Advanced Programming 2025

Textual Monetary Policy Signals and Equity Market Direction: A Comparative Embedding Study

Final Project Report

LARAOUI Aymen
aymen.laraoui@unil.ch
Student ID: 20827556

December 27, 2025

Abstract

This report studies whether the language of FOMC policy statements contains information about the next-day direction of the S&P 500. I pair each statement with the subsequent close-to-close index return and compare a TF–IDF baseline with transformer embeddings (BERT, FinBERT, and GTE) combined with standard machine-learning models. Performance is evaluated primarily with ROC-AUC and robustness checks across fixed model configurations. The results suggest that transformer-based representations extract a weak but more stable directional signal than TF–IDF.

Keywords: data science, Python, machine learning, financial NLP, text embeddings

Contents

1 Introduction

Scheduled central-bank communication is among the clearest recurring information events in financial markets. In the United States, the Federal Open Market Committee (FOMC) releases policy statements at pre-announced times, and these texts are immediately parsed by market participants. Under the semi-strong form of the Efficient Market Hypothesis, any publicly released information should be incorporated into prices quickly, which makes predicting *next-day* equity-market moves from the statement text a deliberately hard task.

This project asks a narrow question: *does the language of FOMC statements contain any residual information about the next-day direction of the S&P 500, and does that depend on how the text is represented?* To answer it, I construct a dataset pairing each statement with the next-day close-to-close S&P 500 return and a binary direction label. The corpus covers the period February 4, 1994 to October 29, 2025 and contains 237 statements. I then compare four text representations (TF–IDF, BERT, FinBERT, and GTE) combined with standard machine-learning models for classification and regression. Because the expected signal at a one-day horizon is weak and threshold choices can be arbitrary, evaluation focuses on directional ROC-AUC, complemented by accuracy relative to a naive benchmark.

The contribution is primarily methodological: (i) a reproducible, end-to-end pipeline that produces a clean statement/return dataset and can optionally updates it when new statements appear, (ii) a controlled embedding comparison under a consistent train/test split to avoid look-ahead bias, and (iii) a robustness-first selection logic (via simple sensitivity checks across fixed configurations) to reduce the risk of over-interpreting lucky hyperparameter choices in a small sample.

The remainder of the report first describes the data and methodology, then presents the results and robustness diagnostics, and finally discusses limitations and future extensions.

2 Literature Review

From a theoretical perspective, the semi-strong form of the Efficient Market Hypothesis provides the natural benchmark for this study. Because Federal Open Market Committee (FOMC) statements are publicly released at known times, their informational content should be rapidly incorporated into asset prices. Once prices have adjusted following the announcement, subsequent market movements—such as next-day returns—should therefore not systematically reflect the content of the statement. This view is supported by empirical work on FOMC communication: Hansen and McMahon (2016) show that textual variation in FOMC statements conveys forward-looking information and triggers immediate market reactions, while Lucca and Moench (2015) document that equity market adjustments related to FOMC announcements occur largely before or at the time of the release.

Against this background, the objective of this project is not to construct a predictive tool, but to examine whether any residual information from FOMC statements can nevertheless be detected in next-day equity market outcomes, and how the ability to extract such information depends on the textual representation used. Early approaches to financial text analysis, such as those discussed by Loughran and McDonald (2011), relied on bag-of-words representations like TF-IDF combined with linear models; while transparent, these methods ignore semantic context and may struggle with nuanced language.

More recent work in natural language processing shows that contextual transformer-based representations, introduced by Devlin et al. (2019) with BERT, provide richer text features by capturing meaning in context.

Araci (2019) further shows that domain-specific pre-training can improve performance on financial NLP tasks; in the context of FOMC statements, which use structured economic language and policy-specific phrasing, a finance-adapted model such as FinBERT is a natural candidate to test whether specialized representations extract information more effectively than a generic BERT baseline.

In parallel, Li et al. (2023) show that general text embeddings trained with multi-stage contrastive learning can perform strongly across a wide range of downstream tasks without domain-specific fine-tuning. Including GTE alongside the finance-specific FinBERT model therefore allows this project to test a simple but important question: does extracting information from central bank communication require a specialized financial language model, or can a strong general-purpose embedder capture whatever residual market-relevant signal (if any) remains in FOMC statements?

Motivated by this literature, the present study compares TF-IDF, BERT, FinBERT, and GTE embeddings within a unified evaluation framework, using FOMC statements as the text corpus and next-day S&P 500 returns as market outcomes. This comparison allows us to assess whether increasingly expressive text representations capture different aspects of central bank communication, even in a setting where market efficiency implies that any remaining information is likely to be weak.

3 Methodology

3.1 Data Description

The main data required for this project consists of FOMC statements and the next-day stock market return following each statement's release date.

FOMC statements are initially collected from two public GitHub repositories. All metrics and results reported in Sections 4–5 are computed on a frozen snapshot containing 237 statements, spanning February 4, 1994 to October 29, 2025. These datasets are used as-is and stored in CSV format. To keep the dataset up to date, the pipeline includes an optional update step (enabled via `python main.py -update-data`) that checks whether new FOMC statements are missing from the existing dataset. When a new statement is detected, the program downloads the corresponding HTML page from the Federal Reserve website and stores it in `data/raw`. The purpose of keeping the original HTML is to preserve an immutable "source-of-truth" snapshot of the statement exactly as published, which makes the pipeline reproducible and auditable: if cleaning rules change, or if a parsing error is discovered, the statement can be re-extracted without re-downloading or relying on a potentially updated webpage. Storing raw HTML also helps diagnose extraction issues caused by changes in webpage structure over time. After storage, the relevant textual content is extracted into a `.txt` file stored in `data/interim`, and the cleaned version is then incorporated into the final processed dataset used for the empirical analysis. Each observation in the final processed dataset corresponds to one statement date and includes a statement identifier and date, the cleaned statement text, the next-day S&P 500 return, and a binary direction label (1 if the return is positive, 0 otherwise). The final dataset used for all empirical analysis is stored as `data/processed/fomc_statements.csv`.

