A Real-Time Financial Data Analysis System with Kafka and Spark Streaming

MINGCHEN CAI
University of Waterloo
m42cai@uwaterloo.ca

Abstract

This paper introduces a streaming analytics multipule pipeline system designed to capture social media posts in near real time, process them through a robust distributed architecture, and derive basic investment insight reports via natural language processing techniques. The system extracts social media reddit posts from static csv files and real-time reddit api, uses Apache Kafka for reliable message-based data transmission, uses Apache Spark Streaming for continuous and scalable data processing, uses spaCy for organization name recognition, uses FinBERT for domain-specific sentiment classification, and then sends the processed company-sentiment data to another topic in Kafka, and the final consumer with a third-party AI api processes these messages to generate the final report.. Our approach demonstrates how open-source technologies can be combined in a way that is both course-related and substantial enough for advanced academic work in data engineering or analytics. We further show how a weighting function, incorporating factors such as recency, upvotes, and comment count, refines the sentiment aggregation.(Although this project was completed by me alone, in order to make the expression more scientific, the "I" in the following text is replaced by "we" or "us".) We evaluate this system using both live Reddit feeds and offline CSV data to illustrate feasibility, performance, and limitations. Our results suggest that a simple buy, sell, or hold recommendation can be made in near real time for companies mentioned in Reddit posts, making the pipeline valuable for rapid sentiment-driven investment insights.

Keywords

Kafka, Spark Streaming, Reddit, FinBERT, NLP, Real-Time Analytics

1 Introduction

In recent years, social networks such as Reddit have gained substantial traction for discussions that span finance, technology, politics, and beyond. Understanding these real-time public sentiments has become a focus for professionals aiming to enhance decision making or build automated systems capable of digesting large volumes of user-generated data. This project explores how data engineering concepts, particularly streaming architecture, distributed analytics, and targeted natural language processing, can be leveraged to transform raw Reddit posts into actionable investment recommendations. The project is course-related, covering material found in advanced data engineering or data science curricula, including topics such as distributed systems, big data frameworks, and NLP-based text mining. It is also sufficiently substantial because it integrates multiple distinct technologies in a single system, requiring design decisions and practical coding, and ultimately yields results that can be connected to real-world financial applications.

The system this paper present consists of four main Python scripts which makes up to two producer-consumer combinations. In the first combination, the first two scripts handle the production side by either scraping live Reddit posts from the “investing” or other specific subreddit or reading offline data from a CSV file containing historical posts. These scripts connect to Apache Kafka, which acts as the messaging backbone that decouples producers from consumers and ensures reliable data delivery. The consumer side is a Spark Structured Streaming job that reads data in micro-batches from Kafka, uses spaCy to recognize organization names mentioned in the text, performs FinBERT sentiment analysis, and applies a weighting function that emphasizes posts with high scores, large comment counts, or recent timestamps. After processing batch post data from reddit realtime api or static csv files together or

in partitions, the sentimental data will be sent to another kafka topic and processed by the other consumer. That is, the consumer of the first combination is also the producer of the second combination. A final textual report is then generated by the comsumer of the second combination, in which each identified organization is listed with its aggregated sentiment distribution and a simple buy, sell, or hold recommendation. Because the entire solution runs continuously, new posts are streamed into Kafka and made available to the Spark job for real-time updates, demonstrating a fluid integration of advanced data systems.

The project scope is closely tied to modern big data engineering coursework because it addresses how to design robust systems for real-time analytics. Although the project targets the financial domain, the underlying architecture can be generalized to any text-heavy scenario that needs near real-time analysis and sentiment classification. In addition, since some content can be partitioned and run, this system is also an excellent practice of distributed big data systems. Such an undertaking also showcases how to operationalize multiple open-source tools in unison while managing complexities such as topic creation in Kafka, JSON schema handling in Spark[1], and GPU-accelerated model inference for FinBERT if desired. By bridging all these elements, the project is a substantial demonstration of skills that are both academically instructive and practically relevant.

2 Data Source

2.1 Static Data Source

The static data source is a publicly available dataset extracted from the WallStreetBets subreddit, accessible through Kaggle [2]. It contains historical posts with attributes such as title, body, number of comments, upvote score, and timestamp. Including this file in the directory named "archive" allows the system to replay and simulate real-time ingestion from a known corpus, facilitating reproducible experiments and demonstrations without dependency on the Reddit API's rate limits or current subreddit activity. This dataset is valuable for testing the pipeline's behavior, verifying functionality, and benchmarking performance in scenarios that do not require a live connection.In addition, when real-time data is insufficient, it can be used as a data source to replace the real-time read stream so that the system can process enough data.

2.2 Dynamic Data Source

The dynamic data source comes directly from the Reddit API, where new submissions in subreddits such as "WallStreetBets" or "investing" can be fetched through Python scripts using the PRAW library [3]. This real-time connection captures the most recent posts and reflects ongoing conversations, enabling a more authentic portrayal of near real-time sentiment detection. By adjusting parameters such as the subreddit name or search keywords, the system can track different communities or topics. Integrating the Reddit API also demonstrates how the pipeline manages unpredictable data rates, ensuring that new posts are queued and processed continuously without losing messages, thanks to Kafka's fault-tolerant buffering and Spark's micro-batch architecture.

