VILNIUS UNIVERSITY

FACULTY OF MATHEMATICS AND INFORMATICS

DATA SCIENCE STUDY PROGRAMME

Master's thesis

Sentiment and Financial Metrics Use for Stock Price Direction Prediction

Akcijų Kainos Krypties Prognozavimas Panaudojant Sentimentą ir Kainos Metrikas

Simonas Mikalkevičius

Supervisor : Doc. Dr. Dmitrij Celov

Vilnius
2026

Summary

The objective of the master's thesis is to create a machine learning model that predicts whether a stock price will exhibit bullish or bearish tendencies after the Quarterly Earnings Call release. With a tailored trading strategy, the model aims to generate a positive return, ideally surpassing that of a passive index buying strategy.

To achieve this, a dataset containing 33,362 earning call transcripts from 685 companies spanning the period from January 1, 2005, to March 31, 2025, was used for transcript and speaker sentiment extraction using the pre-trained BERT, FinBERT, and GPT-5-nano models. Sentiment data was enriched with stock-related data from the Yahoo Finance API before the Earnings Call to capture the quarter's price movement tendencies. Further data analysis, feature engineering, and data set cleaning were performed. To obtain robust results, a few model categories were tested using the same strategy: Lasso and Ridge regression, XGBoost, LightGBM, and Random Forest to identify which can better predict price movement.

The results indicate that earnings calls can serve as short-term catalyst events, and combining sentiment with financial metrics provides an additional predictive signal. While classification performance is low due to market noise, small predictive advantage and sector awareness yielded positive returns at times, beating the market index. Results revealed that increased model complexity does not improve performance, as regularised linear models, such as Ridge regression, delivered more robust results.

Keywords: sentiment extraction, classification, machine learning, earnings call, price movement prediction.

Santrauka

Magistrinio darbo tikslas yra sukurti modelį, gebanti prognozuoti akcijų kainos augimo arba kritimo kryptį po įmonės ketvirčio rezultatų pristatymo. Gautas modelis taikomas trumpalaikei pirkimo–pardavimo strategijai ir vertinamas pagal tai, ar jis generuoja teigiamą finansinę grąžą, lyginant su pasyvia investavimo į rinkos indeksą strategija.

Tyrime naudotas duomenų rinkinys, apimantis 33,362 ketvirčio rezultatus iš 685 įmonių nuo 2005 m. pradžios iki 2025 pirmojo ketvirčio pabaigos. Naudojant iš anksto apmokytus ”Bert” ir ”FinBert” modelius buvo išgautas pristatymo ir kiekvieno dalyvio teigiamas, neigiamas ir neutralus sentimentas. Naujai sudarytas duomenų rinkinys buvo papildytas ketvirčio akcijos kainos judėjimo metrikomis naudojantis viešą ”Yahoo Finance” duomenų bazę. Toliau buvo atlikta duomenų analizė, naujų kintamųjų sukūrimas bei duomenų paruošimas mašininiam mokymui. Siekiant gauti patikimus rezultatus buvo išbandytos kelios modelių kategorijos – regresijos (Lasso ir Ridge), ”XGBoost”, ”LightGBM” ir ”Random Forest” taikant vienodą strategiją

Gauti rezultatai parodė, kad ketvirčio rezultatų pristatymai turi trumpalaikį poveikį akcijų kainos krypčiai ir išgautas sentimentas kartu su finansiniais rodikliais suteikia papildomos informacijos. Kaip ir buvo tikėtasi, dėl rinką veikiančių įvairių veiksnių, klasifikacijos rezultatai nebuvo dideli. Tačiau kaip parodė strategijos simuliacijos rezultatai, net ir nedidelis pranašumas leido pasiekti teigiamą grąžą, tam tikrais atvejais pranokstančią rinkos indeksą. Rezultatai taip pat parodė, kad modelio kompleksiškumas nėra tolygus geresniems rezultatams, nes reguliarizuoti linijiniai modeliai, kaip ”Ridge” regresija, užtikrino stabilesnius rezultatus.

Raktiniai žodžiai: sentimento išgavimas, klasifikacija, mašininis mokymas, įmonių ketvirčio rezultatai, kainos krypties prognozavimas.

List of Figures

List of Tables

Contents

List of abbreviations

LLM	–	Large Language Model
BERT	–	Bidirectional Encoder Representation from Transformers
ML	–	Machine Learning
CAR	–	Cumulative Abnormal Return is a measure for excess return of a stock against an industry-specific index or broader portfolio
RF	–	Random Forest
BMO	–	Before Market Open
AMC	–	After Market Close
EC	–	Earnings Call
AUC	–	Area Under the Curve

Introduction

Financial markets are highly dynamic systems that receive and react to new information. While market efficiency suggests that the price of a product, in this case stock, should reflect available data, in reality, there are a large number of variables. Investor behaviour, information asymmetry, and various institutions can make short-term trading challenging, requiring the combination of multiple information sources instead of relying solely on asset prices.

One known approach to identify temporary inefficiencies is the event study, which analyses corporate events that introduce new information to the system. Such catalyst events can create short-term deviations from the equilibrium price, creating an opportunity to outperform the market. The effectiveness of this strategy depends not only on the event but also on how fast and accurately the information can be interpreted and used.

Earnings calls can be viewed as recurring catalyst events in financial markets, introducing new information which can change current price expectations for the short term. They are publicly available and regularly scheduled, containing quantitative financial disclosures with qualitative communication information. The last one carries additional information in terms of sentiment, tone, and narrative, which impacts investors' interpretation of the company's performance and prospects, shaping further price movement. Recent advances in LLM and ML enabled the extraction of this sentiment, creating an opportunity to incorporate textual sentiment into predictive modeling.

The main objectives of this thesis are twofold:

1. Examine if sentiment extracted from earnings call transcripts and combined with financial metrics can be used for short-term stock price direction prediction.

2. Evaluate through a simulated trading scenario if models trained on the created dataset can yield positive financial returns and compete with a passive market index trading strategy.

The main contribution of this thesis is the development of an empirical framework that uses multiple LLMs to create a structured sentiment and financial metrics-based dataset, enabling short-term stock price direction evaluation. By performing data analysis and comparing linear, ensemble and gradient-based classifiers using both statistical measures and trading simulations, the thesis provides practical insight into the usefulness of sentiment in event-based financial prediction.

The remainder of the thesis is structured as follows:

1. **Literature review**: Reviews existing research on earnings call sentiment, its extraction, and methods used to include it in financial prediction and trading models.

2. **Methodology**: Lists selected models and metrics for their performance evaluation.

3. **Analytical part** drafts empirical workflow and contains detailed overview of dataset, performed pre-processing procedures, data analysis, training and further evaluation results.

4. **Results and Conclusion**: Summarised thesis main findings and discusses its limitations and directions for future improvements.

1 Literature Review

Financial prediction relies on accurate interpretation of diverse and noisy data sources, which play a role in investment decisions, risk management, and academic research. While traditional market analysis focused primarily on quantitative indicators such as prices, volumes, and volatility, the majority of economically relevant information is communicated through unstructured text. Earnings calls are a prominent example of such data: they are publicly available, regularly occurring events in which firms disclose financial results alongside qualitative explanations, future plans, and set the tone for the entire event.

Historically, the qualitative aspect of earnings calls has presented difficulties for empirical models due to the lack of scalable text analysis methods, leading many studies to rely solely on financial variables. Recent advances in natural language processing and large language models enabled the extraction of sentiment and tone from financial text, allowing this information to be integrated into quantitative frameworks. As a result, current research examines various approaches that integrate textual sentiment with financial metrics to more accurately capture market reactions to earnings announcements.

1.1 Background of Earning Call and Its Sentiment

Earnings calls represent structured and informative forms of corporate communication. They typically occur shortly after the publication of quarterly financial results and consist of two main segments:

- the management presentation, where management frames the company's performance and its future outlook.
- the question-and-answer (Q&A) session with financial analysts, where analysts' questions and management answers often reveal additional information to the public.

Due to this, earnings calls have become a popular research focus area for extracting sentiment as an indicator of managerial tone and market expectations.

The language used during these calls often reflects more than just factual information. It conveys emotional and psychological cues such as optimism, uncertainty, and confidence. Studies have shown that textual communication in finance can reveal signals that influence investor perception beyond quantitative data. Management tone, choice of words, and style of communication can influence short-term price movements and long-term investor sentiment, particularly when numeric financial performance or economic situation is unclear [24].

Recent literature emphasises that the structure of earnings calls itself affects the type of sentiment expressed. For example, finance studies show that sentiment in financial language differs from everyday usage and that interpreting it requires context-aware models. They argue that large language models (LLMs) like BERT-base (FinBert, RoBertA) or generative (GPT, Llama) outperform traditional lexicon approaches because they process text contextually by evaluating each word in relation to its surrounding phrases rather than assigning fixed scores. This allows them to detect nuanced signals [10]. Additionally, these types of models learn semantic representation, which enables them

to distinguish between neutral terminology and actual negative sentiment. Their generative architecture can enable LLMs to capture shifts in tone across transcript sections, further enhancing their performance. Therefore, LLMs offer context-sensitive understanding of communication compared to lexicon-based sentiment models.

This structural understanding is reinforced in another research, where subjectivity within the Q&A segments of earnings calls was analysed using a six-dimensional framework [19]. By creating custom-labelled dataset, their approach measures polarity, certainty, subjectivity, specificity, emotional intensity, and engagement of opinion, creating a view of how managers and analysts communicate under different informational pressures. Changes in confidence, use of speculative or emotional language, and conversational interactivity were correlated with subsequent market reactions. This shows that the Q&A session carries informational value.

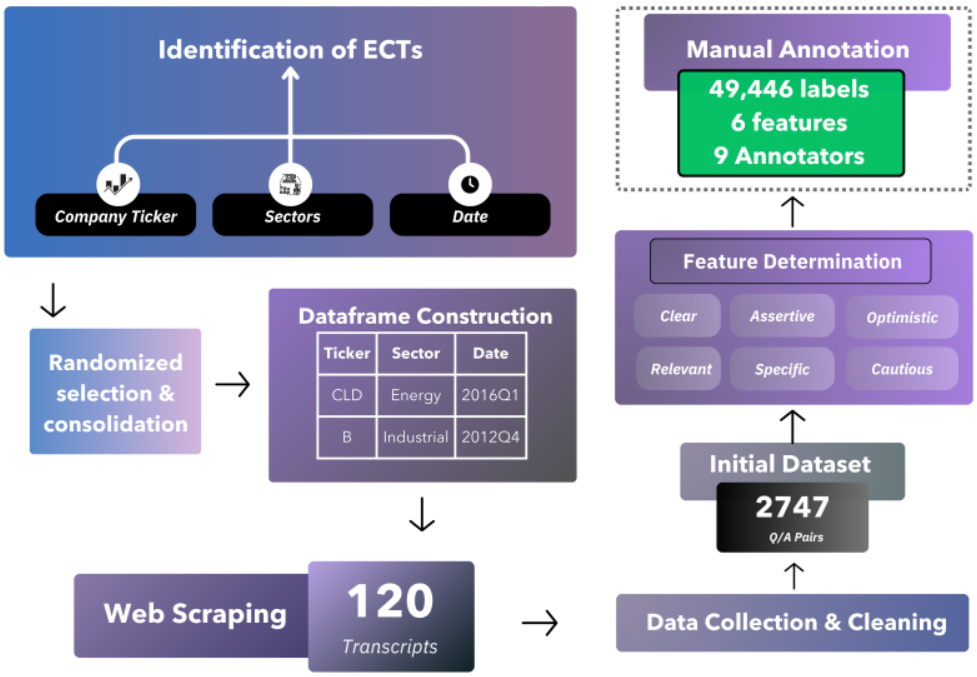

Figure 1: *Six-dimensional dataset overview for SubjECTive-QA model [19]*

At the same time, sentiment in corporate communication is not always a spontaneous reflection of management emotions. Firms can deliberately manipulate tone, selectively emphasising positive aspects or framing negative outcomes in softer language to influence market interpretation. This sentiment management dynamic creates challenges for sentiment extraction models, as signals of genuine managerial confidence can be inflated by strategically planned communication [2]. This cat-and-mouse dynamic underscores the importance of developing and refining domain-specific tools that can distinguish between stylistic tone and underlying informational content.

1.2 Trading strategies

Financial markets operate on the principle that prices reflect available information, yet behavioural and informational frictions often create temporary inefficiencies. Event-driven strategies

aim to capitalise on such inefficiencies by trading on identifiable corporate or macroeconomic events, such as earnings announcements, company acquisitions, mergers, or product launches. At those times, new information enters the market unexpectedly. Earnings calls serve as a recurring catalyst that combines quantitative disclosure with qualitative sentiment, creating short-term opportunities for directional trading.

Sentiment-driven trading is based on the idea that language and tone can serve as early indicators for investor reaction. Traditional strategies primarily rely on quantitative metrics and measures, such as price, volume, and volatility. However, textual and social sentiment add a behavioural dimension that may capture shifts in expectations before they are reflected in stock prices. It has been demonstrated that LLMs can extract sentiment from financial narratives and utilise it directly in trading frameworks. The results show that portfolios formed on LLM-derived sentiment signals outperform baseline benchmarks and that these effects persist even after controlling for common risk factors [8]. This supports the broader hypothesis that market participants miss information embedded in textual data.

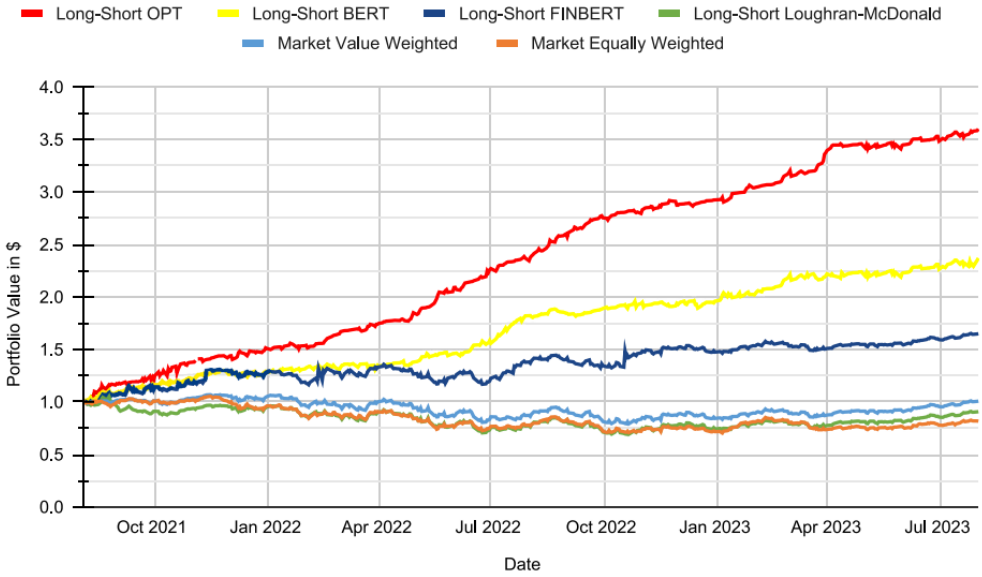

Figure 2: *Cumulative return graph showing how LLM models, like OpenAI – OPT, BERT, FinBERT, outperform traditional Loughran-McDonald dictionary and compare against market value.*

This can be further expanded by utilising social network posts and option trading data. By integrating media sentiment with option time and price volatility indicators, an improvement in predicting market price swings around key events was observed [17]. This reveals the importance that sentiment can inform not only directional bets (long or short) but also timing strategies, where the momentum rather than the direction of the move is traded.

