Chapter 9

Leveraging LLMs for Sentiment Analysis in Trading

In this chapter, we pivot from our previous discussions, where we focused on studying different generative model families in isolation, to explore and build a practical financial application that combines these models. The goal is to create a system that takes speech data as input and outputs trading signals. This system leverages powerful models, such as transformer models for sequence-to-sequence tasks to convert audio signals to text, and transformer models in Large Language Models (LLMs) to assign sentiment scores to text. We will apply these techniques to analyze the sentiment of Federal Reserve (Fed) press conference speeches and generate potentially profitable trading signals.

9.1 Sentiment Analysis in Fed Press Conference Speeches Using Large Language Models

Is it possible to use sentiment analysis on Federal Reserve press conference speeches, computed using LLMs, to generate predictive trading signals?

Let's imagine a scenario where we are watching US Federal Reserve Chair Jerome Powell hold a press conference on November 2, 2022, where he is discussing the latest Fed rate hike and communicating the committee's current economic outlook, as shown in Figure 9.1 (WGN News, 2022).

Figure 9.1 Fed Chair Powell discusses latest Fed rate hike. *Source:* WGN News (2022).

As traders, we might wonder how to use the information provided in such a speech to make informed trading decisions. If you were someone like George Soros, known for his exceptional trading skills and remarkable track record in macro trading, you might be able to filter the relevant information from the speech, converting it into insights that you can combine with other data, such as market data, to predict the market's next moves easily.

However, for those of us without Soros' insights, a more quantitative approach could be helpful. This approach requires converting segments of the video into numerical values from which we can generate buy or sell signals. Our approach has been inspired by reports on JPMorgan AI's analysis of Fed speeches for trading signals from *Bloomberg News* (2023) and *Fortune* (2023), as well as an OpenAI Whisper tutorial on Fed Speech Transcription from Part Time Larry (2022).

There are several ways to design a system with the goal of translating video content into actionable trading signals. Here, we attempt to provide a solution based on a cascade approach that involves a series of subsystems for processing different datatypes. The whole system takes a segment of the Fed press conference video as input and outputs a numerical value representing the sentiment score of that segment. The design of these subsystems will involve the following:

1. Extracting speech from the video of the Fed press conference
2. Converting speech segments to text
3. Converting text to numerical values (sentiment scores) that can generate relevant signals

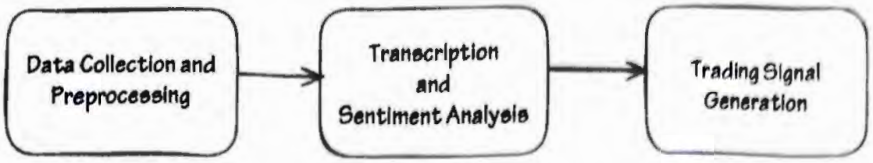

Figure 9.2 System block diagram.

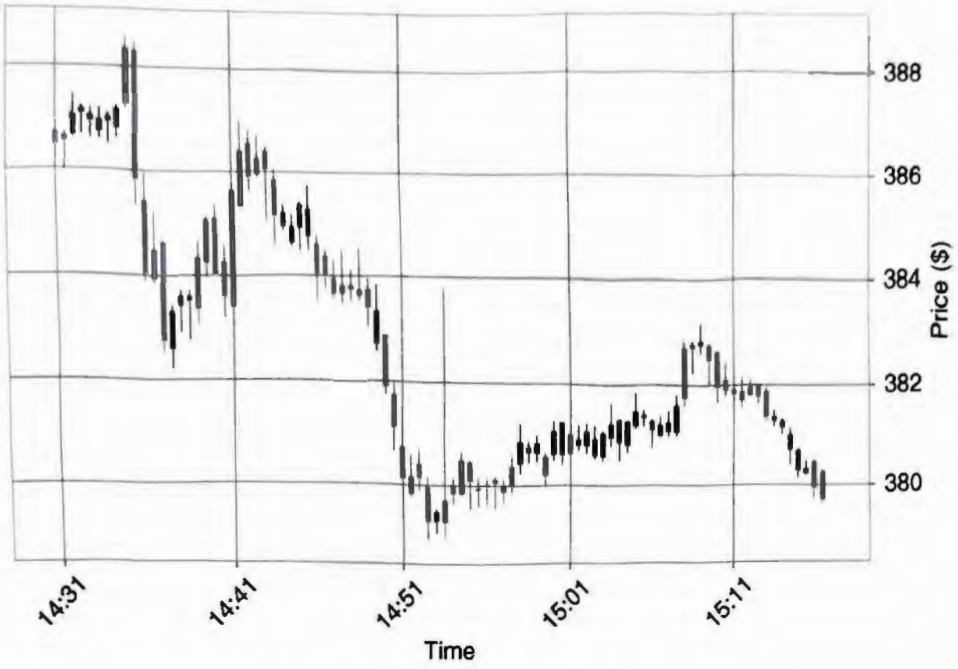

Figure 9.3 SPY Price series during Fed press conference.

The system's block diagram illustrating this process is in Figure 9.2.

To explore this, we will take a step back and simulate making decisions based on the press conference and related market data available to us.

Let's say we want to design a trading strategy to trade the SPY in 30-second intervals. In Figure 9.3, we illustrate the SPY price series during the November 2, 2022, Fed press conference, from 14:30 to 15:30 in 30-second intervals. Ideally, we would like to find some kind of correlation between what Jerome Powell is saying at or up to that particular moment and the subsequent SPY price moves.

For every 30-second trading bar of SPY data, we would do the following:

1. Extract audio from the video up to that particular bar.
2. Perform speech-to-text conversion.
3. Perform Sentiment Analysis based on text.
4. Generate signals to make buy and sell decisions.