Stock market data are obtained for the S&P 500 index. The next-day return is computed as the percentage change between the closing price on the FOMC statement release day and the closing price on the following trading day. Since FOMC statements are typically released during U.S. trading hours, we assume that prices partially incorporate the information on the release day and we try to capture the residual effect using close-to-close returns. If a statement date falls on a non-trading day (e.g., a weekend), the "statement-day" close is taken as the most recent prior trading close, and the "next-day" close is taken as the next available trading close. S&P 500 closing prices are retrieved via `yfinance`, so results can depend on data availability and occasional vendor-side revisions.

Data Collection Limitation: the automated updater scrapes the FOMC calendar page and may not immediately capture extraordinary intermeeting statements if they are not listed in the standard calendar format. While the historical dataset (1994–2025) includes emergency statements that have since been archived on the calendar, future emergency communications may require manual verification. Finally, text cleaning is intentionally minimal to avoid manual bias, so some boilerplate (e.g., contact lines or voting blocks) may remain in the cleaned text.

3.2 Approach

As introduced earlier, the core idea of the technical approach is to convert each FOMC statement into a numerical vector representation and use these vectors as inputs to standard machine-learning models. We compare four embedders: TF–IDF as a simple and transparent baseline, two general-purpose pretrained transformer models (BERT and GTE), and one finance-oriented transformer (FinBERT). This choice is not arbitrary: it is meant to form a clear ladder of text representations, moving from sparse bag-of-words features to contextual embeddings, and to test whether a domain-adapted model such as FinBERT provides an advantage on monetary-policy language.

These embeddings are then used to train three classification models (Logistic Regression, Random Forest, and XGBoost) and three regression models (Elastic Net, Random Forest, and XGBoost). Regularized linear models (Logistic Regression / Elastic Net) provide strong and stable baselines in high-dimensional settings and allow an interpretable reference point. We then include non-linear tree-based methods because the market reaction to policy communication may depend on interactions (e.g., combinations of wording, tone, and context) that linear models cannot capture. Random Forest is included as a relatively robust non-linear model through bagging, and XGBoost is included as a standard high-performing boosting method that can capture more refined non-linear patterns. At the same time, we explicitly acknowledge that the dataset is small (237 statements), so model complexity must be controlled carefully to avoid fitting noise.

For this reason, we do not run a large hyperparameter grid search. With a limited sample, extensive tuning would tend to fit the quirks of this particular train/test split and can overstate performance by selecting hyperparameters that work well by chance rather than generalizing. Instead, we evaluate a small set of fixed configurations (conservative / moderate / aggressive) for each model. This also enables a transparent sensitivity analysis: by comparing performance across these configurations, we can assess robustness to hyperparameter choices and categorize specifications as stable or fragile, which supports selecting a single reference specification for inference based on both performance and robustness.

Overall, the evaluation follows a three-stage procedure. First, for each task (classification and regression), we run the full grid of experiments over all (embedder, model) pairs under the three fixed configurations and record ROC-AUC on the held-out test period. Second, we perform sensitivity analysis by summarizing each (embedder, model) pair with its mean ROC-AUC across the three configurations and dispersion measures (range and CV), which determines whether the pair is robust to hyperparameter choices. Third, we convert these results into a diagnostic categorization: a pair is labeled *Uninformative* if its mean ROC-AUC is close to random (near 0.5), *Dangerous* if it appears informative but is not robust, and *Reliable* if it is both informative and robust. Finally, the "Grand Champion" specification used for inference is selected by a lexicographic rule: we first filter to robust pairs (mandatory), then choose the robust pair with the highest mean ROC-AUC, which will be our inference model.

In terms of preprocessing, the input texts come from the cleaned statement field in `data/processed/fomc_statements.csv`. Preprocessing is intentionally minimal: the objective is mainly to standardize formatting (whitespace normalization and removal of obvious administrative boilerplate when present) without introducing subjective manual filtering. The dataset is split chronologically into training and test sets (70% / 30%, no shuffling) to avoid look-ahead bias. For TF–IDF, the vectorizer is fit on the training texts only and then applied to the test set to prevent leakage (with `min_df=2` *i.e., we drop terms that appear in fewer than two training documents* and a vocabulary size chosen dynamically based on the training corpus). For transformer embedders, statements are tokenized with padding/truncation to a maximum length of 512 tokens. Transformer models are used as frozen feature extractors (no fine-tuning): for BERT and FinBERT, the statement embedding is taken from the last-layer `[CLS]` token (768 dimensions), while for GTE the embedding is computed using attention-mask mean pooling over token embeddings (1024 dimensions). In terms of computational cost, TF–IDF is fast, whereas transformer embeddings are the main bottleneck and scale with the total number of tokens processed; therefore, embeddings are cached to disk in `data/embeddings_cache` after the first run. The cache is reused when the dataset size is unchanged, and automatically invalidated (cleared and recomputed) if new statements are added.

Model performance is evaluated primarily in terms of market direction, and ROC-AUC is the main metric used throughout the project. The reason is that our objective is to detect whether the statement text contains any residual directional information, even if the signal is weak. In this setting, ROC-AUC is directly aligned with the question: it measures whether the model systematically assigns higher scores to days that end up being “up” than to days that end up being “down”.An ROC-AUC above 0.5 by more than a small noise band therefore suggests predictive ordering beyond randomness, which is exactly what we are looking for in a context close to market efficiency. Unlike accuracy, ROC-AUC does not depend on a single arbitrary decision threshold (e.g., 0.5 for probabilities or 0 for predicted returns): accuracy can look poor simply because the cutoff is not optimal, while ROC-AUC summarizes performance across all possible thresholds and is therefore more stable and comparable. We still report accuracy as an intuitive complement, but always interpret it relative to a simple naive benchmark (e.g., always predicting the majority class). For regression, the target is the next-day close-to-close return, but we primarily interpret predictions directionally, because at a one-day horizon expected returns are dominated by noise and estimating precise magnitudes is rarely stable. Regression outputs can naturally be treated as continuous directional scores, so we evaluate them using ROC-AUC as well (and report directional accuracy by taking the sign of the predicted return: positive → up, non-positive → down). This makes regression and classification directly comparable under a single, consistent directional evaluation framework.