Sentiment-based trading can be classified into three main categories:

- Directional – by taking long or short term position based on predicted sentiment values (positive or negative).
- Event timed – by entering trading position before or after expected catalyst (placing a trade before or during earnings call).

- Volatility-based – creating a position when the sentiment shows elevated uncertainty by comparing against implied and realised volatility differences.

Research suggests that returns from sentiment signals are often short-lived and concentrated around the event window. Therefore, potential gains are tied to precise timing and the application of transactional costs. Integrating sentiment with broader financial metrics, such as momentum or option-related data, enhances model robustness and reduces the impact of linguistic noise [8].

1.3 Sentiment extraction

The extraction of sentiment from financial text is a constantly developing field. Early approaches were based on dictionary-type methods, where each word was assigned a predefined class (positive, neutral, or negative), and sentiment was calculated as an aggregate score based on this classification [14]. Intuitive and computationally simple, these approaches suffered from fundamental limitations as they ignored context, domain meaning, and sentence structure. For example, common financial terms such as debt or capital loss can carry neutral or even positive meanings in specific contexts, leading to misinterpretation.

To address these challenges, researchers began working on domain-specific dictionaries tailored for financial communication. The development of sector-aware lexicons and dictionaries has shown that incorporating industry-specific terminology improves classification accuracy [13]. This allows for capturing language nuances particular to industries such as technology, healthcare, or finance, where words may signal different implications for company performance. This creates a need for more aware models that account for the heterogeneity of financial language across sectors.

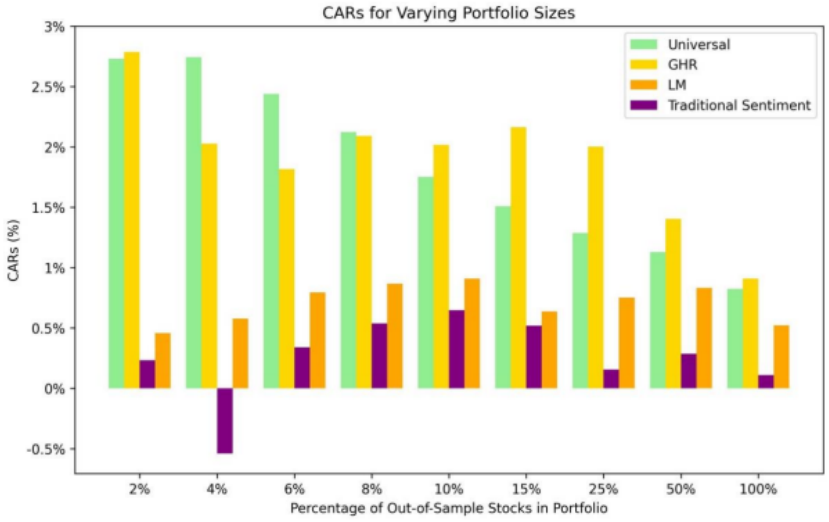

Figure 3: *Comparison of Cumulative Abnormal Returns (CAR) by using four dictionaries with increasing fractions of out-of-sample stocks. Universal: ML-based on industry-specific finance reports, GHR: ML-based on stock price reaction, LM: Loughran-McDonald dictionary, Traditional Sentiment: non-finance-based dictionary use*

A significant shift occurred with the introduction of machine learning text classifiers, specifically transformer-based models. They learn word and sentence representations from context rather than

relying on manually constructed dictionaries. One of the earlier examples of ML use was regression techniques and textual features to link earnings-call tone with stock-market reactions in the Korean market. Their approach demonstrates that even a relatively simple ML method outperformed static dictionary models when sentiment is embedded in contextually rich corporate communication [9].

With the emergence of pre-trained transformer models, sentiment extraction saw an improvement in its precision. FinBERT, an adaptation of BERT finetuned for the financial domain, is widely used for sentiment detection in earnings call transcripts and financial reports. Transformer models consider context bidirectionally, allowing them to detect subtle variations in meaning such as hedging, conditional optimism, or uncertainty. Recent advances improve this capability further by combining LLM reasoning with summarisation components. For instance, using LLM to retrieve specific information from the earnings call and generate its outline (retrieval-augmented generation, RAG) can improve interpretability and reduce associated hallucinations in financial domain applications [3].

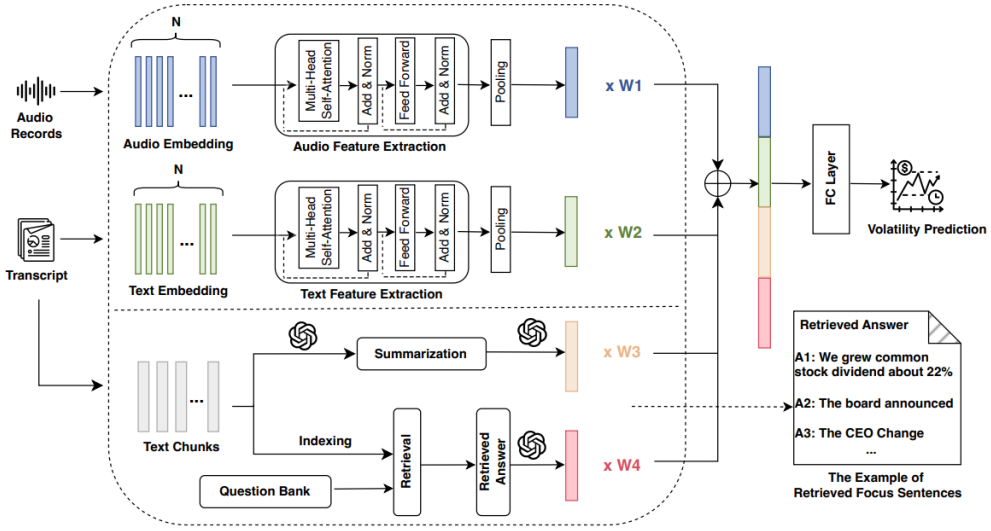

Figure 4: *Example of RAG use in a multi-model framework for volatility prediction.*

Further exploration of these tools led to the development and creation of various frameworks. A structured explanation (STRUX) allowed LLMs to provide not only sentiment classification but also interpretable reasoning paths behind their predictions [15]. Moving sentiment analysis from a predictive task towards explainable decision-making is an important win for risk management associated with trading.

Most recent research introduces agentic AI architectures that employ retrieval-enhanced LLMs to perform deeper multiple-document analysis of corporate communications. These systems are designed to extract and monitor recurring topics, tone dynamics, and emerging risk indicators across earnings calls from different firms and time periods, preserving contextual continuity between transcripts to capture changes in managerial tone and sentiment [1].

Although LLMs have achieved great results, they also face challenges in financial sentiment prediction due to their inherent low signal-to-noise ratio. Many linguistic or financial features that appear predictive in sample data fail to generalise because useful signals are weak and often over-

shadowed by market noise. This limitation has been emphasised in research showing that even advanced models require careful training procedures and feature regularisation to prevent overfitting. Extracting predictive financial information from noisy data depends less on model complexity and more on data design, robust evaluation, and methodology [22].

1.4 Modelling Approaches for Price Direction Prediction

Modelling stock price direction by the use of sentiment and financial indicators has progressed from traditional linear methods to deep learning and now to LLMs. Each model has its own balance for interpretability, predictive accuracy, and resource efficiency.

Early studies had applied logistic and linear regression models, where extracted sentiment was used in combination with financial metrics (traded volume, x-day return and volatility). These approaches were easy to understand in terms of how they work, but they struggled with nonlinear and contextual relationships between language and stock price behaviour. By comparing different types of models, it was shown that transformer-based models, such as FinBERT and GPTs, consistently outperform regression when textual data is mainly used, though at the cost of higher sensitivity to class imbalance and data pre-processing requirements [23].

Further work incorporated temporal modelling techniques by using architectures capable of learning sequential dependencies between linguistic tone and future price movements. A hybrid FinBERT-LSTM was created by combining contextual embeddings from FinBERT with a recurrent neural network layer for capturing the time series structure of sentiment and its delayed effect on market reaction. It was indicated that measurable accuracy improvements and robust performance across multiple time windows were achieved, but at the cost of increased computational requirements and a balanced dataset [20].

Fine-tuning transformer models by combining textual and quantitative signals can enhance performance. An example of this is the DeBERTa sentiment encoder, which is aligned with market-related metrics. This design strengthens the connection between linguistic tone and stock price, giving robust generalisation across the portfolio. However, it reduces the interpretability of how textual features contribute towards the final prediction [21].

With the emergence of LLMs, it was observed that their improved context awareness still had room for improvement in financial text interpretation. This leads towards domain-adapted LLMs training. FinLlama is a finetuned Llama model on a sector-specific sentiment dataset, which improves contextual reasoning and domain consistency similarly to how it was done to FinBERT. Its advantage is flexibility and semantic understanding at the cost of higher computation and sensitivity towards data preparation, which becomes challenging at real-time trading scenarios [11].

More recently, agentic AI frameworks have emerged which integrate multiple retrieval and reasoning components into a pipeline. These systems perform multiple tasks, such as extracting news article topics, tracking sentiment across corporate texts, and combining their findings for adaptive trading. This type of agent is able to adapt and respond to a changing narrative and risks in near real-time. While this type of system expands automation capabilities, its complexity introduces a lack of proper validation, stringent control requirements, and latency in processing new information [1].

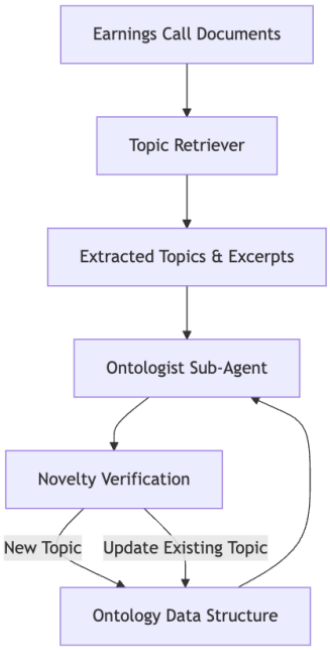

Figure 5: *Agentic system framework.*

In terms of practical application, different modelling families support distinct trading strategies. Regression-based models (linear, tree-based models) are typically used for directional trades, predicting upward or downward price movements based on sentiment score. Hybrid models as FinBERT-LSTM, are better suited for timing strategies, aligning trades with post-announcement drifts. LLMs and agentic systems give more complex volatility and event-type strategy applications, identifying uncertainty and information asymmetries that can guide trades. Across studies, sentiment-enhanced approaches deliver modest to significant improvements, which enhance prediction accuracy and lead to positive economic gains, even when risk and transactional costs are included.

1.5 Alternative Sources for Sentiment Application

Although most financial sentiment studies focus on structured corporate communication as earnings call transcripts, other forms of textual data have also been successfully used to infer market direction and volatility. These examples demonstrate how sentiment extraction from diverse data structures and domains with low signal-to-noise environments can support quantitative measures.

One example is social media data, which can be used for measuring market sentiment. Using public opinion from X and Reddit platforms can provide useful insights into short-term volatility or event-driven actions. By integrating the dataset with option trading metrics, such as premium changes, it was demonstrated that the model exhibited an improvement in volatility prediction [17]. This approach reveals how crowd sentiment for financial markets, where public investors' tones act as early indicators for uncertainty and risk. Despite its predictive value, data derived from social media is highly noisy, requiring additional transformation and filtering to prevent overfitting or false narratives.

Another example is sentiment in financial news headlines. By leveraging short text fragments

from news headlines with limited context, high ambiguity, and lower computational costs, the performance of large language models is currently being explored. These studies examine whether LLMs can accurately interpret tone and events, which could influence a company's stock price movement by not relying on the full article text. Findings reveal that transformer-type models outperform rule or neural network models by capturing tone cues in short news statements [16]. This demonstrates that sentiment modelling can be applied not only to long and structured transcripts but also to shorter textual data, such as headlines. It offers new opportunities for trading in daily and small events.

Beyond the financial sector, sentiment prediction modelling also finds applications in other industries, such as consumer analytics, energy markets, and logistics. In consumer analytics, sentiment is extracted from product or service reviews, which can be further used to predict purchasing behaviour, assess brand perception, and forecast demand. Transformer-based models have demonstrated good performance in predicting customer attitudes and trends [4]. For the energy sector, incorporating sentiment from business news as additional features for volatility prediction can enhance the model's generalisation in predicting better price fluctuations and reactions for oil and gas markets [7]. Similarly, logistics – specifically marine transport – can utilise sentiment for freight rate movement forecasting, providing an additional signal to traditional quantitative measures [5].

1.6 Summary of Literature Review

These studies establish earnings calls as recurring catalyst events that contain not only quantitative but also qualitative information, such as speaker tone, sentiment, and narrative, which can influence short-term stock price movements. Based on this, earnings calls are used as the primary event window and source of sentiment, serving as a potentially informative signal for predicting price direction.

Prior work also shows that sentiment varies greatly in earnings calls and is influenced not only by transcript segments but also by individual speakers. This observation is used to extract sentiment from the most frequent speakers and transcript sections rather than relying on a single aggregated transcript score. Additional features, such as slope and changes throughout the announcement, are tracked to capture tone dynamics and add additional signal to the dataset. For this task, models such as BERT, finance-specific RoBERTa, FinBERT, and the latest available GPT-5-nano with reasoning capabilities are used for sentiment extraction to test how their outputs differ and whether they can support stock price direction prediction.

The literature on sentiment-driven trading strategies, combined with weak signal learning, presents a range of regression, tree-based, and transformer-based models with varying bias trade-offs. As findings suggest that sentiment effects are short-lived in the stock market, the 1-day reaction after announcement is selected as a target feature in further analysis. To finalise key findings, model performance is evaluated not only using classification metrics but also by benchmarking the best performing model against a passive S&P500 index trading strategy.