This process will enrich the SPY open-high-low-close (OHLC) price series with the sentiment scores of Powell's speech during each 30-second interval, making it easier

data_with_sentiment.head(10)

timestamp	text	video_id	sentiment_score	est_timestamp	open	high	low	close	return
2022-06-15 14:31:00-04:00	Good afternoon. I will begin with	1	0.116886	2022-06-15 13:31:00-05:00	374.40	374.67	373.66	373.94	0.000053
2022-06-15 14:31:30-04:00	Good afternoon. I will begin with one overarc...	1	0.360220	2022-06-15 13:31:30-05:00	373.80	374.15	373.47	373.96	-0.001685
2022-06-15 14:32:00-04:00	one overarching message. We at the Fed unders...	1	0.313680	2022-06-15 13:32:00-05:00	374.05	374.07	373.00	373.33	-0.001848
2022-06-15 14:32:30-04:00	If we were to have a sustained period of stro...	1	0.115033	2022-06-15 13:32:30-05:00	373.18	373.30	372.55	372.64	-0.000510
2022-06-15 14:33:00-04:00	rate will be appropriate. In addition, we are...	1	0.034392	2022-06-15 13:33:00-05:00	372.49	372.87	372.12	372.45	0.002121
2022-06-15 14:33:30-04:00	with consumption spending remaining strong. I...	1	-0.311607	2022-06-15 13:33:30-05:00	372.41	373.50	372.30	373.24	0.000831
2022-06-15 14:34:00-04:00	have marked down their projections for economi...	1	-0.186132	2022-06-15 13:34:00-05:00	373.06	373.61	372.90	373.55	0.000428
2022-06-15 14:34:30-04:00	robust. Improvements in labor market conditio...	1	0.622128	2022-06-15 13:34:30-05:00	373.59	374.36	373.45	373.71	-0.000722
2022-06-15 14:35:00-04:00	on wages and prices. The median projection in...	1	0.380036	2022-06-15 13:35:00-05:00	373.69	374.01	373.25	373.44	0.001446
2022-06-15 14:35:30-04:00	or prices rose 4.9%. In May, the 12-month che...	1	-0.127001	2022-06-15 13:35:30-05:00	373.60	374.04	373.34	373.98	0.001230

Figure 9.4 Enriched price series.

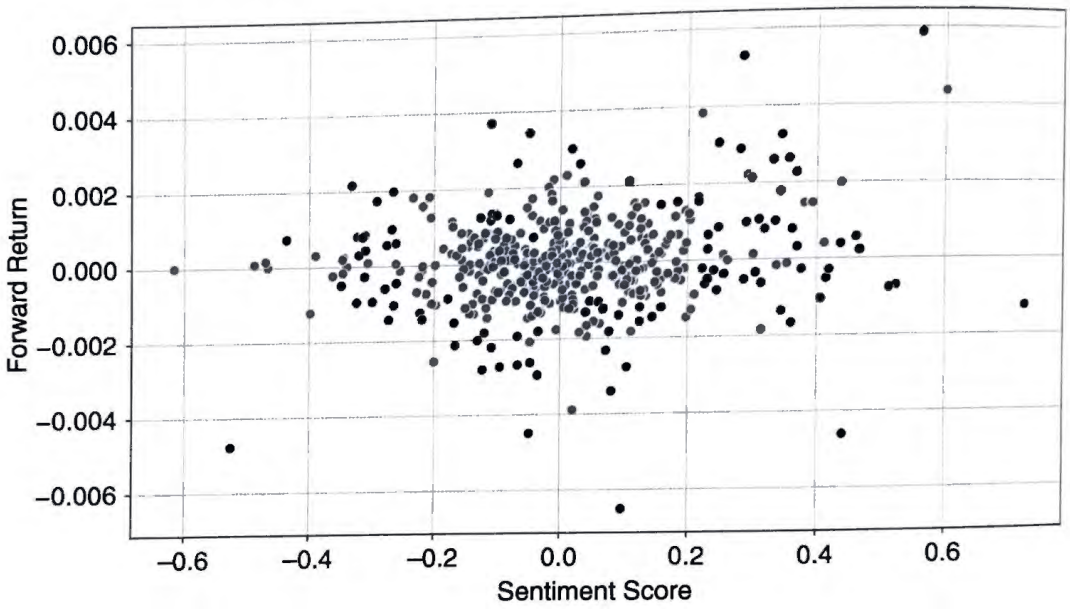

Figure 9.5 Scatter plot of sentiment signal vs forward returns.

to quantify the predictive power of the signal. The desired tabular data format is illustrated in Figure 9.4.

With this enriched data, we can investigate whether there is a statistical relationship between the computed sentiment scores and future returns. In Figure 9.5, we illustrate a scatter plot for a few Fed press conferences videos that took place in 2022 and 2023, where the x-axis represents the sentiment scores and the y-axis represents the SPY forward returns over the next 30 seconds.

In the rest of the section, we will describe in more detail the main components of the system, how they work, and how to put them together. In our experience, using a limited sample size of videos, we found a significant Pearson correlation of 14.14% between the sentiment signal and forward SPY return, with a p-value of 0.199%.

9.2 Data: Video + Market Prices

In this use case, we utilize video data, extract the corresponding audio, process it into text, converting the text into numerical values, and finally joining these numerical values with market price data for model construction and evaluation. This approach leverages the unique insights offered by alternative data sources to enhance trading strategies and market predictions.

As noted by López de Prado (2018):

> Alternative data offers the opportunity to work with truly unique, hard-to-process datasets. Remember, data that is hard to store, manipulate, and operate is always the most promising. You will recognize that a dataset may be useful if it annoys your data infrastructure team. Perhaps your competitors did not try to use it for logistic reasons, gave up midway, or processed it incorrectly.

9.2.1 *Collecting Audio Data*

For collecting audio data from YouTube, we use the pytube tool, which is a lightweight, Pythonic, and command-line utility for downloading YouTube Videos.