3.3 Implementation

The project is implemented in Python and relies on a small set of core libraries. Web data collection is handled with `requests` and `BeautifulSoup` (`bs4`) to parse the Federal Reserve webpages and detect/download missing FOMC statements. Data manipulation is mainly done with `pandas` (loading/saving CSV, date handling, tabular operations) and `numpy` (model inputs, arrays, and cached embeddings). Market data are retrieved through `yfinance` to compute next-day S&P 500 returns. For the machine-learning models and TF–IDF, the implementation relies on `scikit-learn` and `xgboost`. The transformer embedders (BERT, FinBERT, GTE) are implemented with `torch` and the HuggingFace `transformers` library.

Setting up and updating the dataset: The dataset is built in two explicit steps. First, `src/data/update_fomc_statements.py` checks which statement text files already exist in `data/interim`. It then scrapes the FOMC calendar page, identifies statement links, downloads only the missing ones, and stores the original webpages as HTML in `data/raw`. From these HTML files, it extracts the main statement content and saves it as plain text files in `data/interim`. Second, `src/data/prepare_processed_data.py` creates the final dataset used by the experiments: it cleans each statement text (minimal cleaning: whitespace normalization and removal of obvious boilerplate) and computes the next-day S&P 500 close-to-close return. Non-trading days are handled programmatically by using the closest available trading close before the statement date and the next available trading close after it. The resulting file is saved as `data/processed/fomc_statements.csv` and contains the features/labels used by the ML pipeline.

Feature extraction (embeddings): The embedding logic is separated into four modules: `src/embedder_tfidf.py`, `src/embedder_bert.py`, `src/embedder_finbert.py`, and `src/embedder_gte.py`. Each module takes the statement texts and outputs a fixed-length vector per statement. TF–IDF is fit on the training texts and then applied to the test texts (to avoid leakage). For transformer models, embeddings are produced using pretrained encoders loaded from HuggingFace: `bert-base-uncased` (BERT), `ProsusAI/finbert` (FinBERT), and `Alibaba-NLP/gte-large-en-v1.5` (GTE). The pooling strategy is implemented inside the embedder modules: BERT/FinBERT use the last-layer `[CLS]` embedding, while GTE uses attention-mask mean pooling to obtain a single sentence vector. From an engineering perspective, transformer embeddings are computed in CPU mode on purpose to keep the project portable and reproducible in a standard environment (no CUDA/device-specific setup). Although GPU execution can accelerate transformer inference, it also introduces non-trivial overhead (device initialization, tensor transfers, and dependency/configuration constraints), and with a relatively small dataset (237 statements) these fixed costs are not necessarily worth the added complexity. Instead, efficiency is handled through batching at embedding time (the `embed_many_*` functions process texts in mini-batches, default `batch_size=8`), which amortizes tokenization and model forward-pass overhead and provides a practical speed-up on CPU. Runtime is further reduced through the caching system described below.

Caching: Since transformer embedding generation is the most time-consuming part of the pipeline, embeddings are cached to disk using `src/embedding_cache.py`. For each embedder, the cache stores `.npy` arrays for train and test embeddings (e.g., `gte_train.npy`, `gte_test.npy`) and a small `.json` metadata file that records the dataset size used to generate the cache. Cache validity is checked by comparing the current dataset size to the cached size; if they differ, the cache is invalidated (cleared) and recomputed. In principle, one could append only the embedding of a newly released statement, but in this project we intentionally favor correctness and reproducibility over incremental complexity. Since the cache stores separate train/test matrices under a chronological split, adding new observations can shift the split boundary and change which statements belong to train versus test, making simple appending error-prone. More generally, incremental caching requires guaranteeing stable ordering and strict alignment between dataset rows and cached vectors; invalidating and recomputing avoids silent misalignment bugs and remains computationally acceptable at this dataset size. In addition, cached embeddings are only valid under the exact same embedding settings (model choice, pooling method, tokenization parameters such as maximum length, and text preprocessing rules); if any of these change, reusing an old cache would silently mix incompatible representations. TF–IDF embeddings are not cached in the same way because TF–IDF is inexpensive to compute compared to transformer embeddings and depends on the training corpus vocabulary and IDF weights. When the training set changes, TF–IDF must be refit anyway; therefore, recomputing TF–IDF on demand is simpler and avoids managing additional cached objects (e.g., a serialized vectorizer).

Model training, evaluation, and inference: The main experiment is orchestrated by `main.py`. It loads `data/processed/fomc_statements.csv`, performs the chronological split via `src/-`

data_loader.py, generates or loads cached embeddings, and then runs a full comparison across all embeddings, models, and fixed configurations. Wrapper classes for the ML estimators are defined in src/models.py to keep a consistent interface across scikit-learn and XGBoost models (so that fit, predict, and predict_proba work uniformly). Performance evaluation and result aggregation are handled by src/evaluation.py, which also computes the sensitivity analysis and diagnostic summary tables. The final model-selection logic that identifies the single reference specification ("Grand Champion") is implemented in src/model_selection.py. All results are automatically saved to the results/ directory (experiment_results.csv, diagnostic_breakdown.csv, and the selected model as grand_champion_model.pkl).

