** Limit the number of concurrent positions or enforce sector diversification.

4.1.7 Summary of Evaluation Philosophy

The dual evaluation framework (classification metrics + backtest metrics) reflects the principle that **statistical performance and economic performance are distinct but complementary**. High AUC indicates good discrimination but does not guarantee profitability. Conversely, a simple threshold-based strategy can be profitable even with modest AUC if it captures a subset of high-return signals. By reporting both types of metrics, the pipeline provides a complete picture that supports both model development (where AUC guides architecture choices) and strategy deployment (where mean_ret_h and hit_rate guide trading decisions).

4.2 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.

4.2.1 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

4.2.2 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.

4.2.3 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.

4.3 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

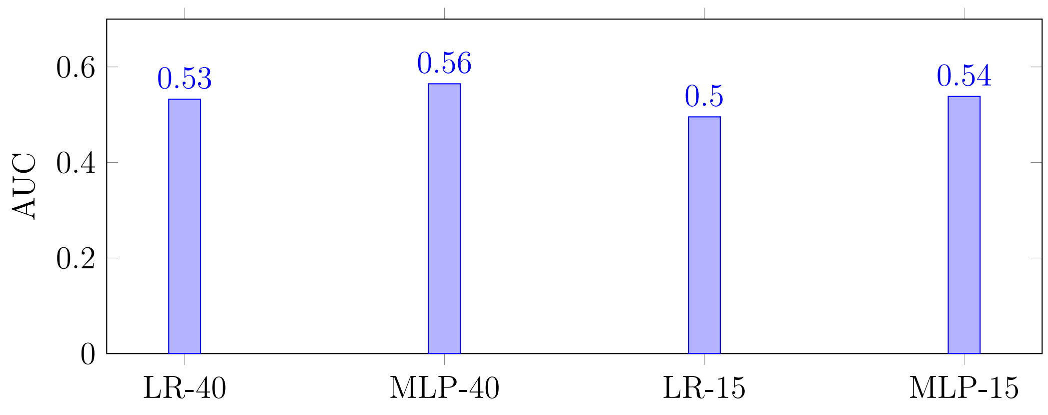

Figure 4.1: AUC comparison by horizon and model.

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`

```
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`

```
@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

```sql
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

```sql
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()
```