Downloading YouTube videos or audios can be easily achieved using pytube. Once the `url` of the desired video is defined, you can use the library as shown below:

```
from pytube import YouTube
video_url = 'http://youtube.com/watch?v=9bZkp7q19f0'
yt = YouTube(video_url)
yt.streams
  .filter(progressive=True, file_extension='mp4')
  .order_by('resolution')
  .desc()
  .first()
  .download()
```

where in the code snippet specified:

- We first import the `pytube` library.
- We specify the video url `video_url`.
- We create a `YouTube` object.
- We then filter `mp4` file streams, order them by resolution, and download the highest resolution file.

9.3 Speech-to-text Conversion

9.3.1 *Whisper Model*

> "We've trained and are open-sourcing a neural net called Whisper that approaches human level robustness and accuracy on English speech recognition." —Introducing Whisper (https://openai.com/index/whisper/)

Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	tiny.en	tiny	~1 GB	~32x
base	74 M	base.en	base	~1 GB	~16x
small	244 M	small.en	small	~2 GB	~6x
medium	769 M	medium.en	medium	~5 GB	~2x
large	1550 M	N/A	large	~10 GB	1x

Figure 9.6 Available models and languages.

Whisper, Radford et al. (2022), is an automatic speech recognition (ASR) system trained on 680,000 hours of multilingual and multitask supervised data collected from the web. This large-scale dataset enables Whisper to transcribe in multiple languages and translate from multiple languages to English. OpenAI has open-sourced Whisper under the MIT License. According the paper results, Whisper is comparable to standard benchmark and often competitive with models that have been finetuned for specific datasets.

In their paper, OpenAI describes various methods for evaluating and collecting data. The data collected for Whisper is weakly supervised, meaning it is not annotated by professional transcribers but instead has labels collected from the web. To build a dataset for model training consisting of audio paired with transcripts, the authors performed a great deal of data engineering work, including multiple preprocessing and heuristic filtering steps for tasks like input language detection and transcript alignment, which are detailed in the paper. Due to the vast amount of data collected, the dataset is highly diverse, encompassing a wide range of speakers, languages, and environments. More details about the Whisper architecture can be found in Radford et al. (2022).

Whisper processes the input audio by splitting it into 30-second chunks and converted into a log-magnitude Mel spectrogram representation, which serves as the input to the encoder. The decoder is trained to predict the corresponding text caption, intermixed with special tokens that direct the model to perform tasks such as language identification, phrase-level timestamps, multilingual speech transcription, and translation to English (see "Introducing Whisper" or the paper for more details).

According to the Whisper model card available in the official OpenAI GitHub repository for Whisper (https://github.com/openai/whisper), it comes in five model sizes, offering different speed and accuracy trade-offs. Four of these models are English-only versions. Figure 9.6 lists the names of the available models and their approximate memory requirements and inference speed relative to the large model.

According to OpenAI experiments, the .en models for English-only applications tend to perform better, especially for the tiny.en and base.en models. See their GitHub cited earlier for more details.

9.3.1.1 Python Usage. We perform transcription using the forementioned Python Whisper library. Following is an example of how to transcribe audio using Whisper:

```
import whisper
audio_input = "audio.mp3"
model_name = "base"
model = whisper.load_model(model_name)
result = model.transcribe(audio_input)
print(result["text"])
```

In this example:

- We first import the Whisper library.
- We specify the audio input file `audio.mp3` and the model we want to use `base`.
- We load the specified Whisper model using `whisper.load_model(model_name)`.
- We transcribe the audio file using `model.transcribe(audio_input)`.
- Finally, we print the transcribed text using `print(result["text"])`.

Internally, the `transcribe` method reads the entire file and processes the audio with a sliding 30-second window, performing autoregressive sequence-to-sequence predictions on each window.

9.3.2 *Whisper on FED Speech Audio Data*

As we have seen, whisper models are trained on 30-second input chunks and cannot process longer audio input at once. This limitation can pose challenges for real-world applications. The authors of the Whisper paper propose a solution for transcribing longer audio sequences by consecutively transcribing 30-second audio windows and shifting this window according to the timestamps predicted by the model. They also emphasize the importance of making correct decisions about decoding heuristics to reliably transcribe long audio, as detailed in section 4.5 of their paper, Radford et al. (2022).

This has several implications for our use case. Fed speech conferences typically last between 45 and 60 minutes. Using such long audio inputs means that the timestamps predicted by Whisper might not be accurate, which could potentially introduce data alignment and look-ahead bias issues when merging data by timestamp between audio and price data-and important issue when backtesting trading strategies. The unreliability of the predicted timestamps for long audio files is something we have also observed during our experimentation.

To ensure the input audio signals are correctly aligned with the corresponding market time-series data, we avoid relying on the transcription timestamps produced by Whisper, whose accuracy decreases as the audio length increases. Instead, we perform audio segmentation (described in Section 9.3.3) beforehand, creating 60-second input chunks sampled every 30 seconds. This process generates 60-second "audio bars" every 30 seconds. This method allows us to determine the start and end timestamps of the corresponding audio, and therefore its transcript, without relying on Whisper for this purpose. Additionally, we introduced an embargo period in the audio series. This means joining price data timestamps with audio/transcribe timestamps 15 seconds earlier, thereby reducing the risk of look-ahead bias even further. While all these design choices

limit the amount of information used to compute the sentiment score in subsequent subsystems, they significantly reduce the risk of introducing look-ahead bias.

9.3.3 Audio Segmentation

For the forementioned audio segmentation, we use FFmpeg (https://ffmpeg.org), which is cross-platform solution to record, convert and stream audio and video.