2 Methodology

2.1 Sentiment Extraction Models

To extract sentiment from earnings call transcript, three transformer based models are used: BERT, FinBERT and GPT-5-nano. These models differ in their training data which creates distinct sentiment representation in general, finance and reasoning domain. Model outputs are pre-processed to ensure consistency and alignment with three class format: positive, neutral and negative.

2.1.1 BERT

BERT is a transformer model trained on large scale text collection and is used as general baseline. In this work, *nlptown/bert-base-multilingual-uncased-sentiment* model with standardised output is used. Although BERT is not trained specifically on financial text, it provides a benchmark to evaluate how much additional signal is gained from finance-tuned or reasoning sentiment models.

2.1.2 FinBERT

For finance specific sentiment extraction *yiyanghkust/finbert-tone* is used. FinBERT is finetuned on financial news and analyst reports. This allows the model to better handle and interpret financial terminology lowering sentiment misinterpretations from lack of domain awareness.

2.1.3 GPT-5-nano

GPT-5-nano differs from BERT as it is unidirectional and process text autoregressively, enabling context based reasoning when producing sentiment assessments. Together, these features and structured prompt are used in transcript sentiment score extraction. To evaluate if reasoning adds additional value, two versions of the model: reasoning enabled and disabled are tested.

2.2 Gradient Boosting

Gradient boosting is an ensemble learning technique that builds a strong classifier by combining multiple weak learners. Each iteration, a new model is trained to minimise loss function by correcting errors made by previous learner. Gradient boosting method is well suited for tabular data and can capture non-linear relationships between financial and sentiment features. Thesis uses two gradient boosting based models to capture possible complex interactions in the dataset.

2.2.1 XGBoost Classifier

Extreme Gradient Boosting or XGBoost is a regularised gradient boosting model with the following objective function:

$$f(\theta) = \sum_{i}^{n} l(y_i, \hat{y_i}) + \sum_{k}^{K} \Omega(f_k),$$

XGBoost is included for its ability to control overfitting though regularisation and performance in weak-signal environments. Hyperparameters such as tree depth, learning rate, subsample size, regularisation strength and number of estimators are tuned by cross-validation.

2.2.2 LightGBM Classifier

LightGBM differs from XGBoost in terms of tree growth strategy, it is performed leaf-wise instead of level-wise growth. This allows the model to improve performance with fewer trees by concentrating on higher residual error regions. Training is performed by picking similar hyperparameters as with XGBoost and its performance is evaluate further in this work to see if tree growth strategy yields any beneficial results.

2.3 Random Forest Classifier

Random forest is an ensemble machine learning method built of multiple decision trees trained on different subsets of data. Final prediction is obtained by aggregating outputs from individual trees, which improves stability, reduces variance and bias. Hyperparameters for RF are similar to gradient boosting models, controlling sample split by node and leaf, feature size, tree depth and estimators. RF performs well with noisy data and captures non-linear relations between features. However, as it tend to generalise patters, it may be less sensitive to weak and short-lived earnings call signals.

2.4 Lasso Regression

Lasso regression is a linear model with L1 regularisation which looks for sparse solutions by lowering some feature coefficients to zero. This allows model to perform feature selection and retain the strongest predictors. In this work, Lasso regression is used to evaluate if predictive information is concentrated within a small subset of features. However, with its L1 regularisation this model can omit weak predictors that in group provide useful information.

2.5 Ridge Regression

Ridge regression uses L2 regularisation which similar to Lasso shrinks coefficients towards zero, but retains all available features. This feature is particularly useful for weak, noisy financial data. By preserving small predictors, Ridge model achieves more stability and robust performance where signals are dispersed.

2.6 Performance Metrics

Model performance is evaluated using classification metrics, each depicting different predictive quality. As post-earnings price movements are noisy and weakly predictable, relying on a single metric would lead to inaccurate assessment.

2.6.1 Accuracy

Accuracy measures proportion of correctly classified observations:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

While accuracy provides an overview of model correctness in finance predictions, it can mislead in class imbalance. For this reason, accuracy is not treated as primary performance indicator in this thesis.

2.6.2 Precision

Precision measures a ratio of predicted positive outcomes that are true positives:

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

It reflects how often predicted stock movement corresponds to expected direction. High precision is important to reduce false positive trades, which would expose to more risk and financial losses.

2.6.3 Recall

Recall measures proportion of actual positives that are correctly classified:

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

Recall for positive price movement is important as missing upward moves results in lost opportunities. In event based strategy, correctly identifying price reaction direction is more important than avoiding every incorrect trade.

2.6.4 F1-Score

The f1-score measures a harmonic mean of precision and recall:

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

This provides a balanced evaluation when both positive and negative classes are important. In this work, f1-score is used to show classifier performance when trade-offs between opportunity capture and risk control are present.

2.6.5 AUC

The area under the received operating characteristic curve (AUC) measures how well model separates positive cases from negative ones across different thresholds. AUC is used as the optimisation metric during model training as it is well suited for probabilistic trading decisions.

2.6.6 Cross-Validation

Cross-validation is used in this work to ensure that model performance is not driven by secific data split. Five-fold cross-validation is used during finetuning to obtain stable performance metrics.

To prevent information leakage, training data is ordered chronologically rather than randomly shuffling as it reflects more realistic prediction conditions.

3 Analytical Part

The main objective of the master's thesis is to test whether combining textual sentiment features with quantitative indicators can improve predictive accuracy. This is achieved by creating a dataset containing the top three speaker-level sentiments (based on recency) derived from earnings call transcripts with the use of multiple LLM, along with each company's market data. The dataset is then used to train ML models for directional movement classification after the event.

To identify the relationship between speaker tone during earnings calls and the subsequent reaction in stock price movement, a workflow is designed to extract speaker sentiment and integrate it with financial data. The work is performed using the Python environment and public APIs to access available data and language models.

The following key components are defined:

- A transcript dataset of 33,362 earnings calls from 685 companies is obtained from Hugging Face.
- Text-based data requires cleaning before it is used by transformer models for positive, negative, and neutral sentiment extraction.
- Based on date and ticker, financial data is retrieved using the Yfinance API for each company to enrich the initial data.
- Feature engineering is performed to derive financial and sentiment metrics, summarising current data and providing additional predictive features.
- From exploratory data analysis results, further feature selection and additional required cleaning are applied before passing data to model training.

Finally, the prepared dataset is used to train classification models for predicting stock direction. It is important to evaluate model performance not only on key metrics (F1-score, precision, recall) but also through trading strategy simulation to assess practical application.

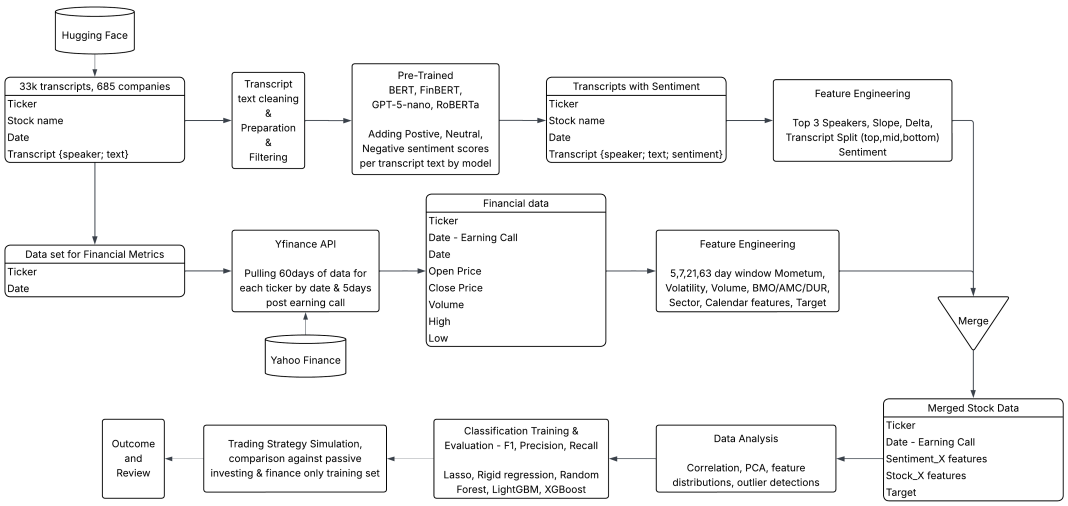

Figure 6: *Layout of Defined Workflow*

3.1 Work Environment

The research work is performed on a personal computer. Python is used to model in the Jupyter Notebook editor, which features a preset machine learning environment containing all the required libraries for this work, including pandas, yfinance, numpy, scikit-learn, xgboost, torch, and transformers. Model training is performed locally using an RTX 3060TI GPU, which reduces the dependency on setting up and using cloud services. For planning the structure of the datasets and workflows, the ”Lucidchart” application is used.

3.2 Data

Downloading quarterly earnings calls, by definition, should be an easy task—they are publicly open and accessible. The challenge lies in the large scope of data collection and locating each company’s website, making it a very manual-intensive process. Alternatives such as stock-related pages (The Motley Fool, Seeking Alpha, Market Gecko) offer easier navigation and stock data aggregation. Unfortunately, most of them have limits on how many earnings calls can be downloaded or require a premium subscription to access properly parsed transcripts. According to the literature overview, many researchers have developed custom web-scraping tools for this task. While this approach significantly reduces manual effort, pulling large amounts of data or placing frequent requests can strain website servers and may violate their terms of service.

In May 2025, a new and, to date, the largest parsed earnings call dataset was released on Hugging Face [12]. It contains 33,362 unique earnings calls of S&P 500 companies in JSON format, structuring each transcript by speakers and their text chronologically. This allows for speaker-level sentiment extraction. The entire dataset comprises 1.1 million speaker entries, approximately 200 million words, with a median of 6,500 words per transcript. It also includes additional data points, such as the company name, ticker symbol, and reporting period.

3.2.1 Pre-Processing

Before sentiment extraction, the transcript dataset requires pre-processing to remove redundant words. This is performed by splitting the transcript into individual words and counting their occurrences. The most frequent stop words, such as ”the”, ”a”, and ”and”, are manually removed to reduce noise and dimensionality. In addition, unique speakers from the dataset are reviewed for any incorrectly assigned or missing names. They are excluded from sentiment analysis because the pipeline is developed to produce sentiment for each speaker individually.

After cleaning the earnings call transcripts, they are parsed into speaker-level text and used for sentiment extraction for LLMs. For this task, three models were chosen:

- BERT – does not represent a finance-specific model and can be used as a baseline for other models.
- FinBERT – a fine-tuned BERT model designed for finance-related sentiment extraction. Picked as one of the most popular open source models available.

- GPT – by choosing to use advanced LLM from OpenAI, we are able to test further development impact on sentiment extraction by comparing to previously listed models.

A pipeline is developed to intake each speaker's text from the iterated transcript, run it through the model, and assign the resulting sentiment back to the initial dataset. However, this showed a few challenges. The first one is related to different model outputs for sentiment. The open-source BERT model had 5 different tones graded by stars. The higher the number of stars, the more positive the sentiment is and vice versa. To normalise it, both 2-star, 3-star, and 4-star ratings are used as a median result and are named as the ”normal” tone. The lowest and highest star ratings are accordingly ”negative” and ”positive”. On the other hand, FinBERT already came with 3 tone classes, meeting the requirement.

Another issue faced is that models have a token limit. As models differ in size and ability to load tokens, a solution was required. To resolve it, longer speaker texts are split into chunks which can be passed to the pipeline, and the median sentiment is derived as the final value for each speaker's text. To ensure further flexibility for feature engineering, all three types of sentiment (positive, neutral, and negative) are kept separate for each model.

As for the GPT-based model, sentiment extraction is performed using OpenAI's GPT-5 nano model. It is a lightweight version of the GPT-5 model optimised for summarisation and classification tasks, with the lowest available token price [18]. Comparing it against BERT and FinBERT models, which rely on contextual understanding, GPT-5 introduces a chain of thought feature. This is particularly helpful when large amounts of unstructured information are used, as the model can better handle complex relationships arising from text structure and narrative. This ability should make it well-suited for financial text analysis, where sentiment often appears indirectly.

Due to the large amount of data and the cautiousness of each token price, the dataset used for GPT sentiment extractions is restricted to three sectors: Technology, Healthcare, and Utility, and only includes the top three speakers for each earnings call. This selection enables integration of GPT sentiment with the previous model's outputs, captures most of the transcript text and manages inherited API costs. To further optimise API calls, transcripts are processed in batches of five to minimise request amount and fit within the model's token limit. To prevent an error between API calls for running out of available tokens, a timer function was implemented, which, upon an unsuccessful request would delay the next execution. In total, 9,888 transcripts are used, which require 12,000 requests, and 255 million tokens are received, corresponding to two versions of sentiment (with reasoning enabled and disabled).

A custom few-shot prompt (***Listing 3***) is used to ensure consistent outputs throughout the whole process. It consists of four parts: a brief overview stating the objective, describing the task, output and three examples for each category. This is passed to the model as instructions to output positive, neutral and negative sentiment scores which sum to 1.00. This allows to use GPT-5 nano LLM capabilities and reasoning ability. To further refine the instruction, a GPT-5.1 is used to expand, provide examples, and improve the overall structure, without losing the initial idea.

Initially, four transformer-type models are used, but only BERT, FinBERT and GPT-5-nano are selected. After three weeks of runtime on the local machine, it is observed that a pre-trained RoBERTa

model on financial documents fails to load its weights correctly, resulting in unreliable sentiment predictions. Therefore, the outputs of the RoBERTa model are dropped from the dataset to finalise the sentiment extraction part.

3.2.2 Feature Engineering

To enrich the transcript dataset, each earnings call is linked to its stock using the ticker and earnings timestamp. For every observation, an event window of 63 days is constructed around the earnings date, and daily price, volume history is retrieved from the yfinance API. In addition, firm level information such as sector and industry are included, together with a timing flag indicating when the earnings call took place: before market hours (BMO), after market hours (AMC) or during trading hours (DUR). It was noticed that the release timing of the information (as of the earnings call) can yield different results by the market on identical disclosures [6].

The reaction day, further noted in this work as a target, is defined as the first trading day on which the market can fully respond to the earnings call. When the call occurs, AMC, the reaction day is shifted to the next trading day, when calls BMO or DUR are assigned to the same trading day. Based on this, all financial features are computed strictly from the information available prior to the reaction day to avoid look-ahead bias and prevent overlapping of previous calls, isolating each as an independent event. Pre-earning market dynamics are summarised over multiple time windows (5, 7, 21, and 63 trading days), each representing a week, a week and a half, a month and a quarter of trading activity. Using multiple horizons provides both context for momentum and risk dynamics leading up to earnings announcements, as observed in other studies [25].

