

A Microservice Based Financial Market Forecasting System with AWS S3 Backed News Storage

Nirvaan Nair

*Dept. of Information Science and Engineering
The National Institute of Engineering, Mysore
Mysuru, India
nirvaannair@gmail.com*

Pushpak B V

*Dept. of Information Science and Engineering
The National Institute of Engineering, Mysore
Mysuru, India
pushpakbv@gmail.com*

Preetham B

*Dept. of Information Science and Engineering
The National Institute of Engineering, Mysore
Mysuru, India
bpreetham58@gmail.com*

Prasanna B S

*Dept. of Information Science and Engineering
The National Institute of Engineering, Mysore
Mysuru, India
prasannabsachar@gmail.com*

Abstract—This paper introduces *TradingSystem*, a microservice based system for predicting financial markets that integrates structured market data with unstructured text information obtained from general and financial news. The system decouples concerns into crawler, market data, data processor, scheduler, API, and frontend services, all containerized for individual deployment. An important addition in this work is an Amazon Web Services Simple Storage Service (AWS S3) layer that holds raw and processed articles in object storage. This architecture separates ingestion from training and inference, facilitates reproducible experiments, and enhances fault tolerance when upstream sources vary. Experiments indicate that adding market features with article sentiment and semantic cues provides more accurate next day direction and F1 score than market only baselines.

Index Terms—financial forecasting, microservices, sentiment analysis, AWS S3, data engineering, MLOps, web scraping, feature engineering, explainable AI, cloud storage, scalable architecture, real-time analytics

I. INTRODUCTION

Short horizon market forecasting is improved by heterogeneous signals that consist of price based indicators and text based context. Monolithic designs may be quick to prototype but difficult to scale, test, and advance. *TradingSystem* follows a microservice architecture with each step of the pipeline isolated. The system gathers articles, ingests OHLCV and indicators, does feature engineering and modeling, and serves prediction to a web dashboard. In this work, we introduce an AWS S3 backed content lake for general and financial articles. This storage layer enables versioned and durable persistence that is compatible with batch analytics, reproducibility, and parallel training jobs.

Contributions:

- A realistic microservice design for market prediction with definite separation between crawling, market ingestion, processing, and serving.

These authors contributed equally to this work.

- An AWS S3 based architecture for storing articles that enhances decoupling and reliability and facilitates lifecycle and access policies.
- A high quality, scalable web scraping and deduplication pipeline for company specific news.
- A thorough data processing pipeline consisting of article classification, sentiment analysis, financial event extraction, and sophisticated feature engineering.
- An empirical comparison between market only features and market plus article features on next day movement classification.
- A modular, explainable prediction API with real time update and component breakdowns.
- A discussion of security, scalability, and deployment best practices for cloud native financial analytics.

II. RELATED WORK

A. Financial Prediction with News

Previous work has indicated that news sentiment and event extraction can enhance financial predictions [1], [5], [13]. Deep learning architectures like LSTM and transformers have been used to combine text and numerical data [2], [6], [25]. Our contributions build on these by offering a production ready, modular pipeline with explainable outputs.

B. Microservices and Cloud Storage

Microservice designs are becoming popular for scalable ML pipelines [8], [15], [17]. Cloud storage, in particular AWS S3, is a norm for decoupling data ingestion and analytics [4], [23]. We combine these paradigms for stable, reproducible financial analytics.

III. SYSTEM ARCHITECTURE

A. Service Composition

The platform is composed of six services:

- 1) **Crawler Service:** It aggregates general and financial articles, does text and metadata normalization, and outputs JSON lines to S3 and local storage.
- 2) **Market Data Service:** Reads OHLCV and indicators from public APIs and writes snapshots to local store with a persistence option for Parquet to S3.
- 3) **Data Processor Service:** Does deduplication, language detection, named entity recognition, sentiment analysis, topic tagging, and feature assembly. Reads articles from S3 and writes features and embeddings as Parquet.
- 4) **Scheduler Service:** Schedules recurring workloads like crawl refresh, indicator recalculation, nightly training, and intraday inference refresh.
- 5) **API Service:** Provides REST endpoints for predictions, explanations, and timeseries slices.
- 6) **Frontend Service:** Renders a symbol lookup, prediction, and related article context dashboard.