You can use ffmpeg-python: Python bindings for FFmpeg (https://github.com/kkroening/ffmpeg-python), which are Python wrappers for FFmpeg. For example, in the code snippet below we can see how to convert between different audio formats:

```
import ffmpeg

# Define input and output files
input_file = 'input_video.mp4'
output_file = 'output_audio.wav'

# Extract audio from video
ffmpeg.input(input_file).output(output_file).run()
```

For audio segmentation, you can specify the start time and duration. This allows you to create specific "audio bars" of defined lengths, containing pure alternative data. These can then be combined with market data to create enriched datasets for model training and evaluation.

```
# Define start time and duration in seconds
start_time = 30
duration = 60

# Extract a specific segment
ffmpeg.input(input_file, ss=start_time, t=duration).output(output_file).run()
```

Results on actual FED data are shown in Figure 9.7.

```
# load the whisper model
model = whisper.load_model(WHISPER_MODEL_NAME)
```

```
100%|████████████████████████████████| 139M/139M [00:01<00:00, 74.6MiB/s]
```

```
If you are running locally with a recent model GPU, the following cell may give the message that the installed version of PyTorch does not support the CUDA level of your
GPU. You can fix the problem by installing the appropriate version of PyTorch using these instructions: https://pytorch.org/get-started/locally/
```

```
transcript = transcribe(model, cut_details, seed=MAIN_SEED)
print(transcript.head())
```

```
started at 2023-09-29 14:57:46.030204
  0%|          | 0/105 [00:00<?, ?it/s]
ended at 2023-09-29 15:03:34.321123
time elapsed: 0:05:48.290919
                                                text           timestamp
0                 Good afternoon. I will begin with 2022-06-15 13:31:00
1   Good afternoon. I will begin with one overarc... 2022-06-15 13:31:30
2   one overarching message. We at the Fed unders... 2022-06-15 13:32:00
3   if we were to have a sustained period of stro... 2022-06-15 13:32:30
4   rate will be appropriate. In addition, we are... 2022-06-15 13:33:00
```

```
transcript.head()
```

	text	timestamp
0	Good afternoon. I will begin with	2022-06-15 13:31:00
1	Good afternoon. I will begin with one overarc...	2022-06-15 13:31:30
2	one overarching message. We at the Fed unders...	2022-06-15 13:32:00
3	if we were to have a sustained period of stro...	2022-06-15 13:32:30
4	rate will be appropriate. In addition, we are...	2022-06-15 13:33:00

Figure 9.7 Whisper output on FED data.

9.4 Sentiment Analysis

After performing the speech-to-text conversion using `whisper`, the next step is to compute the sentiment of the speech based on the transcribed text. For this task, we use the FinBERT model, Araci (2019), a variant of BERT Devlin et al. (2019) fine-tuned specifically financial data. You might wonder—why use FinBERT for this task instead of the more powerful ChatGPT? One simple reason is cost—both economic and computational. Using a BERT-based model for this demo allows you to work with a widely used industry model that can run on your own computer with minimal resources or with free cloud resources available at the time from Google Cloud. Beyond cost considerations, in the next chapter, we will explore how to customize this model. Specifically, we'll cover how to fine-tune it and optimize it for inference, making it more memory-efficient and faster at runtime—something that would be far more challenging with ChatGPT. With a large proprietary model like ChatGPT, you would have to rely entirely on OpenAI's infrastructure, limiting customization and hands-on learning.

9.4.1 BERT

The BERT model, which stand for Deep Bidirectional Representation from Transformers, was introduced in in 2019 by Devlin et al. (2019).

BERT has been highly influential in both academic research and industrial applications. In industry, Google Search results were powered by a BERT model, eBay's recommender system utilizes BERT, and variants of BERT have been successfully applied in Hamlet's own work at Criteo. Academically, extensive work has been done to improve BERT's performance, scalability, and efficiency. As of now, the BERT paper has been cited more than 119,678 times according to Google Scholar.

One reason BERT has been so influential is its transformation of the typical workflow in NLP, both in industry and academia. The paper demonstrated the power of pre-training BERT plus fine-tuning for a variety of downstream NLP tasks, consistently outperforming state-of-the-art models at the time and proving invaluable for scenarios with limited training data. Unlike traditional NLP approaches at the time of BERT's introduction, which often required training separate models for each task, the pre-training and fine-tuning approach enabled a more generalized and efficient workflow.

The concept of pre-training and fine-tuning, as highlighted in the paper by demonstrating state-of-the-art experimental results across the board at the time, has driven the industry and academia to build better models by increasing the amount of training data and model size. This approach has led to the development of foundational models, which can later be fine-tuned for more specific tasks. Fine-tuning techniques will be covered in the next chapter.

For this chapter, we will describe the main aspects of the BERT model, as FinBERT, which we use for sentiment analysis, is a fine-tuned version of BERT specifically for financial data.

9.4.1.1 BERT Overview. BERT is based on a transformer encoder-only architecture, which we covered in Chapter 5 in the Transformers section. Unlike earlier models like GPT-1 (Radford et al., 2018) or ELMo (Peters et al., 2018), which used left-to-right context or a concatenation of left-to-right and right-to-left contexts computed independently, BERT introduced **bidirectional context**. This means that BERT considers both the left and right context of a word simultaneously. Empirical results have shown that this bidirectional context provides embeddings that are better adapted to the surrounding context. The differences in how these models handle contextual information are illustrated in Figure 9.8.

9.4.1.2 Input/Output Representations. To enable BERT to handle a variety of downstream tasks, Devlin et al. (2019) designed a special input representation. This representation allows BERT to unambiguously represent both single sentences and pairs of sentences with a single token sequence, making it suitable for various downstream tasks. Let's explore how inputs are represented in BERT.