Feature	Group	Description
is BMO / AMC / DUR,	Identifiers	Variables for market status (open/closed)
volX, volX_over_volY, volX_zscore_vs_Y	Volume	Averages, ratio and z-scores for X/Y days
rv_before_X, rv_X_annual	RV	Realized volatility before X days
mom_before_X	Momentum	relative price change in X days
weekday, is_month_end, is_quarter_end	Calendar	Dummy variable if reaction date is quarter-end.
target_logret_1d	Target	1-day log return from earnings to reaction date.

Table 1: *Derived stock price & volume features*

The engineered financial feature set captures several complementary dimensions of market behaviour:

- Volume throughout the period, its ratio against different horizons and their z-score is included to give additional insight if abnormal trading interest is building before the earnings.
- Momentum is computed as a relative stock price change over each time period, providing a simple measure of direction prior to the event.
- Realised volatility is calculated in current and annualised form, showing the magnitude of uncertainty prior event during each horizon.
- Calendar variables as weekday, flags for month, and quarter end, are included to give more awareness for timing effects and reporting cycles.

As for transcript sentiment, additional features were added to preserve more information than a simple score aggregation. Since transcripts follow a consistent structure and involve distinct speaker roles, sentiment is aggregated by the top three frequent speakers and equal transcript segments to capture the introduction, presentation and Q&A sections. For each speaker and segment, their mean probabilities of neutral, positive and negative sentiment are derived. In addition to static sentiment, dynamic features (only for BERT and FinBERT models' outputs) were added as sentiment deltas and slopes to capture changes in tone direction. This step allows us to finalise feature engineering with an outcome of a rich dataset covering both static and dynamic sentiment and financial metrics.

3.2.3 Exploratory Data Analysis

The target variable, 1-day stock return after earnings call distribution, can be seen in ***Figure 7***. It shows a nearly symmetric shape with a peak at zero, indicating that nearly 60% of announcements' price changes are either priced in advance or investor reaction is lagged for a longer period. However, tails on both sides of the distribution tell that some high growth or uncertainty companies do experience abnormal returns in the range of ±10-40% range. During model training, it is important to properly label these groups as the difference between 5% or 40% could have an impact on the model's performance.

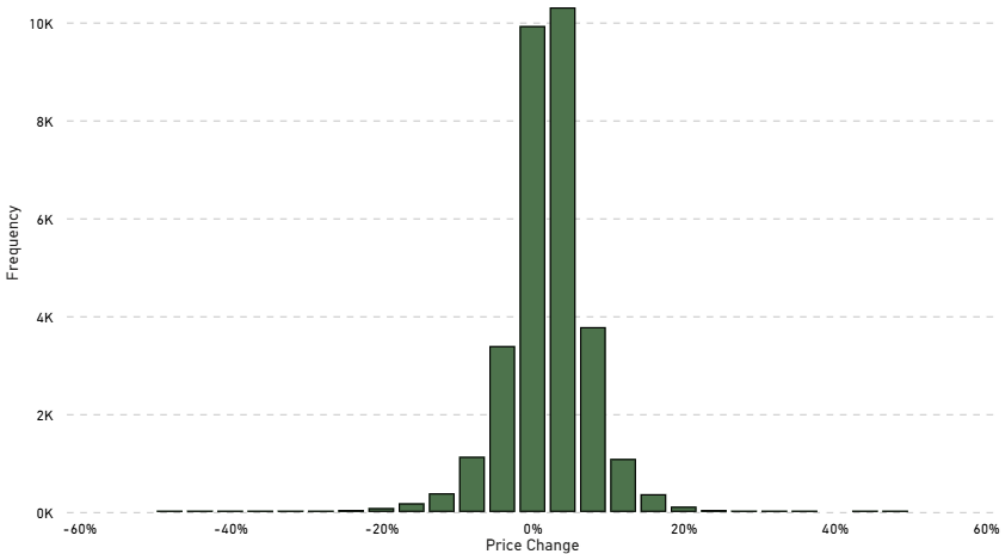

Figure 7: *Price Change of Stock Price (Before and 1-day After Earnings Call)*

Investor reaction for different industries can differ based on earnings announcements due to varying expectations. For example, slower growth in the Utility or Real Estate sectors can be easily tolerated due to their highly regulated environment. Additionally, the majority of projects that could significantly impact a company's image are announced years in advance, creating a predictable future outcome. This makes post-earnings call moves rare, as shown in ***Figure 8***, with a tighter interquartile range.

On the other hand, industries such as Technology and Consumer Cyclical experience high ambiguity. This can be due to high investor expectations, rapid innovation cycles and constant media

coverage. It brings higher reaction and sentiment volatility, which translates into more anomalous returns after 1 day. A new product release or a profitable quarter announcement can dramatically swing the stock price in either direction. For this reason, these sectors have the widest ranges and whiskers in ±10-15%.

These sector differences highlight that post-announcement reactions are not homogeneous across the market. Taking that into account, the sentiment impact could greatly vary by industry. A highly volatile sector can rely more on sentiment than on financial metrics and exhibit larger 1-day movements. In lower-volatility industries, sentiment may carry less predictive value. This heterogeneity indicates that sectors have an impact on 1-day price changes and will be required as a feature in model training to account for sector-specificities.

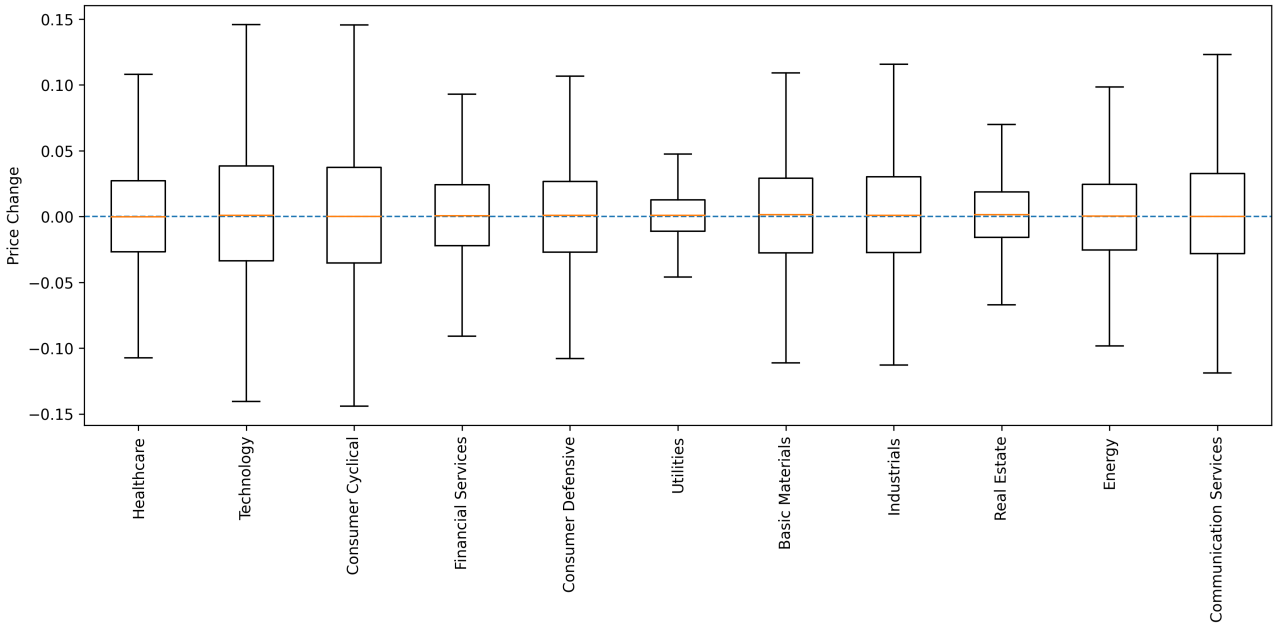

Figure 8: *Price Change of Stock Price by Sectors (Before and 1-day After Earnings Call)*

Sentiment descriptive statistics reveal a clear difference between the outputs of BERT, FinBERT, and GPT-5-nano. Across all speakers, FinBERT's neutral probabilities have the highest mean values, reflecting a conservative bias toward a neutral tone, with average probabilities across the three speakers at 60%. As the model is trained on financial news and documents, where sentiment is rarely extreme, it shows's tendency to predict vague or uncertain statements as neutral rather than positive or negative. This is clearly reflected in ***Figure 9***, showing its high average neutral sentiment score compared to other models.

The BERT model exhibits a more dispersed sentiment distribution across three classes. Because it is not fine-tuned on financial data, earnings calls are interpreted through a general domain view. This misclassify is led by financial terminology and vague phrasing to impact sentiment polarity. Results are more widespread across all three types, as can be seen in ***Figure 10***, where BERT exhibits minimal changes in sentiment for different speakers compared to other models.

As for the GPT-5-nano model, it is less neutral than FinBERT but also shows less variance than the BERT model. This difference could be attributed to an enhanced ability to interpret tone. Furthermore, comparing it with a version that has reasoning enabled shows a slight increase in both

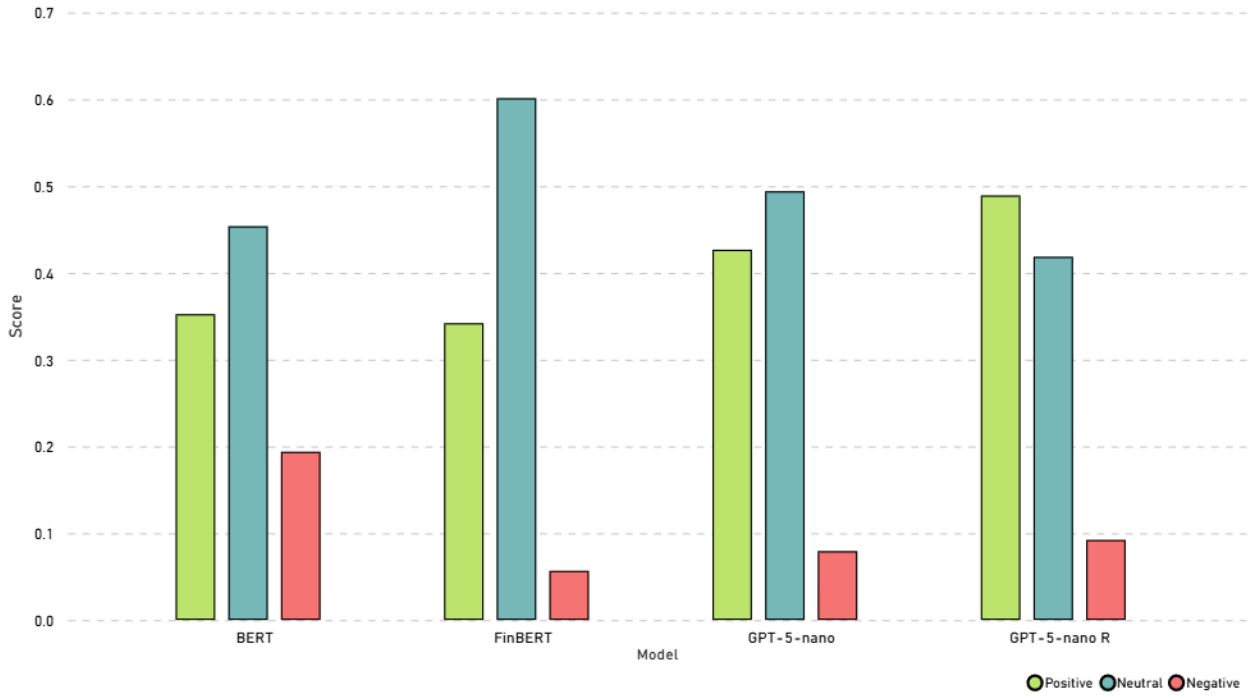

Figure 9: *Average Sentiment Scores Across Models*

positive and negative probabilities. It tells that a reasoning ability enables GPT-5-nano to interpret linguistic cues, giving it more sensitivity to tone dynamics in earnings calls.

By comparing the sentiment distribution across the top three speakers, a pattern emerges. Speaker 1, most likely a senior executive delivering prepared comments, shows the highest positive sentiment. This is due to a careful framing of the company's performance in an optimistic tone. Speaker 2, in most cases, will be a financial representative of the company, presenting a more neutral sentiment, as they tend to focus on quantitative results, metrics, and risks, which results in more balanced language. Last, Speaker 3 represents the analyst from Q&A session. As they tend to look into potential risks, uncertainties and weaknesses, their sentiment can be seen as more polar and having the most negative average outcome from all three speakers.

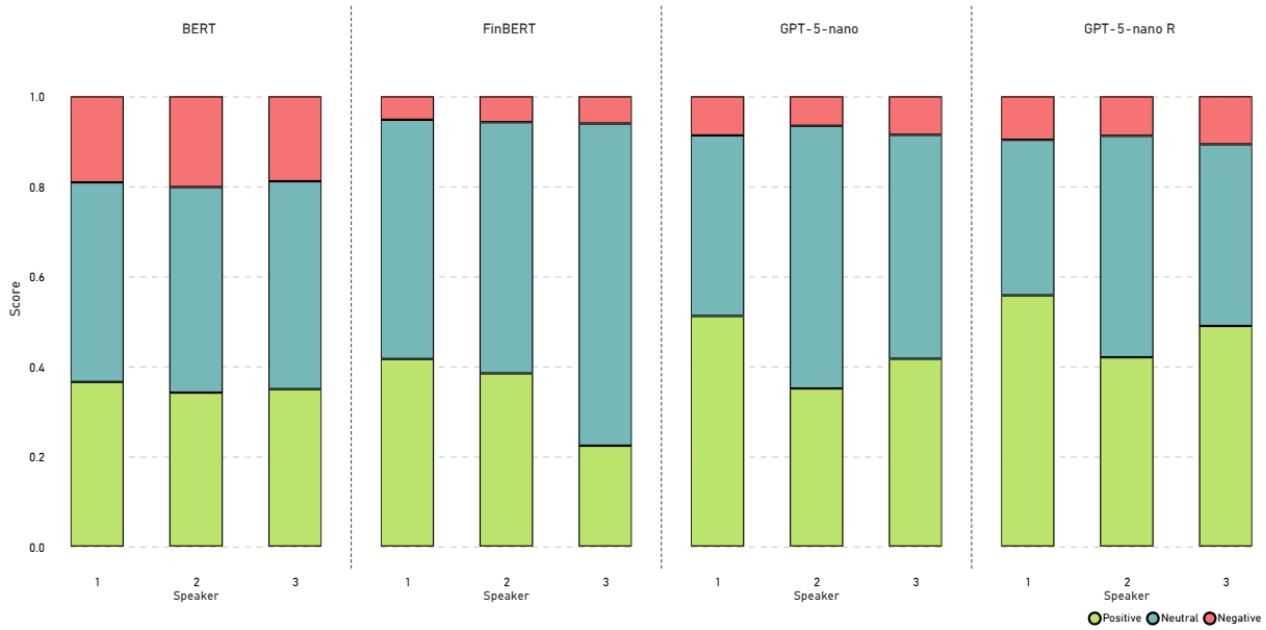

Figure 10: *Average Sentiment Scores Across Speakers by Model*

To identify feature variance and evaluate the dataset in reduced dimensionality, a principal component analysis (PCA) is performed. Its results show that variance is distributed across multiple components rather than being dominated by a single factor. The first principal component (PC) explains 12.8% of the total variance, and the first two PCs account for only 23.3% (***Table 2***). This gradual decrease in explained variance with each PC shows high dimensionality and heterogeneity of the dataset, as it combines finance metrics and linguistic sentiment. Such results are as expected, as information arrives from multiple sources with inherently weak signals.