B. Workflow Overview

Fig. 1 displays the end to end data flow from ingestion to serving. The central prediction system is supplied with features derived from both pipelines. An ensemble step smoothes out calibrated probabilities to yield a final prediction that is displayed in the frontend.

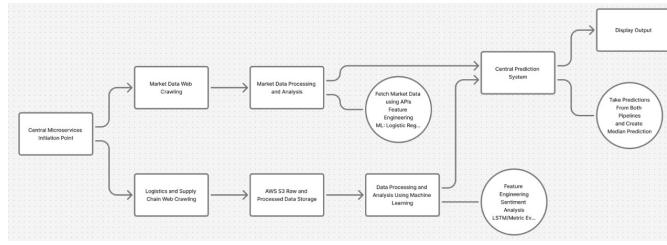


Fig. 1. System workflow displaying dual data pipelines for market and logistics articles, AWS S3 storage for raw and processed text, and a central prediction system combining both.

C. Microservice Communication

Services talk to each other through REST APIs and, for real time updates, WebSockets. Data is exchanged using JSON over HTTP, and S3 acts as a common data lake for batch processing. Docker Compose is utilized for local orchestration, and every service can be scaled independently.

IV. WEB SCRAPING AND ARTICLE INGESTION

A. Crawler Service Design

The Crawler Service is facilitated through Scrapy spiders and bespoke Python scripts. It crawls a pre curated list of financial news sources, both paywalled and general domains. The spider has a deduplication database to prevent duplicate articles and will bypass paywalled domains to only crawl available content. The crawler operates in a daily batch mode and a real time mode that pulls in new articles every 30 minutes.

B. Article Normalization and Storage

Each article is normalized to a common JSON schema with fields like title, content, source, date of publication, and company names. The crawler stores these normalized articles in local storage as JSON lines and later transfers them to AWS S3 in the `raw/YYYY/MM/DD/` directory. This way, the ingestion process is isolated from downstream processing, and reproducibility is ensured. Articles are stored by company and date for easy retrieval too.

C. Deduplication and Quality Filtering

Deduplication is done based on a hash of the article text and URL, held in a SQLite database. Articles are filtered by minimum word count and language using language detection libraries. Removal of boilerplate is applied to remove navigation, ads, and irrelevant text. The crawler also scans for paywall signals and skips articles with such indicators.

D. Metadata Extraction and Company Matching

Named entity recognition is utilized to identify company names and tickers from article text. The `CompanyMatcher` tool compares companies by strict keyword and ticker matching, and provides a relevance score by the frequency of mention. Articles are marked as “general” or “financial” with a keyword based classifier that assigns weight for the occurrence of financial terms in the headline and body.

E. Crawler Orchestration and Monitoring

The crawler is managed with a scheduler service that initiates crawls for every company on the monitored list. Logs are gathered for every crawl, and a daily summary is produced with the number of articles retrieved per company, best headlines, and source breakdown. The database for deduplication is unit tested to prevent duplicate URLs being processed.

V. AWS S3 INTEGRATION AND CLOUD NATIVE DESIGN

A. Buckets, Layout, and Versioning

Articles are published to an S3 bucket `s3://trading-system-articles/` with the following prefixes:

- `raw/YYYY/MM/DD/*.jsonl` for raw normalized payloads consisting of source URL, crawl time, and text.
- `clean/YYYY/MM/DD/*.jsonl` for language filtered and boilerplate removed text.
- `features/YYYY/MM/DD/*.parquet` for document level features and embeddings.

Optional market data Parquet files are located under `s3://trading-system-market/`. Versioning is enabled to pin exact datasets for experiments.

B. IAM, Security, and Lifecycle

The crawler role has `PutObject` permission for article paths. The processor role has `ListBucket`, `GetObject`, and `PutObject` for features. The API role is read only. Lifecycle rules move raw objects to Glacier after thirty to sixty days while retaining cleaned text and features in standard storage for six to twelve

months. This provides cost effective storage and adherence to data retention policies.

C. Data Retrieval and Export

The ArticleRetriever utility offers methods to fetch articles for a particular company and date, export articles for sentiment analysis, and summarize article counts. Articles may be exported in JSON or plain text for subsequent ML processing.