9.4.1.2.1 Input Representations. As discussed in Section 5.3.5, the first step in the NLP pipeline involves a **tokenization** process, which takes a sequence of raw words as input and outputs a sequence of **token IDs**. Once we have this sequence of token IDs, BERT augments the input representation. Let's illustrate this process with two sentences, Sentence A and Sentence B, and see how BERT computes their initial representation and processes them. Using the notation from Devlin et al. (2019):

1. Tokenization
   - Sentence A is tokenized into a sequence of tokens: $\text{Tok}_1, \text{Tok}_2, ..., \text{Tok}_N$, where N is the number of tokens in Sentence A.
   - Sentence B is tokenized into a sequence of tokens: $\text{Tok}_1, \text{Tok}_2, ..., \text{Tok}_M$, where M is the number of tokens in Sentence B.
2. Use of special tokens `[SEP]` and `[CLS]`
   - The tokenized sequences of Sentence A and Sentence B are concatenated and separated by a special token called `[SEP]`. This token helps BERT distinguish between the two sentences when they are combined into a single token sequence.
   - A special classification token `[CLS]` is added to the beginning of the sequence. This token plays a critical role in downstream tasks, as its final representation is treated as the "aggregate sequence representation" of the input; see Devlin et al. (2019).
3. Initial Embeddings: each token is mapped to an initial embedding (embeddings are discussed in more detail in Section 5.3.5)
   - `[CLS]` token is mapped to $E_{[\text{CLS}]}$.
   - `[SEP]` token is mapped to $E_{[\text{SEP}]}$.
   - Sentence A's tokens are associated with embeddings $E_1, ..., E_N$.
   - Sentence B's tokens are associated with embeddings $E'_1, ..., E'_M$.

These steps are part of BERT's input representation process, which is illustrated in Figure 9.9. As an example to understand the dimensions associated with each embedding,

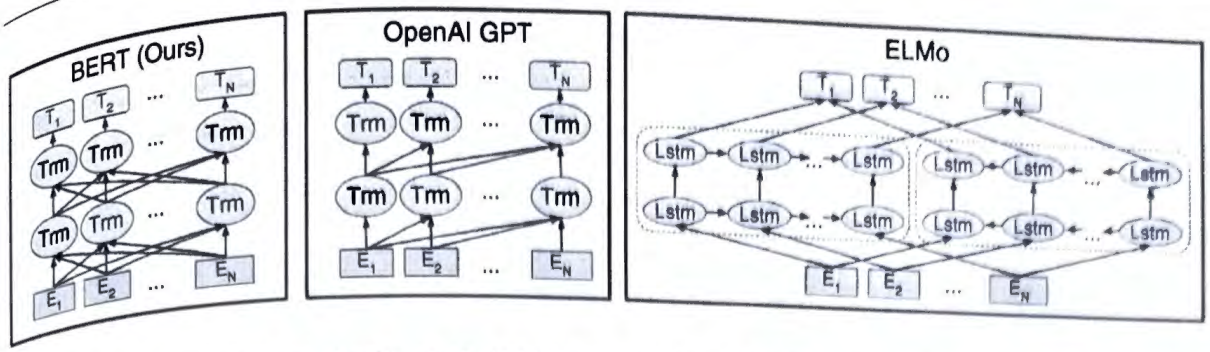

Figure 9.8 Figure 3 from Devlin et al. (2019): Illustration of differences in pre-training model architectures, comparing BERT, OpenAI-GPT, and ELMo OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to- left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

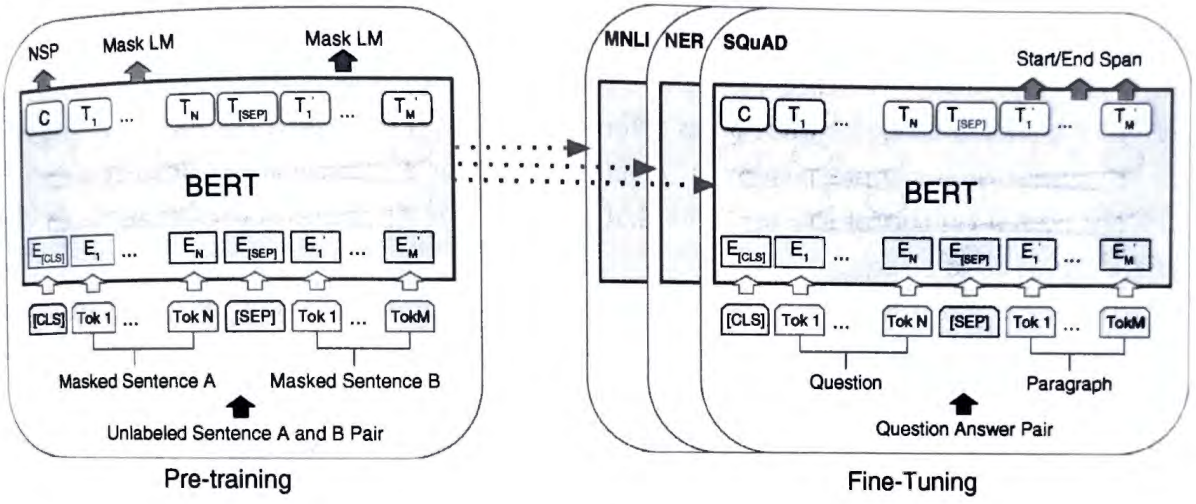

Figure 9.9 Figure 1 from Devlin et al. (2019): Illustration of the input/output BERT representation. Apart from the output layers, the same architecture is used for both pre-training and fine-tuning. The same pre-trained model can be used as an initialization and adapted for multiple tasks, as shown in the image on the right. In the paper, during fine-tuning, all parameters were updated. `[CLS]` is a special symbol added at the beginning of every input example, and `[SEP]` is a special token used to separate pairs of sentences.

the embeddings for each token typically reside in a 768-dimensional hidden space for the BERT-Base configuration or a 1024-dimensional hidden space for the BERT-Large configuration (more details about this are provided in the next section).

9.4.1.2.2 ***Output Representations.*** After the Transformer layers process the input, the initial token embeddings are updated to produce the final hidden states or representations for each token. At the output:

- The final hidden state for the `[CLS]` token is denoted as C.
- The final hidden state for the `[SEP]` token is denoted as $T_{[SEP]}$.
- The final hidden states for tokens in Sequence A and Sequence B are $T_1, ..., T_N$ and $T'_1, ..., T'_M$, respectively.