The file src/inference.py provides an interactive prediction interface for the user once training is complete. When the program finishes and asks, “Do you want to run live predictions with the Grand Champion model? (y/n)”, answering “y” launches this module. It loads the saved model and its configuration (results/grand_champion_model.pkl and results/-grand_champion_config.json), restores the corresponding embedder (TF–IDF, BERT, FinBERT, or GTE), and enters a simple command-line loop. The user can then type any custom text, which is embedded, passed through the trained model, and immediately classified as “Market UP” or “Market DOWN”; for regression models, the predicted return in percent is also displayed. This inference mode demonstrates that the pipeline is fully functional end-to-end—from raw FOMC statements to real-time prediction using the selected model.

4 Results

This section reports the empirical results produced by running main.py. We first describe the experimental setup (hardware/software and key hyperparameters), then explain the evaluation logic used throughout the project (directional ROC-AUC, sensitivity analysis, diagnostic categorization, and final model selection). The resulting performance summaries and selected reference specification are presented here, while their interpretation and limitations are discussed in Section 5.

4.1 Experimental Setup

All experiments were run locally in a standard Python environment on a CPU-only machine (no CUDA/GPU requirement). This design choice is intentional: the goal is to keep the project portable and easy to reproduce on a typical laptop without any device-specific setup. Transformer embeddings are the main computational bottleneck, so efficiency is handled with (i) mini-batch embedding (default batch_size=8) to amortize tokenization and forward-pass overhead on CPU, and (ii) on-disk caching so embeddings are computed once and then reused across repeated runs.

The software environment is fully specified by requirements.txt. The experiments were executed with Python 3.13.9 and the main libraries pinned to fixed versions (to avoid silent behavior changes across installations): numpy==2.2.6, pandas==2.3.3, scikit-learn==1.7.2, xgboost==2.1.3, torch==2.9.1, transformers==4.57.3, yfinance==0.2.66, requests==2.32.5, and beautifulsoup4==4.14.2. Pretrained transformer weights are downloaded from HuggingFace on the first run and then reused from the local cache.

Hyperparameters are fixed and deliberately simple given the small sample size (237 statements). Instead of a large grid search, each model family is evaluated under three preset configurations (conservative / moderate / aggressive) to study sensitivity without over-optimizing on a single split. For classification, Logistic Regression varies the regularization strength C in {0.1, 1.0, 10.0}. Random Forest varies n_estimators in {50, 100, 200} and max_depth in {3, 5, 10}.

XGBoost varies `n_estimators` in {50, 100, 200}, `max_depth` in {2, 3, 5}, and `learning_rate` in {0.01, 0.1, 0.3}. For regression, Elastic Net varies `alpha` in {10.0, 1.0, 0.1} with `l1_ratio=0.5`, while Random Forest and XGBoost use the same capacity ladder as in classification. Tree-based models use a fixed `random_state=42` for reproducibility.

On the text side, TF–IDF uses `min_df=2` and a dynamically chosen vocabulary size computed as a fixed fraction of the training vocabulary (`vocab_ratio=0.35`, meaning we keep only the most frequent terms by document frequency, capped at 35% of the training vocabulary size). This choice is intentional and acts as a simple form of regularization. In small text corpora, vocabulary size grows quickly and follows a long-tail pattern: many terms appear only a handful of times, which creates thousands of sparse features that are unlikely to generalize out of sample. Since the training set contains only 165 statements, keeping the full vocabulary would produce a very high feature-to-sample ratio and increase variance without clear benefit. Setting `min_df=2` removes one-off terms, and capping the vocabulary via `vocab_ratio=0.35` further trims the rare-term tail, retaining the most common and stable tokens while discarding features that tend to add noise rather than robust predictive content. Empirically, a relatively small subset of frequent terms accounts for most token occurrences in the corpus, so this restriction reduces dimensionality substantially with limited loss of information.

Transformer embedders use a maximum token length of 512 with padding/truncation; BERT and FinBERT use the last-layer `[CLS]` embedding, while GTE uses attention-mask mean pooling. These settings are kept constant across runs so that performance differences are driven by model/representation choices rather than preprocessing changes.

4.2 Performance Evaluation

Sensitivity analysis

Sensitivity analysis is crucial in this project because the dataset is small (237 statements), so performance estimates have substantial sampling noise and can vary across reasonable hyperparameter settings. The goal is therefore not only to measure performance, but also to assess whether each (embedding, model) pair behaves consistently when hyperparameters change. If a specification is not robust, it is excluded from the set of candidates for the final prediction model used for inference (the "Grand Champion" specification) and treated as potentially misleading in the diagnostic summary (it may look informative under one configuration but fail once parameters are slightly changed).

Concretely, for each embedding and model family we evaluate three fixed configurations (conservative / moderate / aggressive) rather than running a large grid search. This is deliberate: with limited data, extensive tuning would tend to optimize to idiosyncrasies of the particular split and inflate apparent out-of-sample performance through model selection. For each (embedding, model) pair, we summarize performance by the mean ROC-AUC across the three configurations, and we quantify sensitivity using dispersion statistics computed across these three values: the standard deviation, the absolute range (max–min), and the coefficient of variation

$$\mathrm{CV} = \frac{\mathrm{std}}{|\mathrm{mean}|} \times 100,$$

where the mean and standard deviation are taken across the three ROC-AUC values. CV is used as a normalized measure of sensitivity: it summarizes how much performance varies across configurations relative to the typical performance level of that (embedding, model) pair, which makes stability comparable across specifications.

At the same time, our interpretation is anchored to random performance (ROC-AUC = 0.5):

what matters is whether hyperparameter changes can move a specification close to random or meaningfully away from it. For that reason, we also report the absolute range in ROC-AUC points, which directly captures the worst-case swing across the three configurations. Using both CV and range avoids relying on a purely relative measure when the key question is whether performance shifts are large enough to alter the qualitative conclusion.