PC	PC1	PC2	PC3	PC4	PC5
Var	12.8%	10.5%	5.0%	3.6%	3.2%
TTL Var	12.8%	23.3%	28.3%	31.9%	35.1%

Table 2: *PCA Variance by Component and Their Total Sum*

The composition of the first five PC reveals a clear separation between market and textual sentiment information. The first PC is primarily composed of pre-earnings uncertainty metrics, such as implied and realised volatility throughout different time horizons. This captures how uncertain or volatile investors perceive a firm before the new information from the earnings call is announced. The second PC is primarily composed of BERT and FinBERT sentiment features on the transcript's sections in both positive and neutral tones. These differences between PC1 and PC2 suggest that linguistic tone provides additional information, which is lacking in price-based measures.

Looking further to PC3-5, this observation is further supported. PC3 consists mainly of FinBERT features, which capture the first speaker's positive tone and the transcript's neutral and negative sentiments. This ties to the previously mentioned example of a company's representatives tending to express an overly positive tone during a presentation. With the highest loading scores from all models on speaker sentiment slopes and occurrences, PC4 describes how tone develops throughout the event and the extent to which different speakers influence the overall narrative. As for the final

PC5, it is primarily composed of GPT-5-nano (reasoning mode enabled) speaker-level sentiment, indicating that GPT captures higher-level contextual sentiment, which differs from the output of BERT and FinBERT models. Together, these components demonstrate that the current dataset is multidimensional and sentiment results are clearly dependent on the used models. As LLM outputs load on different components, integrating their sentiment with financial indicators not only derives a stronger context, but also complements each LLM.

3.3 Model

The literature review suggests selecting models that can account for the weak, noisy, and event-driven sentiment signals in financial markets. Studies show that sentiment extracted from earnings calls contains information relevant only to short-term price movements, but this information is spread across multiple textual and financial features. Moreover, sentiment effects are short-lived and tend to concentrate around the earnings announcement window, reinforcing the need for the model to extract weak signals without overfitting the noise. These findings suggest that no single modelling approach is optimal, and a comparative evaluation through different model families is needed.

Based on these insights, further work focuses on model classes commonly used in sentiment-based trading and weak signal prediction tasks. Linear models with regularisation are included as suitable benchmarks due to their simplicity and stability when signals are small or highly correlated. Two main models, Lasso and Ridge regression, allow to test if predictive information is wide or tightly distributed across features. Additionally, tree-based models from the ensemble class are used to capture nonlinear relationships, as sentiment can have different effects across sectors and speakers. Random Forest and gradient-based LightGBM, XGBoost models are widely used in event-based trading studies due to their ability to account for complex relationships and remain robust to noise.

3.3.1 Model Training & Review

Only the healthcare, technology, and utility sectors are selected for model training and evaluation. At first, the dataset is shuffled and split into 80% for training and 20% for testing. After observing stable training performance, the training split is refined by disabling shuffling, which prevents a model from learning patterns at similar time periods for the sector. In a final pre-training adjustment, the dataset is sorted chronologically prior to the train–test split. This resulted in a minor decrease in performance but prevented data leakage and ensured that models are tested on unseen future observations.

For XGBoost, LightGBM, and RF models, a parameter grid is created, and hyperparameters are tuned using a random search to minimise the ROC AUC scoring function. To have robust performance results, a 5-fold cross-validation is applied during training. For Lasso and Ridge regression, a grid search method is used to identify best best-suited regularisation strength. After model fitting, accuracy, precision, recall, f1-score and AUC are recorded, of which results can be found in (***Table 3***). For positive stock movement (class 1) metrics, such as precision and recall, are prioritised, as they reflect the model's ability to correctly identify bullish stock movements. For negative movement

(class 0), higher recall indicates how confident the model is in predicting bearish trends and limits investor exposure to the risk. As expected, due to the market's noisy nature, AUC stays low and no single model outperforms the others. Nevertheless, even small gains above randomness can yield beneficial results.

Model	Class	Accuracy	Precision	Recall	f1-score	AUC
XGBoost	0	0.576	0.61	0.43	0.50	0.600
	1		0.56	0.72	0.63	
LightGBM	0	0.575	0.61	0.40	0.48	0.595
	1		0.56	0.75	0.64	
Random Forest	0	0.572	0.61	0.38	0.47	0.602
	1		0.55	0.76	0.64	
Lasso	0	0.573	0.63	0.35	0.45	0.615
	1		0.55	0.79	0.65	
Ridge	0	0.577	0.59	0.48	0.53	0.602
	1		0.57	0.67	0.61	

Table 3: *Training Results of Finetuned Models*

Upon closer examination, some subtle tendencies can be observed relating to the model base and type. Tree-based models show high recall for class 1 but low recall for class 0, suggesting a bias toward predicting positive outcomes. This can be attributed to the general upward trend of the market, which leads to higher risk exposure. This is also apparent for Lasso, which, due to its sparse feature selection, focuses on momentum-driven features and results in a more bullish, higher-risk model. In contrast, Ridge regression provided more balanced performance, achieving the highest accuracy, recall, F1 score, and AUC. This improvement can be attributed to L2 regularisation, which allows for the retention of weak signal features. These findings indicate that increased model complexity does not lead to better performance, and a regularised linear model, such as Ridge regression, can be better suited for this task.

3.4 Implementation

It can be seen from the results that the models perform similarly to each other, and drawing a conclusion as to which one is a better model would be inaccurate based solely on classification metrics. However, from a finance perspective, they are missing one key measure – an economic payoff evaluation. To address this, model performance is further evaluated by using a trading simulation, where returns generated by each model are compared against a passive market investment strategy. As observed in the exploratory data analysis, each sector is treated separately in the simulation to reduce outcome variability. The simulation framework is set as follows:

1. There are two parties – (a) and (b).
2. Each party starts with 100$ capital and receives 100$ per quarter which:
    (a) Invests by buying shares of S&P500 index.

(b) Invests in specific sector – technology, healthcare, utility stocks and after a position is closed, funds become available for future trades.

3. Each party performs its trades:

   (a) In the middle of each quarter.

   (b) By buying shares on earnings call day based on the model's highest confidence that a stock price will increase and selling it on the next trading day.

4. (b) can perform only 3 trades per sector, quarter. Each trade consumes equal parts of (b)'s available balance. This minimises possible gains or losses compared to A.

5. Each model in (b) uses has same data and thresholds.

6. (a) holds its bought shares till the end of the simulation, and its score is the total return from the invested sum. While B's score is the cumulative return made by his 3 trades each quarter, per sector.

7. Trading period for both parties starts at 2022-07-01 till 2025-04-15. This period matches the available test sample's time.

8. Trading associated costs are excluded due to low number of transactions for both scenarios as it would net similar expenses.

Based on these rules, two functions are constructed. In (a) case, historical S&P 500 closing price data is downloaded by using the yfinance API, buy dates are identified and based on them, a quarterly DCA is calculated. As for (b), the test sample is passed through each finetuned model to obtain probabilities for each earnings call. Only top three predictions over a 0.52 threshold are selected as buy candidates within each sector. Strategy returns are then computed by simulating quarterly capital increases and compounding gains or losses into the rolling portfolio value. Finally, results from both strategies are merged, and the final evaluation dataset is prepared.

3.4.1 Result Evaluation

Starting from the (a) results, the S&P 500 index started negative due to by post covid inflation and interest rate increase. At the end of 2022 Q4, inflation appeared to have peaked, leading to expectations for future rate cuts, which created a modest recovery for the index, as shown in ***Figure 12***. During 2023, a rapid expansion of artificial intelligence added an additional 5% to index performance up to the final interest rate hike announcement in 2023 Q3, which triggered a market correction. In its fallout, AI-oriented companies like Amazon, Google, Microsoft, Meta and Nvidia continued to lift the index until the start of 2025, when U.S. President Donald J. Trump invoked tariffs negatively impacting market sentiment, leading to an index decline. Based on the events through this period, (a) invested 1200$ and accumulated 1404.33$, translating to 17.03% DCA.

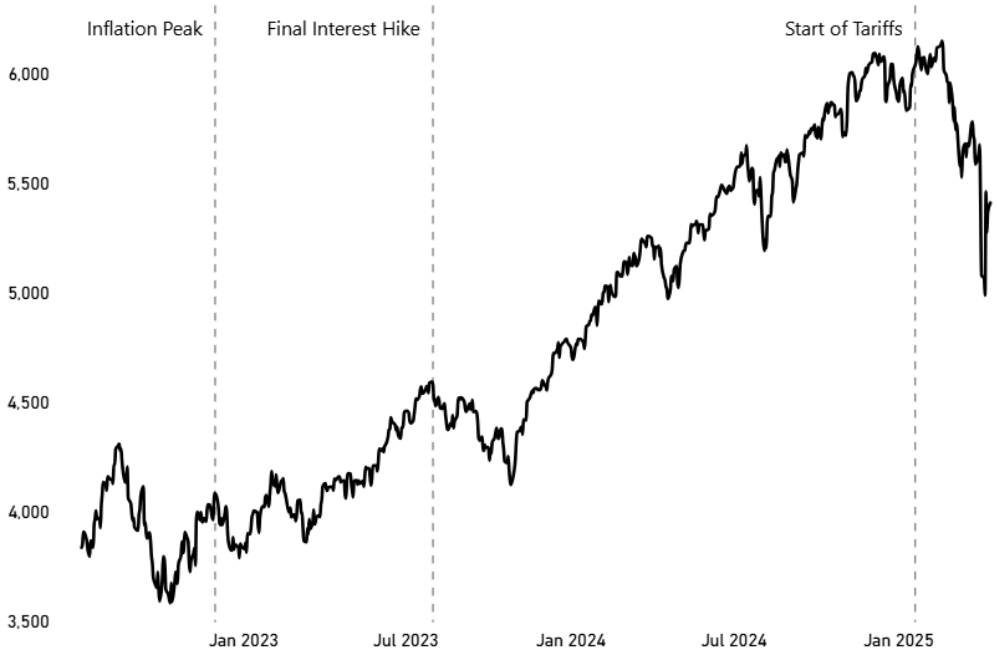

Figure 11: *S&P500 Index Price Movement*

Moving to model performance, great variance is seen in how they handle each sector. In the technology sector, XGBoost and Random Forest closely followed index movements, while Lasso underperformed compared to the benchmark. Only Ridge achieved returns noticeably beating the index. However, each model experienced a drop at the start of 2025, which aligns with tariff-related uncertainty and more negative sentiment during earnings calls. Within this sector, all models delivered returns higher than the index. In the healthcare sector, Random Forest returns remained relatively flat throughout the simulation period, while the rest of the models experienced steady growth. Lasso had the strongest performance in this sector, outperforming the index by more than 9% and exceeding its own performance in the technology sector. In contrast, utilities showed the weakest overall performance. All models generated negative returns initially, which reversed only in the second half of 2024. Within this sector, LightGBM and Ridge showed the best recovery compared to other models.

Sector	XGBoost	LightGBM	RF	Lasso	Ridge	Average
Healthcare	17.96%	11.82%	7.26%	28.12%	19.29%	16.89%
Technology	20.67%	18.23%	24.22%	18.47%	31.82%	22.68%
Utilities	3.53%	8.80%	3.18%	2.44%	9.82%	5.55%

Table 4: *Simulation Returns By Sector And Model (Financial + Sentiment Features)*

Differences for each sector are explained based on previous observation in the post-earnings price change distribution. Technology had the highest one, which explains its higher return results. Meanwhile, the utility sector had the smallest differences; thus, expecting this sector to yield high returns is very unlikely. From a modelling perspective, tree-based models effectively captured broad sentiment and financial trends, resulting in performance that closely tracked the market index. LightGBM achieved improvements in technology and utilities, likely due to its leaf-wise growth, allowing the model to capture more sector-specific changes. By focusing on a limited subset of the

strongest signal features and achieving the highest recall for class 1, Lasso adopted a more bullish and higher-risk position, which performs well in stable and growth periods. In contrast, Ridge regression achieved one of the better overall results due to its L2 regularisation, which greatly managed weak financial and sentiment-related signals, leading to more robust performance.

Sector	XGBoost	LightGBM	RF	Lasso	Ridge	Average
Healthcare	-0.46%	1.85%	8.33%	-5.82%	-3.28%	0.12%
Technology	3.30%	-21.46%	-1.22%	6.40%	6.41%	-1.31%
Utilities	-4.81%	1.27%	6.88%	8.64%	7.48%	3.89%

Table 5: *Simulation Returns By Sector And Model (Only Financial Features)*

Sector	XGBoost	LightGBM	RF	Lasso	Ridge
Healthcare	18.42%	9.97%	-1.07%	33.94%	22.57%
Technology	17.37%	39.69%	25.44%	12.07%	25.41%
Utilities	8.34%	7.53%	-3.70%	-6.20%	2.34%
Mean $\triangle$ AUC	0.100	0.096	0.091	0.108	0.103

Table 6: *Differences In Simulation Returns With And Without Sentiment Features*

To isolate sentiment contribution to improved returns, all models were trained using only financial features, setting a baseline for comparison. As seen in ***Table 5*** and ***Table 6***, use of sentiment leads to improvements in most sectors, specifically in healthcare and technology, where higher post-earnings reactions were observed. For example, in healthcare sector Lasso model improved from -5.82% to 28.12% and in technology LightGBM increased its returns from -21.46% to 18.23%. However, utility sectors show smaller gains, reflecting sector predictability and weaker investor reaction after announcement. Additionally, inclusion of sentiment features results in positive mean $\triangle$AUC for all models, derived as mean quarterly differences between AUC score from full feature and finance-only models. These results together with PCA show that sentiment adds meaningful signal as can be seen in sectors with higher uncertainty, where models with sentiment outperform finance-only baselines.