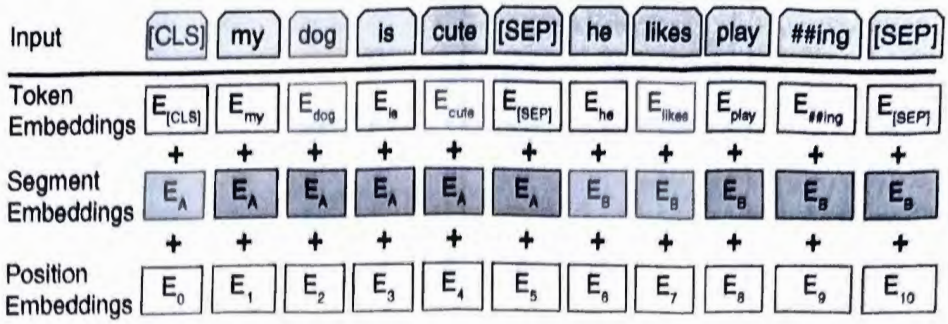

Figure 9.10 Figure 2 from Devlin et al. (2019): Illustration of the BERT input representation. The input embeddings to the Transformer model are the sum of the token embeddings, segment embeddings, and positional embeddings.

This is also visualized in Figure 9.9.

The role of the special token `[CLS]`, short for classification, is of special importance, it is used as the **aggregate representation** of the entire input sequence. Devlin et al. (2019) used C in experiments related to classification tasks showing its efficacy. In fact, FinBERT also uses C to fine-tune the model to perform sentiment analysis. In such task, C is a passed to a classifier, often referred to as the "classification head," to predict the sentiment of the input sequence. (Remember that a classification head is a neural network added on top of the encoder that maps C, the encoded representation, to the desired number of classes.)

Another key feature of BERT is the introduction of a third type of embedding for encoding the input, known as **segment embeddings**. If you recall from our introduction to Transformers in Section 5.3.5, the input embeddings to transformers are typically computed as a combination of **token embeddings** and **positional embeddings**, which are usually combined by addition. In BERT, **segment embeddings** are learned during the training process and used to indicate whether a token belongs to Sentence A or Sentence B. In summary, inputs embeddings are the sum of three components: token embeddings, positional embeddings, and segment embeddings. This is illustrated in Figure 9.10.

This input representation allows BERT to unambiguously represent both a single sentence and pairs of sentences in one token sequence, which enables BERT to be use for downstream task like Question and Answering, where we typically provide the dataset as a collection of (Question, Answer) pairs. Moreover, this representation makes the Next Sentence Prediction (NSP) pre-training objective possible, which we will discuss next.

9.4.1.2.3 Pre-training Objectives. The pre-training objectives of BERT include two tasks:

1. Masked Language Model (MLM) task: In this task, words in a sentence are randomly masked, and the model is trained to predict the missing words. In the BERT paper, 15% of the words are masked. Of these, 80% of the time, the [MASK] token is used, 10% of the time the correct word is used, and the remaining 10% of the time a random word is used.
2. Next Sentence Prediction (NSP): This task involves predicting whether a given sentence B logically follows sentence A. The model is trained on pairs of sentences from a corpus.

The bidirectional context and the segment embeddings introduced by BERT, along with its two pre-training objectives, are considered the main innovations of the BERT work (Devlin et al., 2019).

The data used for pre-training BERT included the BookCorpus (900M words) (Zhu et al., 2015) and English Wikipedia (2,500M words), with some additional pre-processing described in the paper.

BERT comes in two configurations:

- BERT-Base: 12-layer (transformer blocks), 768-hidden dimension, 12 attention-heads, with a total number of parameters equal to 110M
- BERT-Large: 24-layer (transformer blocks), 1024-hidden dimension, 16 attention-heads, with a total number of parameters equal to 340M

The process of pre-training plus fine-tuning BERT for specific downstream NLP applications outperformed state-of-the-art models at the time, including GPT-1 from OpenAI, in all considered NLP tasks.

9.4.1.3 Fine-tuning BERT for Enhanced Financial Sentiment Analysis: Producing FinBERT. Fine-tuning BERT on financial data is crucial because general-purpose sentiment models may not accurately capture the nuances of financial vocabulary and market-specific sentiments. For example, the sentence "Mark left Facebook" might be considered neutral by most sentiment models, but in the financial context, it could significantly impact the market. By fine-tuning BERT on financial texts, we enable the model to learn new concepts, names, and entities specific to finance, resulting in more accurate sentiment analysis. Fine-tuning BERT for multiple tasks is straightforward, as demonstrated by Devlin et al. (2019) in multiple experiments.

As discussed in the previous section, fine-tuning BERT for a classification task can be done by using the final hidden representation of the classification token `[CLS]`, which serves as the "aggregate representation" of the entire sequence. This representation is then fed to a classification head, which outputs the probabilities associated with each sentiment class.

To train the entire model, we simply need a collection of (input, output) pairs. During training, all the model's parameters are updated, including those of BERT and the classification head. In the literature, BERT is often referred to as the "Body" of the network, to distinguish it from the additional layer appended to perform a specific task, such as classification, which is called the "classification head."

In Hamlet's experience, there are variations in what parameters to train or finetune in practice. For certain applications, fine-tuning only the classification head can yield good results. However, this depends on the specific use case and requires empirical validation. As we will see in the next chapter, fine-tuning all the parameters might also be very time-consuming and costly, but it often improves performance. To address this, more efficient fine-tuning methods can be utilized, which will be discussed in Section 10.5.1.