A pair is classified as *Robust* if either (i) CV $< 5\%$ or (ii) the absolute range across configurations is below 0.10 ROC-AUC points. We use an OR rule because CV and range capture complementary aspects of stability (overall dispersion versus worst-case swing), and requiring both would be overly restrictive in a small-sample setting. For completeness, pairs with $5\% \leq \text{CV} < 15\%$ are labeled *Moderately Sensitive*, and pairs with CV $\geq 15\%$ are labeled *Highly Sensitive*. This robustness classification is then used as a first-stage screen before the diagnostic categorization and before selecting the final inference model.[1]

Diagnostic

The diagnostic step is designed to summarize the full grid of experiments while keeping the two dimensions that matter in this project: (i) is a specification meaningfully different from random, and (ii) is that conclusion stable across reasonable hyperparameter choices. For each task (classification and regression-direction), and for each (embedding, model) pair, we first compute the mean ROC-AUC across the three fixed configurations (conservative / moderate / aggressive). We then combine this mean performance with the robustness label obtained from the sensitivity analysis.

We use three categories. A pair is labeled *Uninformative* if its mean ROC-AUC is close to random performance, defined as $|\text{mean AUC} - 0.5| < 0.02$. A pair is labeled *Reliable* if it is outside this noise band and classified as `Robust` in the sensitivity analysis. Finally, a pair is labeled *Dangerous* if it is outside the noise band but not robust (i.e., `Moderately Sensitive` or `Highly Sensitive`): in that case, the apparent signal is too dependent on the chosen configuration and could be misleading for inference.[2]

This diagnostic is reported in four summary tables printed by `main.py`: for each task, an *Embedding Diagnostic* table counts how many model families (out of 3) fall into each category for a given embedding, and a *Model Diagnostic* table counts how many embeddings (out of 4) fall into each category for a given model family. In addition, the tables display a *Max AUC* column; in the current implementation this value corresponds to the highest *mean* ROC-AUC among the pairs classified as Reliable within the row group (and is reported as N/A if no Reliable pair exists). The complete pair-level breakdown (mean AUC, distance from random, CV, range, robustness label, and diagnostic category) is saved in `results/diagnostic_breakdown.csv`. To keep the console output readable while still being fully transparent, the composition underlying each frequency count (which specific embeddings/models contribute to each category) is saved separately in `results/diagnostic_composition.csv`.

Grand champion selection

After sensitivity analysis and the diagnostic categorization, we select a single "Grand Champion" specification to use for inference. First, we apply a lexicographic rule within each task: robustness is mandatory, so we keep only (embedding, model) pairs classified as `Robust`. Among

[1] The range cutoff (0.10) is intentionally coarse because only three configurations are evaluated; it is used to flag only extreme hyperparameter fragility rather than to claim tight stability.

[2] Because the held-out test sample is small, ROC-AUC estimates are noisy. The ±0.02 band around 0.5 is therefore not treated as a statistical test; it is a deliberate "practical no-signal zone" that prevents over-interpreting trivial fluctuations around random ordering.

these robust candidates, we then select the pair with the highest mean ROC-AUC across the three preset configurations (conservative / moderate / aggressive). This yields one best robust candidate for classification and one best robust candidate for regression-direction.

Second, because both tasks are evaluated on the same directional ROC-AUC scale, we choose the overall Grand Champion by comparing the mean ROC-AUC of the two task-level winners and selecting the higher one. We still report a contextual metric alongside ROC-AUC (accuracy for classification, directional accuracy for regression) and interpret it relative to a naive benchmark (always predicting the majority class), but ROC-AUC remains the primary selection criterion because it is threshold-independent and directly measures whether the model produces a meaningful ordering beyond randomness.

Finally, once the Grand Champion is selected, it is retrained one last time on the full dataset (all 237 statements) and saved for interactive inference. This is done on purpose: after model selection is completed using the held-out test split, retraining on all available observations maximizes the information used by the final deployed predictor and makes the system benefit immediately from newly appended statements. Importantly, this full-data retraining is performed *after* evaluation and does not change the reported out-of-sample results; it only produces the final inference artifact saved in `results/grand_champion_model.pkl` together with its metadata in `results/grand_champion_config.json`.

4.3 Visualizations

Classification diagnostic (directional prediction). Table 1 and Table 2 summarize the 12 classification specifications (4 embeddings × 3 model families) using the diagnostic categories defined earlier. Each cell is a *count* of (embedding, model) pairs falling into *Uninformative / Dangerous / Reliable*, where *Reliable* means (i) robust to the conservative/moderate/aggressive hyperparameter settings and (ii) meaningfully different from random ordering (outside the $|\text{mean AUC} - 0.5| < 0.02$ noise band). The *Max AUC* column reports the best *mean* ROC-AUC among pairs classified as Reliable in that row (N/A if there is no Reliable pair). Note that ROC-AUC can be below 0.5: this indicates a systematic *inverse* ordering (i.e., swapping the labels or negating the score would yield $1 - \text{AUC}$), so what matters in the diagnostic is distance from 0.5, not only "above 0.5".

By embedding. GTE is the only representation that is Reliable across *all three* classifiers (Logistic Regression, Random Forest, and XGBoost), which makes it the most consistent embedding for directional classification. FinBERT is Reliable for Logistic Regression and Random Forest but becomes Uninformative for XGBoost, suggesting that the finance-adapted representation does help, but not uniformly across model families. For BERT, the Reliable outcomes come from Random Forest and XGBoost, while Logistic Regression is Uninformative, indicating that BERT's signal (when present) is not captured by a purely linear decision boundary. Finally, TF–IDF produces no Reliable specifications: Logistic Regression and Random Forest are classified as Dangerous (signal appears but is not robust to hyperparameters), and XGBoost is Uninformative, consistent with sparse bag-of-words features being unstable at this sample size.