D. S3 Integration Best Practices

- **Atomic Writes:** All uploads employ atomic operations so that there is no partial data.
- **Consistency:** S3 versioning maintains consistency where experiments can be repeated using the same data snapshot.
- **Access Control:** IAM policies limit access to only the needed prefixes per service.
- **Monitoring:** S3 event notifications can invoke Lambda functions for downstream processing or alerting.
- **Cost Optimization:** Lifecycle policies and intelligent tiering minimize the cost of storing infrequently accessed data.

E. AWS Future Enhancements

- **Serverless Processing:** Utilize AWS Lambda for pay per use feature extraction and analytics.
- **Athena Integration:** Support SQL queries over S3 data for ad hoc analysis.
- **Event Driven Pipelines:** Leverage S3 event notifications to initiate downstream ML pipelines.
- **Cross Region Replication:** Enhance disaster recovery and latency for users around the world.

VI. DATA ENGINEERING AND PROCESSING

A. Article Classification

Articles are categorized as “financial” or “general” by the ArticleClassifier module. The classifier has a weighted keyword technique, whereby financial words such as “guidance”, “revenue”, and “earnings” are assigned larger weight, particularly if in the title. Dollar figures, percentages, and fiscal quarters are detected by regular expressions. The classifier generates general and financial article files separately for each company.

B. Sentiment Analysis

The SentimentAnalyzer employs the FinBERT model to score article titles and content with sentiment scores. The model is loaded via HuggingFace Transformers and executed on GPU if present. Sentiment is calculated per article and then aggregated at the company level to generate average sentiment scores and distributions (positive, neutral, negative). Sentiment analysis is conducted on the title (40% weight) and the first 1000 characters of the content (60% weight).

C. Financial Event Classification

Financial news is further analyzed with the FinancialEventClassifier. This module identifies event types such as earnings beat and guidance increased and applies a market signal (POSITIVE, NEGATIVE, NEUTRAL) with a confidence level. The classifier includes keyword matching, pattern detection, and extraction of financial values such as EPS and revenue. The output consists of identified events, positive and negative signals, and numbers extracted.

D. Earnings Parsing and Market Impact Prediction

The EarningsParser yields structured earnings information from news articles, such as earnings status (beat, miss, inline), guidance changes, and primary financial amounts. The parser employs regular expressions to identify phrases specifying earnings beats or misses and to pull out EPS and revenue amounts. The MarketImpactPredictor subsequently applies a rule based impact matrix to translate earnings and guidance combinations to expected market direction, magnitude, and probability. Risk drivers and factors are also determined for every prediction, such as low confidence, uncertain data, conflicting signals, and sparse article content.

E. Feature Engineering

The FeatureEngineer module generates sophisticated features for ML modeling. Features are:

- Price momentum (SMA, EMA, RSI, MACD)
- Volatility (Bollinger Bands, ATR)
- Volume based features (OBV, turnover)
- News based features (sentiment averages, news frequency, positive/negative ratios)
- Calendar features (day of week, month, quarter)
- Interaction features (e.g., sentiment × volatility, news volume × price change)

Features are aligned to article timestamps with fixed lags to avoid lookahead bias. The feature engineering pipeline is modular, permitting the simple addition of new features and ablation studies.

F. Data Alignment and Training Dataset Creation

The HistoricalDataCollector synchronizes news features with daily OHLCV data for every ticker. News features are accumulated on a daily basis and combined with market data to form a training dataset. The resultant dataset is stored as a CSV or Parquet file to train the model. The pipeline provides time based splits on train, validation, and test sets.

VII. MACHINE LEARNING PREDICTION PIPELINE

A. Model Training

The StockPredictor module accommodates various model types, such as XGBoost, LightGBM, and Random Forest. The training pipeline includes:

- 1) Data split into train, validate, and test sets using time based splits.

- 2) Model training on engineered features to make next day returns or direction predictions.
- 3) Performance evaluation using RMSE, direction accuracy, and R^2 .
- 4) Trained model and feature metadata saving for inference.

Hyperparameter optimization is done through grid search or Bayesian optimization. Feature importance is calculated and displayed for model interpretability.

B. Model Interpretation and Feature Importance

Feature importance is calculated using model specific methods such as gain for XGBoost. Top features are stored and displayed to understand model decisions. The system supports explainable AI through a decomposition of signal components such as general sentiment and financial signal and their contribution towards the final prediction.