As discussed, fine-tuning for tasks like text classification (e.g., sentiment analysis), the input text is fed into BERT, and the final `[CLS]` token representation is passed into

an output layer for classification. This is the approach followed by the authors of Araci (2019) to produce the FinBERT model for text classification. The authors also experimented with producing a FinBERT for regression, where the main difference from the classification case is the use of a continuous target, an output layer for regression, and a regression loss function, specifically mean squared error. This score can serve as a representation summarizing the processing of relatively complex alternative data in a downstream multi-factor trading strategy, for example. If you want to learn more about applying Transformer-based models for time series, check out Section 5.3.6, where we have already covered this use case.

The datasets used to fine-tune BERT into FinBERT were as follows:

1. TRC2-financial, which is a subset of Reuters' TRC2. It consists of 1.8M news articles that were published by Reuters between 2008 and 2010. These data are used to pre-train BERT. The corpus can be obtained for research purposes by applying here: https://trec.nist.gov/data/reuters/reuters.html.
2. Financial PhraseBank, see Malo et al. (2014), which is the main sentiment analysis dataset used in this paper.
3. FiQA Sentiment, see *WWW '18: Companion Proceedings of the Web Conference 2018* (2018), a dataset created for the WWW '18 conference financial opinion mining and question-answering challenge. It includes 1,174 financial news headlines and tweets with their corresponding sentiment score.

For more details about the datasets, see Araci (2019).

So, let's explore how we can leverage the FinBERT model for our practical application.

9.4.1.4 Using FinBERT. In this experiment, we use FinBERT, which has been trained by Araci (2019) and made available at ProsusAI/finbert (https://huggingface.co/ProsusAI/finbert). FinBERT is based on fine-tuning BERT-Base, so it has a model size of 110M parameters. We have also written some additional code on top of the ProsusAI/finbert to make sentiment evaluation quite easy.

Here, we illustrate how we can make use of this model for computing the sentiment of sentences. Let's start with a simple example.

To compute the sentiment of the sentence "Mark has left Facebook," we can use the following code:

```
# negative
finbert_sa.predict_overall_sentiment('Mark has left facebook',
  finbert_model)
>>> -0.45569813
```

The sentiment score of −0.45569813 indicates a negative sentiment. Similarly, we can compute the sentiment of other sentences. For example, consider the sentences "Jeff bezos acquires Netflix":

```
# seems neutral
finbert_sa.predict_overall_sentiment('Jeff bezos acquires Netflix',
  finbert_model)
>>> -0.0069163926
```

This sentence has a sentiment score of −0.0069163926, which is quite neutral. Let's look at another example, "Business magnate Elon Musk initiated an acquisition of American social media company Twitter":

```
# seems neutral
finbert_sa.predict_overall_sentiment(
  '''Business magnate Elon Musk
initiated an acquisition of American
social media company Twitter''',
  finbert_model)
>>> 0.090285115
```

The sentence has a slightly positive sentiment score of 0.090285115. Finally, let's evaluate the sentiment of a more complex sentence related to the Federal Reserve:

```
# negative
finbert_sa.predict_overall_sentiment(
  '''The Federal Reserve held interest rates steady,
  while also indicating it still expects one more hike
  before the end of the year and fewer cuts than
  previously indicated next year.''',
  finbert_model)
>>> -0.13109717
```

This sentence has a negative sentiment score of −0.13109717.

We can also process multiple sentences at one. For example:

```
sentences = [
    'Mark has left facebook',
    'Jeff bezos adquires Netflix',
    '''Business magnate Elon Musk initiated an
    acquisition of American social media company Twitter''',
    '''The Federal Reserve held interest rates steady,
    while also indicating it still expects one more
    hike before the end of the year and fewer cuts
    than previously indicated next year.'''
]

sentiments = finbert_sa.process_sentences(finbert_model, sentences)

sentiments

>>> array([-4.5569813e-01, -1.7964840e-04,
  9.0285115e-02, -1.3109717e-01], dtype=float32)
```

This array of sentiment scores makes it easy to analyze the sentiment of multiple sentences efficiently.

9.5 Experiment Results

In this section, we describe the results of the model pipeline used to convert FED speech audio data into trading signals. As a reminder, the model pipeline consists of the following steps:

1. Collecting audio from YouTube using pytube.
2. Audio segmentation using ffmpeg.
3. Transcribing the audio bars into text using the Whisper model.
4. Computing sentiment scores for the transcribed text using FinBERT.
5. Generating trading signals based on the sentiment scores, with buy or sell decisions made according to predefined thresholds from the sentiment score distribution.

Price SPY data are sampled at 30-second intervals (30-second bars), and based on sentiment, we attempt to make a trading decision every 30 seconds.

In our test, we collected a total of 5 hours of speech data from the Fed using pytube, corresponding to five different Fed conferences. The audio was downloaded with a specified bit rate (ABR) of 160kbps to ensure high quality.

As previously explained, after downloading the data, the audio was segmented to create 60-second "audio bars," sampled at 30-second intervals, with timestamps delayed by 15 seconds (embargo) to reduce the risk of introducing look-ahead bias. For transcription, we used the Whisper `base` model for its trade-off between speed and accuracy. In our experiments related to the speech processing part, we noticed that the results were sensitive to using larger versions of Whisper for producing the transcriptions, but this did not significantly affect the results. Instead, the quality of the input data had a more important impact. Therefore, we downloaded the highest quality available that was common across all videos.

Once the audio bars were created along with the corresponding transcriptions, we used FinBERT to compute the sentiment score of each audio bar. Trading signals were generated based on the computed sentiment scores. The sentiment score distribution was analyzed, and thresholds were set based on its quantiles. If the sentiment score was above the upper threshold, we generated a buy signal for the SPY; if it was below the lower threshold, a short signal was generated. Otherwise, no action was taken unless a position was already open, in which case we closed it.