By model family. Random Forest is the most consistently Reliable classifier: it is Reliable under three embeddings (BERT, FinBERT, GTE) and only fails under TF–IDF, which aligns with bagging providing stability in a small-sample, high-dimensional setting. Logistic Regression is mixed: it is Reliable with FinBERT and GTE, but Uninformative with BERT and Dangerous with TF–IDF, highlighting its sensitivity to representation choice. XGBoost is Mostly reliable: it is Reliable with BERT and GTE, but Uninformative with FinBERT and TF–IDF. For full transparency, the exact composition behind each count (which specific models/embeddings populate each category) is reported in Appendix Additional Figures (and saved programmatically in `results/diagnostic_composition.csv`).

Table 1: Classification diagnostic by embedding (frequency summary)

Embedding	Uninformative	Dangerous	Reliable	Max AUC
GTE	0	0	3	0.534
FinBERT	1	0	2	0.524
BERT	1	0	2	0.476
TF–IDF	1	2	0	N/A

Table 2: Classification diagnostic by model (frequency summary)

Model	Uninformative	Dangerous	Reliable	Max AUC
Random Forest	0	1	3	0.523
Logistic Regression	1	1	2	0.533
XGBoost	2	0	2	0.534

Regression diagnostic (direction extracted from predicted returns). Table 3 and Table 4 summarize the 12 regression specifications (4 embeddings × 3 model families) under the same diagnostic logic as for classification, but with *directional* evaluation: each regressor outputs a continuous next-day return prediction, which is treated as a score for ROC-AUC (higher predicted return ⇒ higher confidence in "up"). This keeps the evaluation consistent with the project's goal (detect any residual directional ordering) and avoids over-interpreting noisy return magnitudes at a one-day horizon. As before, each cell is a *count* of pairs in *Uninformative / Dangerous / Reliable*, and *Max AUC* is the best *mean* ROC-AUC among Reliable pairs in that row.

By embedding. TF–IDF is Uninformative for all three regressors, which reinforces the conclusion that sparse bag-of-words features do not yield usable out-of-sample directional ordering in this small corpus. Among transformer representations, FinBERT and GTE each produce exactly one Reliable, one Dangerous, and one Uninformative specification, i.e., they contain some exploitable signal but it is not consistently stable across model families. BERT yields one Reliable and two Uninformative outcomes, with no Dangerous cases; in other words, when BERT shows signal it tends to be stable (robust), but it is absent more often. Note that the diagnostic is based on distance from 0.5 on the ROC-AUC scale: a Reliable mean AUC can be below 0.5, which indicates a stable *inverse* ordering (it would become $1 - \text{AUC}$ if the score sign were flipped). What matters for the diagnostic is whether the ordering is systematically different from random and robust to hyperparameter changes.

By model family. Random Forest is again the most robust choice: it produces Reliable outcomes under two embeddings (FinBERT and GTE) and is otherwise Uninformative, with no Dangerous cases. XGBoost yields one Reliable outcome (under BERT) and is Uninformative under the remaining embeddings, suggesting that boosting does not consistently extract directional signal from return regression in this dataset. Elastic Net is the least suitable here: it has no Reliable outcomes and appears either Uninformative or Dangerous, meaning any apparent signal is unstable across its regularization settings. For the exact mapping of each count to its contributing embeddings/models, in the appendix, Additional Figures reports the full composition (also saved in `results/diagnostic_composition.csv`).

Table 3: Regression (directional) diagnostic by embedding (frequency summary)

Embedding	Uninformative	Dangerous	Reliable	Max AUC
FinBERT	1	1	1	0.552
GTE	1	1	1	0.530
BERT	2	0	1	0.479
TF–IDF	3	0	0	N/A

Table 4: Regression (directional) diagnostic by model (frequency summary)

Model	Uninformative	Dangerous	Reliable	Max AUC
Random Forest	2	0	2	0.552
XGBoost	3	0	1	0.479
ElasticNet	2	2	0	N/A

5 Discussion

This project asks a deliberately hard question: whether the text of FOMC statements contains any residual information about next-day S&P 500 direction. Given market efficiency and the short horizon, any signal—if it exists—should be weak, unstable, and easy to overstate. The evaluation framework (directional ROC-AUC + robustness screening) is therefore designed to separate *consistent* ordering information from results that appear only under a lucky configuration.

Main findings. Across the full grid of experiments, transformer embeddings dominate TF–IDF in terms of stability. In the classification diagnostic, GTE is categorized as *Reliable* for all three model families (3/3), while FinBERT and BERT are reliable for two out of three models. TF–IDF has no reliable specification and is frequently flagged as *Dangerous*, meaning it can look informative in isolated cases but does not behave consistently once hyperparameters move. The regression-direction diagnostic is more mixed (as expected), yet the strongest robust result emerges from **Random Forest + FinBERT** with a mean ROC-AUC of **0.5519**.

Interpretation of the magnitude. AUC values around 0.53–0.55 are modest. This is not a failure: it is exactly what one should expect if markets are close to efficient and the label (next-day direction) is dominated by noise. In that setting, the relevant question is not whether we can achieve high accuracy, but whether the model produces a *non-random ranking* of up vs. down outcomes. That is what ROC-AUC captures. The results suggest that some representations (notably GTE for classification and FinBERT for regression-direction) extract a small but reproducible directional ordering from the statement text.

Why accuracy can look worse than a naive baseline. The Grand Champion (regression + Random Forest + FinBERT) reports a mean directional accuracy of 49.5%, below the naive baseline of 51.4% (always predicting "Down"). This is not contradictory with an AUC above 0.5. Directional accuracy depends on a *single* threshold (here, the sign threshold at 0), while AUC evaluates *all* thresholds and only cares about ranking. A model can rank up-days above down-days better than random (AUC > 0.5) while still making suboptimal hard decisions at the default cutoff, especially when the class balance is slightly asymmetric. In other words, the model may carry weak ordering information without being well calibrated for binary decisions at threshold 0.