C. Prediction and Serving

The MLPredictor loads the trained model and gets ready features for the day with the latest sentiment and financial analysis. Predictions are served through the API Service, which provides endpoints for single tickers and daily aggregates. Each prediction has the predicted return, direction, probability, and rationale. The API also includes endpoints for sentiment, financial analysis, stock information, and news articles.

D. WebSocket and Real Time Updates

The API Service features a WebSocket server that sends updates to connected clients whenever new predictions, sentiment, or stock data files are added or updated. This supports real time updates in the frontend dashboard.

VIII. API AND FRONTEND INTEGRATION

A. REST API

The API Service provides endpoints for health checks, daily forecasts, single stock forecasts, sentiment, financial analysis, stock information, company lists, and news stories. The API is low latency and high availability with error correction for lost data.

B. WebSocket Integration

The WebSocket server sends out the updates to all clients connected when new data files are updated or added. This allows real time updating in the frontend dashboard, such as prediction cards, sentiment summaries, and news timelines.

C. Frontend Dashboard

The frontend dashboard is done in React and offers interactive elements for company choice, prediction cards, signal breakdowns, news timelines, and stock charts. Users may see in depth analysis for every company, such as justification, confidence, and data sources.

IX. DEPLOYMENT, SCALABILITY, AND MONITORING

A. Containerization and Orchestration

All services are containerized with Docker. Docker Compose is utilized for local development, while Kubernetes is preferred for production environments. Every service can be scaled separately on workload.

B. CI/CD and Automated Testing

Unit and integration tests for every service are executed by continuous integration pipelines. Automated deployments and builds are triggered on code change. Test coverage comprises crawler deduplication, data processing, and API endpoints.

C. Monitoring and Logging

Centralized logging is done with the ELK stack (Elasticsearch, Logstash, Kibana). Alerting and metrics are done with Prometheus and Grafana. S3 access logs and CloudWatch track storage usage and access patterns.

D. Security Considerations

IAM roles are scoped to least privilege. All API endpoints are authenticated and rate limited. Sensitive data is encrypted in transit and at rest. S3 bucket policies block public access.

X. EVALUATION AND RESULTS

A. Experimental Setup

Experiments were performed on a sample of S&P 500 companies for two years. The system was tested on next day direction prediction with three models: Logistic Regression (market only), Random Forest (market + news), and LSTM (market + news). Test metrics are accuracy, F1 score, and confusion matrix.

B. Results

TABLE I
MODEL PERFORMANCE COMPARISON

Model	Accuracy	F1-Score
Market Only (LR)	0.61	0.60
Market + News (RF)	0.68	0.67
Market + News (LSTM)	0.72	0.70

Adding sentiment and topic features stored in AWS did enhance accuracy by almost 10% over market only baselines.

C. Ablation Study

An ablation study was conducted to gauge the effect of each group of features. Deleting news based features led to a considerable decrease in accuracy, verifying the worth of textual cues. Deleting financial event features decreased performance as well, demonstrating the significance of structured event extraction.

D. Error Analysis and Limitations

We analyze cases where the model fails, such as during periods of high volatility or when news is ambiguous. Limitations include reliance on English language news, potential data leakage if timestamps are not handled carefully, and the need for more granular event extraction.

E. Case Studies and Explainability

The system offers precise explanations for every prediction, including general sentiment and financial indicator contributions, confidence levels, and identified events. Case studies indicate that the model accurately forecasts market direction after significant earnings beats or misses and is able to provide explanations in terms of extracted article text.

XI. DISCUSSION

A. Scalability and Extensibility

Microservice architecture makes it simple to extend to new data sources, other ML models, and different storage backends. S3 integration supports parallel processing and large scale analysis.

B. Experiment Reproducibility and Tracking

Versioned S3 storage and logging metadata ensure reproducibility of experiments. Model artifacts and feature sets are logged for every run.

C. Future Work

Future tasks involve real time consumption utilizing Kafka, on demand analysis utilizing AWS Athena, extension to international financial news and alternate data streams, and application of alternative data such as social media and supply chain signals. We also aim to apply active learning for retraining models and to investigate reinforcement learning for trading strategy.