Figure 12: *Simulation Results (Finance + Sentiment Features)*

Results and Conclusions

This thesis confirms that earnings calls act as short-term catalyst events in financial markets. As publicly available and regularly scheduled company disclosures, they introduce new information that can shift investor expectations and price dynamics. Data analysis supports this by showing a presence of stock movement during this short one-day window, making earnings calls suitable for a short-term prediction task. While quantitative information is often already known and expectations for it are set, the qualitative features of it, such as tone and narrative, can provide additional context and influence post-announcement price movements.

Thus integration of sentiment features with financial metrics proves to be important for capturing these effects. PCA revealed a clear separation between volatility and sentiment components, indicating that textual information contributes and provides additional signals. Speaker-level and section-based sentiment further enriches it by acknowledging participants and the event's structure. Together, these results show that sentiment features in combination with financial metrics enhance available information for predictive tasks, even in a weak signal environment.

Despite the low classification performance, as expected for this type of task, the trading simulation demonstrated that even small predictive power can yield positive economic returns if properly leveraged. Final results also show that increased model complexity does not yield better performance. Regularised linear models, like Ridge regression, showed more balanced classification metrics and by retaining low signal features, demonstrated great results. This suggests that proper feature preparation, data awareness, such as sector-based returns, and proper regularisation are more important than model complexity in direction prediction.

This thesis leaves ample room for future improvements that could enhance the strategy's return. First, a finer sector or similar industry analysis, clustering individual stocks by type and return spread, would allow us to identify the highest netting target group for possible trades. Secondly, the exploration of a finetuned sector-aware LLM for sentiment extraction and its grouping into speaker roles, rather than individuals, could provide additional or stronger signals. Lastly, training the model only on one sector could yield better performance by focusing on key features. An example is the Lasso model and its application in the healthcare sector.

References and Sources

[1] G. G. Anant Gupta Rajarshi Bhowmik. “Agentic Retrieval of Topics and Insights from Earnings Calls.” In: (2025). URL: https://doi.org/10.48550/arXiv.2507.07906.

[2] J. Berkovitch, D. Israeli, R. Kasznik. “Sentiment Management: AI-based Evidence from Earings Guidance.” In: *S&P Global Market Intelligence* (2025). URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5051800.

[3] Y. Cao, Z. Chen, Q. Pei, N. J. Lee, K. P. Subbalakshmi, P. M. Ndiaye. “ECC Analyzer: Extract Trading Signal from Earnings Conference Calls using Large Language Model for Stock Volatility Prediction.” In: (2024). URL: https://arxiv.org/abs/2404.18470v2.

[4] A. Daza, N. Daniel, M. Sonia, W. Filomeno, M. Elena. “Sentiment Analysis on E-Commerce Product Reviews Using Machine Learning and Deep Learning Algorithms: A Bibliometric Analysis, Systematic Literature Review, Challenges and Future Works.” In: *International Journal of Information Management Data Insights, 2024, Volume 4, Issue 2* (2024). URL: https://doi.org/10.1016/j.jjimei.2024.100267.

[5] Y. Gong, Y. Peng, L. Xu, K. Chen, W. Shi. “Shipping news sentiment as a predictor of iron ore freight rates: Hybrid evidence from lexicon-based analysis and threshold autoregression modelling.” In: (2025). URL: https://doi.org/10.1016/j.tranpol.2025.05.003.

[6] D. Yu, Z. Liu, H. He. “Same Company, Same Signal: The Role of Identity in Earnings Call Transcripts.” In: (2024). URL: https://arxiv.org/pdf/2412.18029.

[7] F. Jawadi, D. Bourghelle, P. Rozin, A. I. Cheffou, G. S. Uddin. “Sentiment and energy price volatility: A nonlinear high frequency analysis.” In: *Energy Economics, 2024, Volume 133* (2024). URL: https://doi.org/10.1016/j.eneco.2024.107465.

[8] G. G. Kemal Kirtac. “Sentiment trading with large language models.” In: *Finance Research Letters, Volume 62, Part B* (2024). URL: https://doi.org/10.1016/j.frl.2024.105227.

[9] E.-C. Kim, N. Y. Lee, J. Jang. “Market Reactions to Earnings Call Transcripts Using Textual Analysis, Evidence from Korea Market.” In: (2025). URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5079964.

[10] K. Kirtac, G. Germano. “Large language models in finance: what is financial sentiment?” In: (2025). URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5166656.

[11] T. Konstantinidis, G. Iacovides, M. Xu, T. G. Constantinides, D. Mandic. “FinLlama: Financial Sentiment Classification for Algorithmic Trading Applications.” In: (2024). URL: https://arxiv.org/abs/2403.12285.

[12] kurry. *Hugging Face*. URL: https://huggingface.co/datasets/kurry/sp500_earnings_transcripts (viewed 2025-09-23).

[13] J. Lees. “Predicting Stock Prices: A New Approach to ML-Driven Sentiment Analysis.” University of Technology Sydney, 2023.

[14] T. Loughran, B. McDonald. “When Is a Liability Not a Liability? Textual Analysis, Dictionaries, and 10-Ks.” In: *Journal of Finance* (2010). URL: https://ssrn.com/abstract=1331573.

[15] Y. Lu, Y. Hu, H. Foroosh, W. Jin, F. Liu. “STRUX: An LLM for Decision-Making with Structured Explanations.” In: (2025). URL: https://arxiv.org/pdf/2410.12583.

[16] M. Muhammad I.; Rospocher. “On Assessing the Performance of LLMs for Target-Level Sentiment Analysis in Financial News Headlines.” In: *Algorithms 2025, 18, 46* (2025). URL: https://doi.org/10.3390/a18010046.

[17] S. V. Mulakala, U. Vangapally, B. Larkey, A. Henrichs, C. Wojslaw. “Analyzing Public Sentiment to Gauge Key Stock Events and Determine Volatility in Conjunction with Time and Options Premiums.” In: (2025). URL: https://arxiv.org/html/2502.05403v1.

[18] OpenAI. *Open AI GPT-5 nano Model*. URL: https://platform.openai.com/docs/models/gpt-5-nano (viewed 2025-11-23).

[19] H. Pardawala, S. Sukhani, A. Shah, V. Kejriwal, et al. “SubjECTive-QA: Measuring Subjectivity in Earnings Call Transcripts’ QA Through Six Dimensional Feature Analysis.” In: (2025). URL: https://doi.org/10.48550/arXiv.2410.20651.

[20] B. Sarkar, A. Shahid. “Hybrid FinBERT-LSTM Deep Learning Framework for Stock Price Prediction: a Sentiment Analysis Approach Using Earnings Call Transcripts.” In: (2025). URL: https://easychair.org/publications/preprint/FvJX.

[21] J. Schuettler, F. Audrino, F. Sigrist. “Does sentiment help in asset pricing? A novel approach using large language models and market-based labels.” In: (2025). URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4905533.

[22] Z. Shen, D. Xiu. “Can Machines Learn Weak Signals?” In: (2025). URL: https://www.nber.org/papers/w33421.

[23] O. Shobayo, S. Adeyemi-Longe, O. Popoola, B. Ogunleye. “Innovative Sentiment Analysis and Prediction of Stock Price Using FinBERT, GPT-4 and Logistic Regression: A Data-Driven Approach.” In: *Big Data and Cognitive Computing* (2024). URL: https://www.mdpi.com/2504-2289/8/11/143.

[24] H. T.H.S., A. Kalyani, S. M., A. Tahoun, L. van Lent. “Text as Data in Economic Analysis.” In: (2025). URL: https://doi.org/10.20955/wp.2024.022.

[25] Y. Zhang. “The Impact of Earning Press Releases Sentiment on Abnormal Stock Returns and Volatility.” PhD thesis. School of Business, University of Bristol, Bristol, UK, 2024.

Appendix 1. Software and Tools Utilized

During thesis development, the following applications were used:

1. Execution of code was performed using the Python programming language in the ”Jupyter Notebook” editor.

2. To visualise derived results and explore relationships between variables, the ”Microsoft Power BI” application was used. Use of it allowed for visually inspect sectors and accordingly plan exploratory data analysis.

3. ”Lucidchart” was used for initial thesis workflow planning, structuring and merging input tables.

4. To aid in resolving Python code errors, improving code coherence, ensuring academic tone and text flow clarity, the ChatGPT-5.1 tool was used.

   (a) For code-related issues prompt template was used:

   ```
   "I am trying to resolve the following issue (error received). Im
   working on X to achieve Y, here is my current code Z"
   ```

   (b) Text rephrasing template:

   ```
   "You are an academic reviewer with a task to refine my shared text
   snippet. I need you to improve its clarity, fix any grammar mistakes
   and reformat it to suit an academic tone. Keep my idea and original
   text structure, do not rewrite it or change context. Here is my
   current text: X"
   ```

All inputs generated by ChatGPT were reviewed by the thesis author before their use in this work. In cases when the generated output did not follow requirements or logical flaws were observed, clarifying prompts were used. To limit ChatGPT’s context leakage and limit the risk of hallucinations, new chat sessions were used per topic.

Appendix 2. Python Code

```
# As a first step sector and industry is added to initial dataset
tickers = list(df['symbol'].unique())
for t in tickers:
    stock = yf.Ticker(t)
    info = stock.info
    sector_dic[t] = info.get('sector')
    industry_dic[t] = info.get('industry')
df['sector'] = df['symbol'].map(sector_dic).fillna('Unknown')
df['industry'] = df['symbol'].map(industry_dic).fillna('Unknown')

# Helper functions created
def _safe_div(a, b):
    try:
        return a / b if b not in (None, 0, np.nan) else np.nan
    except Exception:
        return np.nan
def _days_to_years(d):
    return d / 365.0

# Main finance feature function
def finance_metrics(ticker: str, earnings_dt: Union[str, pd.Timestamp]) -> pd.
    DataFrame:
    earnings_dt = pd.to_datetime(earnings_dt)
    window = 63
    annualize = 252
    iv_target_days = 30

    start = (earnings_dt - pd.Timedelta(days=window * 3)).strftime('%Y-%m-%d')
    end = (earnings_dt + pd.Timedelta(days=window * 2)).strftime('%Y-%m-%d')

    data = yf.download(ticker, start=start, end=end, progress=False, auto_adjust
    =False)
    if data.empty:
        return pd.DataFrame([{'ticker': ticker, 'earnings_date': earnings_dt, '
    error': 'no price data'}])
    if isinstance(data.columns, pd.MultiIndex):
        data.columns = [' '.join([str(c) for c in col]).strip() for col in data.
    columns.values]

    cols = list(data.columns)
    adj_col = _find_col_contains(cols, ['adj', 'adjust']) or _find_col_contains(
    cols, ['close'])
    high_col = _find_col_contains(cols, ['high'])
    low_col = _find_col_contains(cols, ['low'])
    open_col = _find_col_contains(cols, ['open'])
    vol_col = _find_col_contains(cols, ['volume', 'vol'])
```

```
    close_col = _find_col_contains(cols, ['close'])

    if adj_col is None and close_col is None:
        return pd.DataFrame([{'ticker': ticker, 'earnings_date': earnings_dt, '
error': 'no data', '': cols}])

    df = data.copy()
    if adj_col is not None:
        df['Adj Close'] = pd.to_numeric(df[adj_col], errors='coerce')
    elif close_col is not None:
        df['Adj Close'] = pd.to_numeric(df[close_col], errors='coerce')
    if high_col is not None:
        df['High'] = pd.to_numeric(df[high_col], errors='coerce')
    if low_col is not None:
        df['Low'] = pd.to_numeric(df[low_col], errors='coerce')
    if open_col is not None:
        df['Open'] = pd.to_numeric(df[open_col], errors='coerce')
    if close_col is not None:
        df['Close'] = pd.to_numeric(df[close_col], errors='coerce')
    if vol_col is not None:
        df['Volume'] = pd.to_numeric(df[vol_col], errors='coerce')

    df = df.sort_index()
    df['log_ret'] = np.log(df['Adj Close']).diff()
    df['returns'] = df['Adj Close'].pct_change()
    df['prev_close'] = df['Adj Close'].shift(1)

    high = df['High'] if 'High' in df.columns else pd.Series(np.nan, index=df.
index)
    low = df['Low'] if 'Low' in df.columns else pd.Series(np.nan, index=df.index
)
    prev = df['prev_close']
    high = pd.to_numeric(high.squeeze(), errors='coerce')
    low = pd.to_numeric(low.squeeze(), errors='coerce')
    prev = pd.to_numeric(prev.squeeze(), errors='coerce')

    df['tr1'] = (high - low)
    df['tr2'] = (high - prev).abs()
    df['tr3'] = (low - prev).abs()
    df['true_range'] = df[['tr1', 'tr2', 'tr3']].max(axis=1)

    hour, minute = earnings_dt.hour, earnings_dt.minute
    if (hour < 9) or (hour == 9 and minute < 30):
        timing = 'BMO'
    elif hour >= 16:
        timing = 'AMC'
    else:
        timing = 'DUR'