Linear vs. non-linear models. The model ladder also behaves as expected. Regularized linear baselines (Logistic Regression / Elastic Net) provide stable reference points in high-dimensional embedding spaces, but the best-performing robust specifications typically involve tree-based methods. This is consistent with the hypothesis that market reactions can depend on interactions and non-linear combinations of wording and context that linear models cannot represent directly. That said, the gap between the best linear and best non-linear robust candidates is not huge in absolute terms, reinforcing the idea that any exploitable signal in this setting is weak.

Limitations and threats to validity. The most important limitation is statistical: the dataset is small (237 statements; 72 observations in the test window), so performance estimates have wide uncertainty bands and are sensitive to regime changes. The chronological split mitigates look-ahead bias, but it also means the model is asked to generalize across decades of evolving Fed communication style and market structure. A second limitation is economic: next-day close-to-close returns aggregate many confounding shocks unrelated to the statement itself. A cleaner event-study design would focus on a narrow window around the release time (intraday) and/or incorporate market expectations (surprises) to isolate the component attributable to the statement. Finally, the project evaluates representations as frozen feature extractors; fine-tuning is intentionally avoided to reduce overfitting risk, but it may also leave performance on the table.

Technology diffusion and time mismatch of representations. The out-of-sample ROC-AUC estimates are computed correctly given the implemented procedure, but they should be interpreted with caution because the text representations are generated using modern pre-trained embedders (e.g., GTE, FinBERT) on statements released in earlier years, including the test period. This creates a *time mismatch*: at the time those statements were released, such embedding technology did not exist (or was not widely deployable), whereas our evaluation implicitly assumes today's representation quality is available for the entire historical sample. As a result, the reported AUC should be read as a retrospective measure of whether *modern* NLP tools can extract directional information from historical FOMC communication, not as performance that would necessarily have been achievable in real time at those dates. Moreover, as these technologies diffuse, any exploitable signal may be incorporated into prices faster, and the relationship can weaken over time due to competitive arbitrage, evolving Fed communication, and changing market reaction dynamics.

What the results do and do not imply. The project provides evidence that certain embedding/model combinations yield a small, robust directional ordering beyond randomness, and that TF–IDF is generally not reliable in this low-sample setting. However, the results do *not* imply an economically meaningful trading strategy as-is: AUC improvements are small, accuracy is not consistently above naive baselines, and no transaction-cost or risk-adjusted performance is assessed. The correct takeaway is methodological: the diagnostic + sensitivity framework helps identify which representations are stable enough to interpret, and it highlights where apparent "signal" is likely an artifact of configuration choice.

Concrete next steps. The most valuable extensions would be (i) intraday event windows around statement release, (ii) rolling/expanding-window evaluation to quantify time-variation in performance, (iii) confidence intervals for AUC via bootstrap to communicate uncertainty explicitly, and (iv) threshold calibration (or asymmetric loss) if the goal shifts from detecting ranking information to making hard directional decisions.

6 Conclusion and Future Work

6.1 Summary

This project tests whether FOMC statement text contains residual information about next-day S&P 500 direction, by comparing four embeddings (TF–IDF, BERT, FinBERT, GTE) combined with standard ML models under a controlled sensitivity protocol. Across specifications, TF–IDF is largely uninformative or unstable, whereas transformer embeddings more often yield reliable (signal + robustness) performance. Using a robustness-first selection rule, the strongest directional ranking signal is obtained with the regression-direction specification Random Forest + FinBERT (mean ROC-AUC 0.552), while the best classification-direction specification is XGBoost + GTE (mean ROC-AUC 0.534). Overall, the results support the presence of weak but measurable historical ranking ability in statement language when modern representations are used, while also showing that robustness constraints materially affect which conclusions are defensible.

6.2 Future Directions

Event timing and return definition. This project uses next-day close-to-close returns, which is simple and reproducible but not tightly aligned with how markets react to scheduled news. A natural extension is to evaluate reactions on an event-study timeline: for example, measuring intraday returns from the statement release timestamp to the close, or from the release to the next trading open, rather than using close-to-close windows that mix the announcement effect with unrelated overnight information. This would better isolate the information content of the FOMC communication and reduce label noise caused by other news arriving between closes.

Multimodal information from press conferences. Another promising direction is to move beyond the written statement and analyze the live diffusion of the FOMC press conference. The conference contains additional signals not present in the text: prosody and voice features (pace, pitch, pauses, stress) and visual non-verbal cues (facial expressions, gaze, micro-movements). A multimodal pipeline could combine (i) transcript embeddings, (ii) audio features (e.g., pitch contours, speech rate, silence patterns), and (iii) video features (e.g., facial action units or simple expression descriptors) into a single predictive representation. The main constraint is data availability and consistency: full high-quality audio/video may not exist for early years, and recording/format standards change over time, which would require careful sample construction and robustness checks.

Calibrating diagnostic thresholds. The diagnostic framework relies on explicit cutoffs: a noise band around random performance ($|\text{mean AUC} - 0.5| < 0.02$) to label pairs as uninformative, and stability thresholds based on CV and absolute range (e.g., CV $< 5\%$ or range < 0.10) to label pairs as robust. In the current project these values are intentionally simple and transparent heuristics, chosen to keep the methodology interpretable in a small-sample setting and to prevent over-reading tiny performance differences. A natural extension is to *calibrate* these thresholds using data-driven uncertainty estimates rather than fixed constants. For example, one can estimate the sampling variability of ROC-AUC on the held-out test set via bootstrap confidence intervals (or a permutation test) and define the "noise band" as the smallest deviation from 0.5 that is statistically distinguishable from random ordering at a chosen confidence level. Similarly, stability thresholds can be calibrated by running the same three-configuration sensitivity analysis under a permutation null (randomly permuting labels) to measure how large CV and range values can become purely by chance. This would convert the current cutoffs into empirically justified decision rules while preserving the overall logic of the approach (robustness first, then performance).