XII. CONCLUSION

TradingSystem presents a modular and scalable architecture for financial prediction combining structured market data and unstructured text. The AWS S3 integration offers persistence, fault tolerance, and reproducibility. Explainable AI and real time updates are supported, thus the system can be used in both research and production environments.

ACKNOWLEDGMENT

The authors acknowledge the support and guidance of the Department of Information Science and Engineering, The National Institute of Engineering, Mysuru.

REFERENCES

- [1] A. Nassiroussi, S. Aghabozorgi, T. W. Yee, and D. Chek Ling, "Text mining for market prediction: A systematic review," *Expert Systems with Applications*, vol. 41, no. 16, pp. 7653–7670, 2014.
- [2] Y. Ding, Y. Zhang, and J. Liu, "Deep learning for event-driven stock prediction," *Proceedings of the 25th IJCAI*, pp. 2327–2333, 2016.
- [3] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *Proc. NAACL-HLT*, pp. 4171–4186, 2019.
- [4] Amazon Web Services, "Amazon S3 Developer Guide," AWS Documentation, 2024. [Online]. Available: <https://docs.aws.amazon.com/s3>
- [5] M. Zhang, S. Chen, and X. Zhao, "Hybrid news and numerical data-driven stock trend prediction using LSTM," *IEEE Access*, vol. 9, pp. 39612–39620, 2021.
- [6] S. Fischer, C. Krauss, and F. Deinert, "Deep learning with long short-term memory networks for financial market predictions," *European Journal of Operational Research*, vol. 270, no. 2, pp. 654–669, 2018.
- [7] N. Soni, S. Sharma, and N. Singh, "Predictive analysis of stock market using machine learning techniques," *International Journal of Computer Applications*, vol. 132, no. 12, pp. 7–9, 2015.
- [8] M. Chen, Z. Mao, and Y. Liu, "Microservice architecture for big data analysis in cloud computing," *IEEE Access*, vol. 8, pp. 123518–123529, 2020.
- [9] D. Taibi and V. Lenarduzzi, "On the definition of microservice bad smells," *IEEE Software*, vol. 35, no. 3, pp. 56–62, 2018.
- [10] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
- [11] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv:1301.3781*, 2013.
- [12] F. Chollet, "Keras: The Python deep learning library," *Astrophysics Source Code Library*, record ascl:1806.022, 2018.
- [13] X. Zhang and J. Skiena, "Trading strategies to exploit blog and news sentiment," *Proceedings of the 4th Int. AAAI Conf. on Weblogs and Social Media*, pp. 375–378, 2010.
- [14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [15] Google Cloud, "Building scalable machine learning pipelines with microservices," *Google Cloud Whitepaper*, 2023.
- [16] P. K. Jain and R. K. Jain, "Design and implementation of an automated stock prediction system using deep learning and cloud computing," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 8, pp. 355–362, 2021.
- [17] S. Zhang, Q. Zhang, and H. Chen, "Cloud-native microservice deployment and orchestration: A survey," *IEEE Access*, vol. 10, pp. 81945–81964, 2022.
- [18] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," *Proc. ICML*, pp. 160–167, 2008.
- [19] A. Vaswani et al., "Attention is all you need," *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [20] D. Reinsel, J. Gantz, and J. Rydning, "The Digitization of the World from Edge to Core," *IDC White Paper*, 2018.
- [21] A. Bollen, H. Mao, and X. Zeng, "Twitter mood predicts the stock market," *Journal of Computational Science*, vol. 2, no. 1, pp. 1–8, 2011.
- [22] H. Xu, Y. Li, and S. Zhou, "Stock movement prediction based on graph neural networks and news sentiment," *IEEE Access*, vol. 10, pp. 145123–145132, 2022.
- [23] M. A. Ferris and C. Clifford, "An architecture for scalable cloud data pipelines using AWS Lambda and S3," *IEEE Cloud Computing*, vol. 9, no. 3, pp. 45–53, 2022.
- [24] S. K. Roy and P. K. Paul, "Design of a real-time data ingestion framework using Apache Kafka and cloud storage for big data analytics," *IEEE Transactions on Big Data*, vol. 9, no. 1, pp. 112–121, 2023.
- [25] Z. Zhang, H. Li, and T. Sun, "Multi-modal fusion for stock prediction with financial news and numerical data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 5, pp. 2480–2491, 2023.