```

```
    trading_days = df.index.normalize()
    if timing == 'AMC':
        cand = trading_days[trading_days > earnings_dt.normalize()]
    else:
        cand = trading_days[trading_days >= earnings_dt.normalize()]
    if len(cand) == 0:
        return pd.DataFrame([{'ticker': ticker, 'earnings_date': earnings_dt, '
error': 'no trading day found after earnings'}])
    reaction_day = cand[0]
    reaction_pos = next(i for i, d in enumerate(df.index.normalize()) if d >=
reaction_day)
    before = df.iloc[:reaction_pos]
    after = df.iloc[reaction_pos:]

    f = {'ticker': ticker,
        'earnings_datetime': earnings_dt,
        'reaction_date': pd.Timestamp(reaction_day).to_pydatetime(),
        'earnings_timing': timing,
        'is_BMO': int(timing == 'BMO'),
        'is_AMC': int(timing == 'AMC'),
        'is_DUR': int(timing == 'm'),
        'days_of_history': len(before)}

    windows = [5, 7, 21, 63]
    for w in windows:
        tail = before['log_ret'].tail(w).dropna() if len(before) >= 1 else pd.
Series(dtype=float)
        f[f'rv_before_{w}d_raw'] = float(tail.std()) if tail.shape[0] >= 1 else
np.nan
        f[f'rv_before_{w}d_annual'] = float(tail.std() * np.sqrt(annualize)) if
tail.shape[0] >= 1 else np.nan
        if len(before) >= 1:
            n_back = min(w, len(before))
            f[f'mom_before_{w}d'] = float(before['Adj Close'].iloc[-1] / before[
'Adj Close'].iloc[-n_back] - 1.0) if n_back >= 1 else np.nan
        else:
            f[f'mom_before_{w}d'] = np.nann

    if 'Volume' in before.columns and len(before) >= 5:
        vol5 = before['Volume'].tail(5).mean()
        vol63 = before['Volume'].tail(63).mean() if len(before) >= 21 else
before['Volume'].mean()
        vol63_std = before['Volume'].tail(63).std() if len(before) >= 21 else
before['Volume'].std()
        f['vol5'] = float(vol5)
        f['vol63'] = float(vol63)
        f['vol5_over_vol63'] = _safe_div(vol5, vol63)
        f['vol5_zscore_vs_63'] = _safe_div((vol5 - vol63), vol63_std)
    else:
```

```
        f['vol5'] = np.nan; f['vol63'] = np.nan; f['vol5_over_vol63'] = np.nan; f['vol5_zscore_vs_63'] = np.nan

    f['weekday'] = int(pd.Timestamp(earnings_dt).weekday())
    f['is_month_end'] = int(pd.Timestamp(earnings_dt).is_month_end)
    f['is_quarter_end'] = int(pd.Timestamp(earnings_dt).is_quarter_end)

    if after.shape[0] >= 1 and len(before) >= 1:
        close_before = before['Adj Close'].iloc[-1]
        close_after_1 = after['Adj Close'].iloc[0] if after.shape[0] >= 1 else np.nan
        logret_1d = np.log(close_after_1 / close_before) if pd.notna(close_before) and pd.notna(close_after_1) else np.nan
        f['target_logret_1d'] = float(logret_1d) if not pd.isna(logret_1d) else np.nan
        f['target_abs_logret_1d'] = float(abs(logret_1d)) if not pd.isna(logret_1d) else np.nan
    else:
        f['target_logret_1d'] = np.nan; f['target_abs_logret_1d'] = np.nan

    return pd.DataFrame([f])

# Placeholder dataset to which results are appended
cols = ['ticker', 'earnings_datetime', 'reaction_date', 'earnings_timing',
        'is_BMO', 'is_AMC', 'is_DUR', 'days_of_history', 'rv_before_5d_raw',
        'rv_before_5d_annual', 'mom_before_5d', 'rv_before_7d_raw',
        'rv_before_7d_annual', 'mom_before_7d', 'rv_before_21d_raw',
        'rv_before_21d_annual', 'mom_before_21d', 'rv_before_63d_raw',
        'rv_before_63d_annual', 'mom_before_63d', 'atr_before_21d',
        'vol5', 'vol63', 'vol5_over_vol63',
        'vol5_zscore_vs_63', 'weekday', 'is_month_end', 'is_quarter_end',
        'target_logret_1d']
df_empty = pd.DataFrame(columns=cols)

leng = len(df)
for index, row in df.iterrows():
    print(f'{index}/{leng}', row['symbol'])
    a = build_earnings_dataset_simple(row['symbol'], row['date_ts'])
    df_empty = pd.concat([df_empty,a])
```

Listing 1: Financial Metric Feature Creation

```
# LLM pipelines
pipe_roberta = pipeline(
    'text-classification',
    model='NLPScholars/Roberta-Earning-Call-Transcript-Classification',
    tokenizer='NLPScholars/Roberta-Earning-Call-Transcript-Classification',
    function_to_apply='sigmoid',
    return_all_scores=True, truncation=True, padding='longest', max_length=512,
    batch_size=32, device=device
```

```
)
pipe_bert = pipeline(
    'sentiment-analysis',
    model='nlptown/bert-base-multilingual-uncased-sentiment',
    return_all_scores=True, truncation=True, padding='longest', max_length=512,
    batch_size=32, device=device
)
pipe_finbert = pipeline(
    'sentiment-analysis',
    model='yiyanghkust/finbert-tone', tokenizer='yiyanghkust/finbert-tone',
    return_all_scores=True, truncation=True, padding='longest', max_length=512,
    batch_size=32, device=device
)

# Unifying differrent LLM outputs to Positive, Neutral and Negative sentiment
    classes
map_roberta = {'Positive':'Positive','Negative':'Negative',
               'Uncertain':'Neutral','Constraining':'Neutral','Litigious':'
    Negative'
              }
map_bert = {'1 star':'Negative','2 stars':'Negative',
            '3 stars':'Neutral','4 stars':'Positive','5 stars':'Positive'
           }

# Input slicer to not extend passed LLM token limit
def chunk_text(text: str, tokenizer):
    toks = tokenizer(text, return_tensors='pt', add_special_tokens=False).
    input_ids[0]
    return [ tokenizer.decode(toks[i : i + 512], clean_up_tokenization_spaces=
    True) for i in range(0, len(toks), 512)]

# Sentiment extraction functions made for each model, where X - roberta, bert,
    finbert
def classify_X_batch(texts):
    # Pipeline execution by chunk
    chunks_per = [chunk_text(t, pipe_X.tokenizer) for t in texts]
    flat_chunks = [chunk for chunks in chunks_per for chunk in chunks]
    raw_outs = pipe_X(flat_chunks)

    # Final result assignment to passed item in a batch
    results = []
    idx = 0
    for chunks in chunks_per:
        acc = {'Positive': [], 'Neutral': [], 'Negative': []}
        for _ in chunks:
            for e in raw_outs[idx]:
                lbl = map_X.get(e['label'], 'Neutral')
                acc[lbl].append(e['score'])
            idx += 1
```

```
        results.append({k: (sum(v)/len(v) if v else 0.0) for k, v in acc.items()
    })
    return results

# Dataset preparation
df = pd.read_parquet('hf://datasets/kurry/sp500_earnings_transcripts/
    parquet_files/part-0.parquet')
shorter_df = df[['company_name','symbol','year','quarter','date','
    structured_content']]
content = shorter_df['structured_content']

# Sentiment extraction performed by looping through speaker fragments, batching
    them and passing to LLM pipeline. Each result is individualy assigned back to
     the main file. Each step is printed for progress tracking.
batch_size = 64
for n in range(0,len(content)):
    print(f'Processing #{n} (segments={len(content[n])})')
    for start in range(0, len(content[n]), batch_size):
        batch = content[n][start : start + batch_size]
        texts = [seg['text'] for seg in batch]

        # Run all three LLMs
        finb_res  = classify_finbert_batch(texts)
        robs_res  = classify_roberta_batch(texts)
        berts_res = classify_bert_batch(texts)

        # Assign output to dataset
        for i in range(len(batch)):
            idx = start + i
            content[n][idx]['Finbert'] = finb_res[i]
            content[n][idx]['Roberta'] = robs_res[i]
            content[n][idx]['Bert']    = berts_res[i]
```

Listing 2: BERT FinBERT And RoBERTa Sentiment Extraction Code

```
# Filtered primary dataset is used containing only technology, healthcare,
    utility sector and top 3 speakers.
df = pd.read_json('gpt_input.json', orient='records', lines=True)
api_key = os.environ['OPENAI_API_KEY']='XXXX'
client = OpenAI(api_key=api_key)

# Refined instruction prompt used to format and guide GPT model.
instruction = (
"""
You are a financial-text sentiment classifier.
You will receive multiple financial text fragments from technology, healtcare
    and utilities sectors.
Each fragment appears as:
<> segment N <> [text fragment]
(where N is the fragment number).
```

```
14
15 For each fragment:
16 - Assign probability scores for the categories: Positive, Neutral, Negative.
17 - Scores must be numeric with exactly two decimals.
18 - The three scores must sum to exactly 1.00 (normalize if needed).
19 - Use these sentiment definitions:
20 1. Positive: clearly optimistic, confident, strong results, improving metrics,
       raised guidance, strong demand.
21 2. Neutral: balanced or mixed tone; neither clearly positive nor clearly
       negative.
22 3. Negative: cautious, pessimistic, weak metrics, deteriorating performance,
       lowered guidance, emphasized risks.
23 - Evaluate them from a financial analyst perspective who wants to bring
       meaningful signals to his investor.
24
25
26 Output rules:
27 - Output one line per fragment.
28 - Use this exact format (no deviations): segment=N, positive=X.XX, neutral=Y.YY,
        negative=Z.ZZ
29 - Do NOT add explanations, commentary, or extra text.
30 - Maintain the same order as the fragments provided.
31
32 Examples:
33 <> segment 1 <> ["We saw strong growth in North America and have raised full-
       year guidance."]
34 segment=1, positive=0.85, neutral=0.10, negative=0.05
35 <> segment 2 <> ["While revenue met expectations, margins remained flat and
       macro uncertainty persists."]
36 segment=2, positive=0.20, neutral=0.70, negative=0.10
37 <> segment 3 <> ["We are reducing our outlook due to weakening demand and
       increased headwinds in the supply chain."]
38 segment=3, positive=0.05, neutral=0.15, negative=0.80
39 """
40 )
41
42 # As OpenAI models have token limit per minute a wait function was used till
       tokens refreshed.
43 def with_retry(func, *args, **kwargs):
44     for i in range(12):
45         try:
46             return func(*args, **kwargs)
47         except openai.RateLimitError as e:
48             wait = 5
49             print(f'Waiting {wait}')
50             time.sleep(wait)
51     raise Exception('Failed reset')
52
53 # Reasoning was enabled by passing 'medium' variable and disabled with 'low'
```

```
anss = []
for n in range(0, len(df), 5):
    fragments = []
    s=''
    # As each input and output consumes tokens, to limit costs they were aggregated into batches with maximum token limit per chunk. Speakers were iterated by changing 'speaker_X' value to speaker_1, speaker_2, speaker_3
    for kk in range(n, min(n + 5, len(df))):
        for item in df['speaker_X'].iloc[kk]:
            text = item.get('text', '')
            if 150 < len(text) < 10000:\
                s=s+text
        label = f' <> segment {kk} <>'
        fragments.append((label, s))
    if not fragments:
        continue

    prompt = ''
    for label, txt in fragments:
        prompt += f'|| [ {label}: {txt} ]'
    response = with_retry(client.responses.create,
        model='gpt-5-nano',
        instructions = instruction,
        input=prompt,
        reasoning={'effort': 'medium'}
    )

    output = getattr(response, 'output_text', '')
    answers = output.splitlines()
    anss = anss + answers
    print(f'{n+5}/{len(df)} processed.')

# Results extraction to individual sentiment class
segment = []
positive = []
neutral = []
negative = []
for n in anss:
    texts = n.replace('segment=','').replace('positive=','').replace('neutral=','').replace('negative=','').split(', ')
    try:
        segment.append(texts[0])
        positive.append(texts[1])
        neutral.append(texts[2])
        negative.append(texts[3])
    # In case of GPT-5-nano not complying with instructions, fragment was skipped
    except IndexError:
        segment.append(None)
```

```
        positive.append(None)
        neutral.append(None)
        negative.append(None)

# Each speaker output was saved as individual .csv file. X = 1,2,3
output_df = pd.DataFrame({
        'segment':segment,
        'positive':positive,
        'neutral':neutral,
        'negative':negative
})
output_df.to_csv('gpt_outputs_speakerX.csv')

# After all sentiment extraction each speaker's GPT output is loaded and merged
    with earning calls transcript
speak_1 = pd.read_csv('gpt_outputs_speaker1.csv')
speak_2 = pd.read_csv('gpt_outputs_speaker2.csv')
speak_3 = pd.read_csv('gpt_outputs_speaker3.csv')

speak_1 = speak_1.rename(columns={'positive':'speaker1_GPT_r_Positive_series',
                                  'neutral':'speaker1_GPT_r_Neutral_series',
                                  'negative':'speaker1_GPT_r_Negative_series',})
speak_2 = speak_2.rename(columns={'positive':'speaker2_GPT_r_Positive_series',
                                  'neutral':'speaker2_GPT_r_Neutral_series',
                                  'negative':'speaker2_GPT_r_Negative_series',})
speak_3 = speak_3.rename(columns={'positive':'speaker3_GPT_r_Positive_series',
                                  'neutral':'speaker3_GPT_r_Neutral_series',
                                  'negative':'speaker3_GPT_r_Negative_series',})

df__ = df.merge(speak_1[['segment','speaker1_GPT_r_Positive_series','
    speaker1_GPT_r_Neutral_series','speaker1_GPT_r_Negative_series']], on='
    segment', how = 'left')
df__ = df__.merge(speak_2[['segment','speaker2_GPT_r_Positive_series','
    speaker2_GPT_r_Neutral_series','speaker2_GPT_r_Negative_series']], on='
    segment', how = 'left')
df__ = df__.merge(speak_3[['segment','speaker3_GPT_r_Positive_series','
    speaker3_GPT_r_Neutral_series','speaker3_GPT_r_Negative_series']], on='
    segment', how = 'left')
```