References

1. Hansen, S. & McMahon, M. (2016). *Shocking Language: Understanding the Macroeconomic Effects of Central Bank Communication.* Journal of International Economics, 99, 114–133.

2. Lucca, D. O. & Moench, E. (2015). *The Pre-FOMC Announcement Drift.* Journal of Finance, 70(1), 329–371.

3. Federal Open Market Committee (FOMC). Official policy statements. Federal Reserve Board. Available at: `https://www.federalreserve.gov/monetarypolicy/fomccalendars.htm`.

4. Tasca, P. (2020). *FOMC statement scraping repository.* GitHub repository. Available at: `https://github.com/vtasca/fed-statement-scraping`.

5. Federal Open Market Committee (FOMC). *Historical FOMC statements repository.* GitHub repository. Available at: `https://github.com/fomc/statements`.

A Additional Figures

Table 5: Classification – diagnostic by embedding (composition)

Group	Uninformative items	Dangerous items	Reliable items
TF–IDF	`XGBoost`	`Logistic Regression` Random Forest	-
BERT	`Logistic Regression`	-	`Random Forest` XGBoost
FinBERT	`XGBoost`	-	`Logistic Regression` Random Forest
GTE	-	-	`Logistic Regression` Random Forest XGBoost

Table 6: Classification – diagnostic by model (composition)

Group	Uninformative items	Dangerous items	Reliable items
Logistic Regression	`BERT`	`TF-IDF`	`FinBERT` GTE
Random Forest	-	`TF-IDF`	`BERT` FinBERT GTE
XGBoost	`FinBERT` TF–IDF	-	`BERT` GTE

Table 7: Regression – diagnostic by embedding (composition)

Group	Uninformative items	Dangerous items	Reliable items
TF–IDF	`ElasticNet` Random Forest XGBoost	-	-
BERT	`ElasticNet` Random Forest	-	`XGBoost`
FinBERT	`XGBoost`	`ElasticNet`	`Random Forest`
GTE	`XGBoost`	`ElasticNet`	`Random Forest`

Table 8: Regression – diagnostic by model (composition)

Group	Uninformative items	Dangerous items	Reliable items
ElasticNet	`BERT` TF–IDF	`FinBERT` GTE	-
Random Forest	`BERT` TF–IDF	-	`FinBERT` GTE
XGBoost	`FinBERT` GTE TF–IDF	-	`BERT`

B Code Repository

GitHub Repository: `https://github.com/laraouiaymen-cmd/fomc-statements-analysis`

Repository structure (key files)

```
project_root/
main.py
requirements.txt
README.md
src/
data_loader.py
evaluation.py
model_selection.py
models.py
inference.py
embedding_cache.py
embedder_tfidf.py
embedder_bert.py
embedder_finbert.py
embedder_gte.py
data/
update_fomc_statements.py
prepare_processed_data.py
data/
interim/                # one .txt per statement (YYYYMMDD.txt)
processed/
fomc_statements.csv   # final dataset used by main.py
embeddings_cache/       # created on first run (may be absent initially)
results/                  # generated outputs (CSV tables + saved champion model)
```

Main entry point: `main.py` runs the full experiment end-to-end and writes outputs to `results/`.

How to reproduce the results

1. Ensure the final dataset exists at `data/processed/fomc_statements.csv`.
2. Run the full experiment:

   ```
   python main.py
   ```

3. Outputs are written to `results/`, including:
   - `experiment_results.csv` (all runs),
   - `diagnostic_breakdown.csv` and `diagnostic_composition.csv`,
   - `grand_champion_model.pkl` and `grand_champion_config.json`.

Notes on runtime. Runtime is dominated by (i) loading transformer weights (BERT/FinBERT/GTE) and (ii) computing embeddings. On the first run, if the pretrained weights are not already cached on the machine, Hugging Face will download them once; this can take from a few minutes to tens of minutes depending on the internet connection. After the weights are available locally, embedding computation remains the main cost (performed in mini-batches on CPU). Subsequent runs reuse `data/embeddings_cache/` when the dataset size is unchanged.

Optional: update the dataset and rebuild the processed CSV

The repository supports an *optional* dataset update step to incorporate newly released FOMC statements. **This will change the sample and therefore the results compared to the frozen snapshot used in this report.**

Recommended (single command). Run the full pipeline with the update step enabled:

```
python main.py --update-data
```

Alternative (two explicit steps). If you prefer to update data manually, you can run the two scripts separately:

1. Download missing statements from the Federal Reserve website and store extracted text in `data/raw/` (HTML) and `data/interim/` (extracted text):

   ```
   python src/data/update_fomc_statements.py
   ```

2. Recompute `data/processed/fomc_statements.csv` (clean text + next-day S&P 500 return via `yfinance`):

   ```
   python src/data/prepare_processed_data.py
   ```

Optional: interactive inference

After running `main.py`, the program launch an interactive prompt to classify custom text using the saved "Grand Champion" model (stored in `results/`). This is triggered at the end of `main.py` when answering `"y"` to the prompt requesting live predictions.

C Helper Tools

ChatGPT: Used strictly as a writing assistant (structure, wording, grammar, and LaTeX phrasing) and for brainstorming.

GitHub Copilot: Used as a coding assistant to speed up implementation work (debugging, small refactors, and readability improvements). In particular, it was used to help add or improve **docstrings, type hints, and short inline comments** so that key functions, variables, and constants are self-explanatory when first introduced (e.g., train/test split parameters, batching settings for transformer embedders, caching/invalidation logic, and diagnostic thresholds used in the evaluation). It was also used to improve the readability of `main.py` outputs by adding clear checkpoints and status messages. All Copilot suggestions were reviewed, edited when necessary, and validated by running the full pipeline; the final code and all methodological choices remain my responsibility.