A summary of the main steps performed by each subsystem is described here:

1. Data collection and preprocessing
   - Collected 5 hours of FED speech data
   - Downloaded audio with bit rate (ABR) of 160 kbps
   - Segmented the audio into 60-second bars with a 30-second sampling interval and a 15-second embargo
2. Transcription and sentiment analysis
   - Used the Whisper `base` model for transcription
   - Applied FinBERT to compute sentiment scores for each audio bar

3. Trading signal generation
   - Analyzed the sentiment score distribution to set thresholds
   - Generated buy or sell signals based on sentiment scores

Figure 9.11 shows the SPY forward return and sentiment score time series produced by the system:

In Figure 9.12 we present the scatter plot between the sentiment score and the SPY forward returns, which reveals a positive correlation between them.

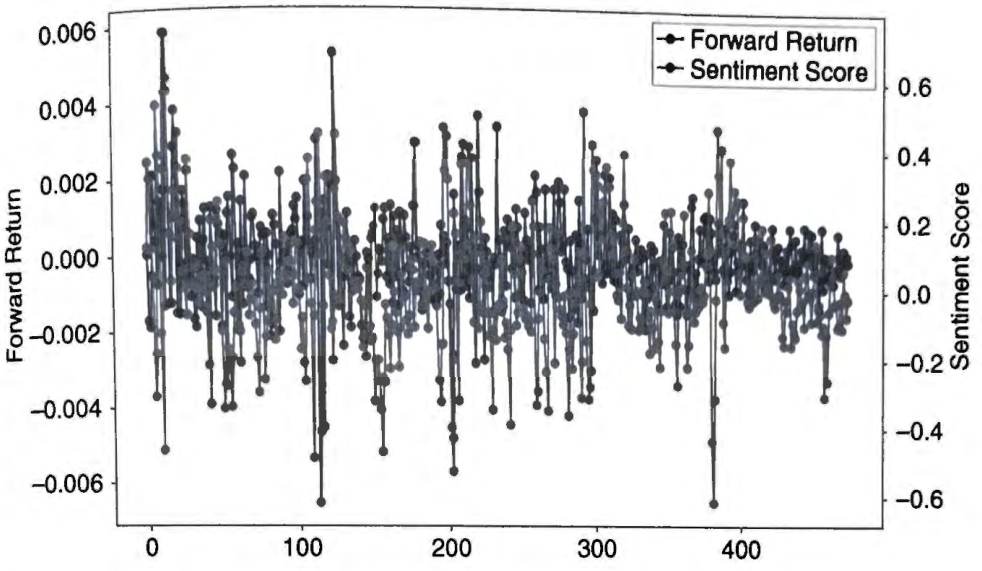

Figure 9.11 Time-series sentiment signal and forward returns.

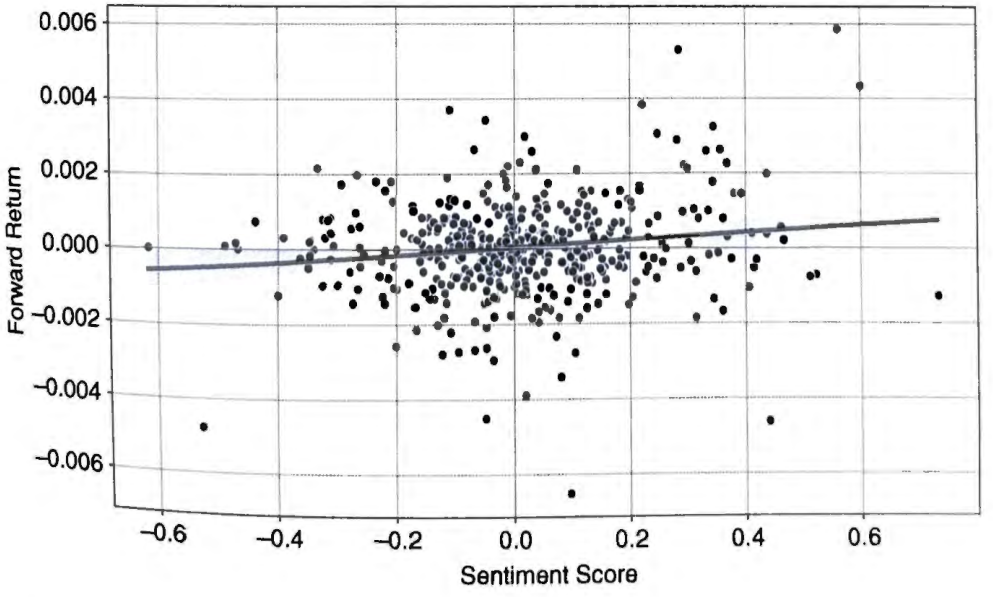

Figure 9.12 Scatter plot of sentiment signal vs forward returns.

Table 9.1 Performance table.

Unnormalized Sharpe ratio	Accuracy	Pearson Correlation of Sentiment Signal vs SPY	*p*-value
11.3844%	53.1532%	14.1458%	0.1999%

Finally, Table 9.1 summarizes the key performance results of the strategy. These results include a simple constant model for transaction costs, assuming 5 basis points (bps) per transaction. Overall, across all videos, the trading signal predicts the sign of the SPY forward returns with an accuracy of 53.15%. The correlation between the sentiment score and the SPY forward returns is 14.14% with a *p*-value of 0.199%.

9.6 Conclusion

The experimental results demonstrate how we can effectively process alternative data, in this case, audio, to generate actionable trading signals. By collecting high-quality audio from the internet, using appropriate audio segmentation, and employing complex models for transcription and sentiment score computation, we were able to generate profitable high-frequency trading signals to trade the next 30 seconds of the SPY. This application highlights the power of deep autoregressive transformer models for modeling audio and text, as covered in Chapter 5. Our experiments demonstrated the critical role of FinBERT, a fine-tuned BERT model for financial data. Unlike general sentiment analysis models, which often miss financial nuances, FinBERT outperformed other tested models not trained on financial data, significantly improving our performance metrics. The next chapter explores methods for customizing LLMs for specialized tasks. This approach highlights the potential for composing very complex deep learning models to address complex financial applications.