Listing 3: GPT Sentiment Extraction Code

```
#####################################
# DATA LOADING AND SPLIT PREPARATION #
#####################################
# Financial metric dataset loading
stock_yh = pd.read_csv('financial_metrics.csv')
stock_df = stock_yh.drop_duplicates()
stock_df['earnings_datetime'] = pd.to_datetime(stock_df['earnings_datetime'],
    format='mixed', errors='coerce')
# BERT and FinBERT sentiment loading
```

```
sentiment_df = pd.read_csv('sentiment_df.csv')
sentiment_df['date'] = pd.to_datetime(sentiment_df['date'], format='mixed',
    errors='coerce')
# GPT produced sentiment loading
gpt_sentiment = pd.read_json('gpt_input_noreason.json', orient='records', lines=
    True)
gpt_r_sentiment = pd.read_json('gpt_r_input.json', orient='records', lines=True)
gpt_sentiment['date'] = pd.to_datetime(gpt_sentiment['date'], format='mixed',
    errors='coerce')
gpt_r_sentiment['date'] = pd.to_datetime(gpt_r_sentiment['date'], format='mixed'
    , errors='coerce')

sentiment_dfs = pd.merge(sentiment_df, gpt_sentiment[['symbol','date','
    speaker1_GPT_Positive_series',
        'speaker1_GPT_Neutral_series', 'speaker1_GPT_Negative_series',
        'speaker2_GPT_Positive_series', 'speaker2_GPT_Neutral_series',
        'speaker2_GPT_Negative_series', 'speaker3_GPT_Positive_series',
        'speaker3_GPT_Neutral_series', 'speaker3_GPT_Negative_series']], how='
    left', left_on=['symbol', 'date'], right_on=['symbol', 'date'])
sentiment_r_dfs = pd.merge(sentiment_dfs, gpt_r_sentiment[['symbol','date','
    speaker1_GPT_r_Positive_series',
        'speaker1_GPT_r_Neutral_series', 'speaker1_GPT_r_Negative_series',
        'speaker2_GPT_r_Positive_series', 'speaker2_GPT_r_Neutral_series',
        'speaker2_GPT_r_Negative_series', 'speaker3_GPT_r_Positive_series',
        'speaker3_GPT_r_Neutral_series', 'speaker3_GPT_r_Negative_series']], how=
    'left', left_on=['symbol', 'date'], right_on=['symbol', 'date'])
# Sector information per ticker loading
stock_sectors = pd.read_csv('tickers_w_sector.csv')
stock_sectors['date_ts'] = pd.to_datetime(stock_sectors['date_ts'], format='
    mixed', errors='coerce')

stock_merge_ = pd.merge(stock_df, stock_sectors, how='left', left_on=['ticker',
    'earnings_datetime'], right_on=['symbol', 'date_ts'])
stock_merge_temp = pd.merge(stock_merge_, sentiment_r_dfs, how='left', left_on=[
    'ticker', 'earnings_datetime'], right_on=['symbol', 'date'])

stock_df_final_v = stock_merge_temp.drop(columns=[
    'earnings_date', 'date_x', 'date_y', 'date_ts', 'quarter_x', 'year_y',
    'symbol_x', 'symbol_y', 'company_name_x', 'company_name_y'
], errors='ignore')

df = stock_df_final_v.copy()
df['day'] = df['earnings_datetime'].dt.day

df_clean = df.loc[df['error'].isna()]
df_clean = df_clean.loc[~df_clean['end_finbert_negative_avg'].isna()]
df_clean = df_clean.loc[~df_clean['speaker3_Finbert_Negative_slope'].isna()]
df_clean = df_clean.loc[~df_clean['mom_before_21d'].isna()]
df_clean = df_clean.loc[~df_clean['speaker1_GPT_Positive_series'].isna()]
```

```
df_clean = df_clean.loc[~df_clean['speaker2_GPT_r_Positive_series'].isna()]
df_clean = df_clean.loc[~df_clean['speaker3_GPT_r_Negative_series'].isna()]
df_clean = df_clean.loc[~df_clean['speaker3_GPT_r_Positive_series'].isna()]

for col, count in df_clean.isna().sum().items():
    if count > 0:
        print(f'{col} -- {count}')

# Data sorting by time
df_clean = df_clean.sort_values(['earnings_datetime'])

# Binary target creation
Y = df_clean[['target_logret_1d', 'year_y','quarter_y']]
Y_b = Y.copy()
Y_b['target_logret_1d'] = Y_b['target_logret_1d'].where(Y_b['target_logret_1d']
    == 0, (Y_b['target_logret_1d'] >0).astype(int))

# Removing pre-process columns
drop_cols = [
    'ticker', 'earnings_datetime', 'reaction_date','year_y',
    'earnings_timing', 'speaker_list','error','target_logret_1d',
    'speaker1_Bert_Positive_series', 'speaker2_Bert_Positive_series', '
    speaker3_Bert_Positive_series',
    'speaker1_Bert_Neutral_series', 'speaker2_Bert_Neutral_series', '
    speaker3_Bert_Neutral_series',
    'speaker1_Bert_Negative_series', 'speaker2_Bert_Negative_series', '
    speaker3_Bert_Negative_series',
    'speaker1_Finbert_Positive_series', 'speaker2_Finbert_Positive_series', '
    speaker3_Finbert_Positive_series',
    'speaker1_Finbert_Neutral_series', 'speaker2_Finbert_Neutral_series', '
    speaker3_Finbert_Neutral_series',
    'speaker1_Finbert_Negative_series', 'speaker2_Finbert_Negative_series', '
    speaker3_Finbert_Negative_series',
]

X = df_clean.drop(columns=drop_cols, errors='ignore')

# Ensuring that no object dtypes are present
category_mappings = {}
for col in X.select_dtypes(include=['object', 'category']).columns:
    X[col] = X[col].astype('category')
    category_mappings[col] = dict(enumerate(X[col].cat.categories))
    X[col] = X[col].astype('category').cat.codes

# Disabled shuffling in data split
X_train, X_test, Y_b_train, Y_b_test, Y_train_actual, Y_test_actual =
    train_test_split(X, Y_b, Y, test_size=0.20, random_state=13, shuffle=False)

####################
```

```
# XGBoost Training #
####################

xgb_clf = xgb.XGBClassifier(
    objective='binary:logistic',
    tree_method='hist',
    device='cuda',
    random_state=13,
    eval_metric='auc',
    enable_categorical=True,
    verbosity=1
)

param_dist = {
    'n_estimators': [100, 200, 300, 400, 500],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5, 6, 8, 10],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'gamma': [0, 0.1, 0.2, 0.5],
    'min_child_weight': [1, 3, 5]
}

random_search = RandomizedSearchCV(
    estimator=xgb_clf,
    param_distributions=param_dist,
    n_iter=100,
    scoring='roc_auc',
    cv=5,
    verbose=2,
    random_state=13,
    n_jobs=1
)

random_search.fit(X_train, Y_b_train['target_logret_1d'])
xgboost_model = random_search.best_estimator_
y_pred_xgboost = xgboost_model.predict(X_test)

######################
# LightGBM Training #
######################

lgb_estimator = lgb.LGBMClassifier(
    objective='binary',
    random_state=13,
    n_jobs=-1,
    device='gpu',
    verbose = 1
)
```

```
137
138 param_grid = {
139     'n_estimators': [200, 500, 800],
140     'learning_rate': [0.01, 0.05, 0.1],
141     'max_depth': [1, 4, 6, 8],
142     'num_leaves': [31, 63, 127],
143     'min_child_samples': [20, 50, 100, 200],
144     'bagging_fraction': [0.6, 0.8, 1.0],
145     'feature_fraction': [0.6, 0.8, 1.0],
146     'lambda_l1': [0.0, 0.1, 0.5, 1.0],
147     'lambda_l2': [0.0, 0.5, 1.0, 5.0],
148     'min_split_gain': [0.0, 0.01, 0.1],
149 }
150
151 random_search = RandomizedSearchCV(
152     estimator=lgb_estimator,
153     param_distributions=param_grid,
154     n_iter=80,
155     scoring='roc_auc',
156     cv=5,
157     verbose=1,
158     random_state=13,
159     n_jobs=1
160 )
161
162 random_search.fit(X_train, Y_b_train['target_logret_1d'])
163
164 lightgbm_model = random_search.best_estimator_
165 y_pred_lgbm = lightgbm_model.predict(X_test)
166
167 ###############
168 # RF Training #
169 ###############
170
171 rf_clf = RandomForestClassifier(
172     n_estimators=500,
173     random_state=13,
174     class_weight='balanced_subsample',
175     bootstrap=True
176 )
177
178 param_dist = {
179     'n_estimators': [300, 500, 800, 1200],
180     'max_depth': [4, 6, 8, 12, 16],
181     'min_samples_split': [2, 5, 10, 20],
182     'min_samples_leaf': [1, 2, 5, 10],
183     'max_features': ['sqrt', 'log2', 0.3, 0.5, 0.8],
184     'bootstrap': [True],
185 }
```

```
186
187 random_search = RandomizedSearchCV(
188     estimator=rf_clf,
189     param_distributions=param_dist,
190     n_iter=80,
191     scoring='roc_auc',
192     cv=5,
193     verbose=2,
194     random_state=13,
195     n_jobs=1
196 )
197
198 random_search.fit(X_train, Y_b_train['target_logret_1d'])
199
200 rf_model = random_search.best_estimator_
201 y_pred_rf = rf_model.predict(X_test)
202
203 ###################
204 # Lasso Training #
205 ###################
206
207 lasso_pipe = Pipeline([
208     ('scaler', StandardScaler()),
209     ('model', LogisticRegression(
210         penalty='l1',
211         solver='saga',
212         max_iter=5000,
213         random_state=13
214     ))
215 ])
216
217 lasso_param_grid = {
218     'model__C': [0.001, 0.01, 0.05, 0.1, 0.5, 1.0]
219 }
220
221 lasso_search = GridSearchCV(
222     lasso_pipe,
223     lasso_param_grid,
224     scoring='roc_auc',
225     cv=5,
226     verbose=2,
227     n_jobs=1
228 )
229
230 lasso_search.fit(X_train, Y_b_train['target_logret_1d'])
231
232 lasso_model = lasso_search.best_estimator_
233 y_proba_lasso = lasso_model.predict_proba(X_test)[:, 1]
234 y_pred_lasso = (y_proba_lasso > 0.5).astype(int)
```

```
235
236 ##################
237 # Ridge Training #
238 ##################
239
240 ridge_pipe = Pipeline([
241     ('scaler', StandardScaler()),
242     ('model', LogisticRegression(
243         penalty='l2',
244         solver='lbfgs',
245         max_iter=5000,
246         random_state=13
247     ))
248 ])
249
250 ridge_param_grid = {
251     'model__C': [0.001, 0.01, 0.05, 0.1, 0.5, 1.0, 5.0]
252 }
253
254 ridge_search = GridSearchCV(
255     ridge_pipe,
256     ridge_param_grid,
257     scoring='roc_auc',
258     cv=5,
259     verbose=2,
260     n_jobs=1
261 )
262
263 ridge_search.fit(X_train, Y_b_train['target_logret_1d'])
264 ridge_model = ridge_search.best_estimator_
265 y_proba_ridge = ridge_model.predict_proba(X_test)[:, 1]
266 y_pred_ridge = (y_proba_ridge > 0.5).astype(int)
267
268 # After each model training their results are reviewed individualy
269 # print('Best Par:', X_search.best_estimator_)
270 # print('Acc:', accuracy_score(Y_b_test['target_logret_1d'], y_pred_X))
271 # print(classification_report(Y_b_test['target_logret_1d'], y_pred_X))
```

Listing 4: Model Training Workflow Code

```
1 ##################
2 # MODEL TRADING #
3 ##################
4 df_test = X_test.copy()
5 # Extracting probabilities that stock price will go up. Pretrained model is
      selected in pretrain_model variable
6 df_test['price_up'] = pretrain_model.predict_proba(X_test)[:, 1]
7 df_test['pred'] = (df_test['price_up'] > 0.52).astype(int)
8
```

```
# Adding actual price movement and sorting trades by probability to take 3 trades with highest probability score
df_test['actual_return'] = Y_test_actual['target_logret_1d'].values
quarter_to_month = {1: 1, 2: 4, 3: 7, 4: 10}
df_test['year'] = Y_test_actual['year_y']
df_test['month'] = Y_test_actual['quarter_y'].map(quarter_to_month)
df_test['date'] = pd.to_datetime(dict(year=df_test['year'], month=df_test['month'], day=1))
df_test = df_test.sort_values('date').reset_index(drop=True)
df_test['sector'] = df_test['sector'].map(category_mappings['sector'])

trd = 3
df_test = df_test.sort_values(['sector', 'date', 'price_up'], ascending=[True, True, False])
df_test['take_trade'] = (df_test.groupby(['sector', 'date']).cumcount() < trd)
df_test['trade_return'] = np.where(df_test['take_trade'] & (df_test['pred'] == 1),df_test['actual_return'],0.0)
sector_strat = df_test.loc[df_test['take_trade'] & (df_test['pred'] == 1)].groupby(['sector', 'date']).agg(trade_return=('actual_return', 'mean'), trades_taken=('actual_return', 'size')).reset_index()
sector_strat['strategy_return'] = (sector_strat.groupby('sector')['trade_return'].cumsum())

# Strategy return by sector calculation compounding generated value into portfolio
income = 100.0
sector_strat = sector_strat.sort_values(['sector', 'date']).copy()

sector_frame = []
for sector, i in sector_strat.groupby('sector', sort=False):
    portfel = 0.0
    total_contributed = 0.0
    equity_path = []

    all_dates = pd.date_range(i['date'].min(), i['date'].max(), freq='QS')
    n = i.set_index('date')['trade_return'].reindex(all_dates, fill_value=0.0)

    for dt, logret in n.items():
        portfel += income
        total_contributed += income
        portfel *= np.exp(logret)
        equity_path.append((dt, portfel, total_contributed))

    sector_df = pd.DataFrame(equity_path, columns=['date', 'portfel', 'total_contributed'])
    sector_df['sector'] = sector
    sector_df['strategy_return'] = (sector_df['portfel'] / sector_df['total_contributed'] - 1)

```

```
    sector_frame.append(sector_df)

value_by_sector = pd.concat(sector_frame, ignore_index=True)

##################
# INDEX BASELINE #
##################

# Downloading and creating S&P500 stock's close price dataset
sp500 = yf.download('^GSPC', start='2022-07-01', end='2025-04-15')
sp500.columns = ['_'.join(col).strip() if isinstance(col, tuple) else col for
    col in sp500.columns]
sp500 = sp500[['Close_^GSPC']].rename(columns={'Close_^GSPC': 'sp500_close'})
sp500.index = pd.to_datetime(sp500.index)
sp500['return'] = np.log(sp500['sp500_close'] / sp500['sp500_close'].shift(1))
sp500 = sp500.dropna().copy()

# Indexing days when index buys can be performed
trade_days = sp500.loc[(sp500.index.day >= 12) & (sp500.index.day <= 15)].copy()
buy_days = (trade_days.groupby([trade_days.index.year, trade_days.index.
    to_period('Q')]).head(1).index)
sp500['buy_day'] = sp500.index.isin(buy_days)

income = 100.0
portfel = 0.0
ttl_return = []

# Calculating index performance by ttl returns and DCA
for dt, row in sp500.iterrows():
    if row['buy_day']:
        portfel += income
    portfel *= np.exp(row['return'])
    ttl_return.append(portfel)

sp500['portfel_value'] = ttl_return
sp500['ttl_contribution'] = income * sp500['buy_day'].cumsum()
sp500['strategy_return'] = (sp500['portfel_value'] / sp500['ttl_contribution'] -
    1) * 100
sp500['sector'] = 'S&P500'

df_comparison = pd.concat([value_by_sector[['date','sector', 'strategy_return'
    ]],sp500[['date','sector', 'strategy_return']].loc[sp500['buy_day']==True]]).
    reset_index()

# Visualising strategy returns by date
plt.figure(figsize=(12,6))
sectors = df_comparison['sector'].unique()
for sector in sectors:
    sector_data = df_comparison[df_comparison['sector'] == sector]
```

```
92      plt.plot(sector_data['date'],
93               sector_data['strategy_return'],
94               label=sector)
95  plt.xlabel('Date')
96  plt.ylabel('Cumulative Returns, %')
97  plt.title('Trading Simulation vs Market')
98  plt.legend()
99  plt.show()
```

Listing 5: Simulation Code