3 Relevant Technologies

3.1 Kafka

Apache Kafka [4] is a distributed streaming platform that uses a publish-subscribe model and writes messages to a fault-tolerant log structure. It was chosen because it provides high throughput, horizontal scalability, and reliable message delivery. By separating data producers from consumers, the system can continue collecting Reddit posts even if the analytics layer experiences downtime or needs to be restarted. This decoupling makes the overall architecture more robust and ensures that no data is lost during temporary failures. Using Kafka also offers the flexibility of working with partitioned data for load balancing and fine-grained control, which becomes valuable in high-traffic scenarios.

3.2 Spacy

SpaCy is a modern natural language processing library used primarily here to detect organization names in Reddit posts [5]. It excels in both accuracy and performance, providing efficient entity recognition through a pipeline that supports batch processing, reducing the overhead of handling each text individually. Because the goal involves identifying company references and the text often lacks explicit cues like "Company X," a robust

named entity recognizer such as spaCy is more effective than simple string matching or regex. This library further supports future customization or domain-specific model fine-tuning, enhancing potential adaptability.

3.3 FinBERT

FinBERT is a domain-specific variant of the BERT model fine-tuned on financial texts, making it well suited for sentiment analysis in stock- or market-related discussions[6]. Its specialized training means it recognizes finance vocabulary more accurately than general-purpose sentiment models. The system benefits from more precise detection of positive, neutral, or negative sentiment in typical investment-related language, which in turn enriches the buy, sell, or hold recommendations. Integrating FinBERT with Spark through the Transformers library allows GPU acceleration and batch inference, supporting potential scaling for larger data volumes.

3.4 Third party AI API

ChatGPT (OpenAI GPT-3.5 in this project because its cheapest rate among AI Api models) is leveraged for generating human-readable financial reports after the weighted sentiment data has been aggregated[7]. Rather than presenting merely numerical results, the system calls the ChatGPT API to transform raw output into a coherent narrative. This approach improves the interpretability of results by providing context, advice, and explanations in natural language. It also offers flexibility in report style or depth by adjusting the prompt, which can be crucial for users who prefer quick summaries or require more detailed analysis and justification.

4 System Architecture

The entire project can be roughly divided into two Kafka data stream pipelines. There are two producers in the first data stream pipeline mainly extract data from static or dynamic data sets. The producers consist of two Python scripts named reddit_scrapper.py and static_reddit_scrap_scrapper.py script interacts with the Reddit API via the Python Reddit API Wrapper (PRAW) . The script reads from a CSV file of historical Reddit data, typically containing older posts from subreddits such as WallStreetBets. Both scripts structure each post's data—title, content, creation time, upvotes, comment count, and more—into JSON objects before publishing them to Kafka. The reason for introducing Kafka is that it provides a robust message queue with high throughput, horizontal scalability, and fault tolerance. It is known for its ability to handle spikes in data volume without dropping messages, thereby decoupling data ingest from data consumption. When the Spark job or any other consumer is offline, Kafka retains the messages until the consumer is ready, minimizing data loss and facilitating robust real-time analytics. The consumer side in the first data stream pipeline is handled by spark_straming_reddit.py, which instantiates a SparkSession configured to read from the Kafka source. In a single conceptual flow, Spark consumes the JSON messages in micro-batches and interprets them according to a predefined schema. This step ensures that data fields such as title or created_utc are mapped to typed Spark DataFrame columns. Once ingested, the text is subjected to a series of transformations, particularly named entity recognition using spaCy. The recognized entities of type ORG form the basis for classifying each post according to which companies are discussed. This classification step ensures that multiple posts mentioning the same company can be aggregated in real time, forming a continuous stream of sentiment data for each mentioned organization. After that, the pipeline applies a FinBERT model to measure the sentiment of each post. FinBERT is specialized for finance-related language, having been fine-tuned on a dataset of financial texts, making it more accurate than a generic sentiment model when analyzing terms such as stock, investment, or yield. After the classification analysis, this python class will also appear as the producer of the second data stream pipeline, specifically sending the processed data to report_generator.py. The sending method is similar to the sending method described above, that is, the processed sentiment information for each company is summarized and sent to another Kafka topic, and then extracted by report_generator.py. Before generating the report, as the only consumer of the second data stream pipeline, report_generator.py needs to further summarize the data extracted from the second topic. The reason is that the producer of the second data stream pipeline may run in partitions, and it is very likely that different posts of the same company will be processed in each partition. After the aggregation, the summary information needs to be sent to the

third-party AI interface. Here we use the chatgpt interface and let AI produce the final real-time investment report.

5 Methodology

The methodology unifies streaming data ingestion, real-time text analytics, and incremental reporting. The reddit_scrapper.py script serves as a live mode, connecting to the “investing” subreddit and fetching posts that match a keyword-based search. Each post is immediately serialized into JSON. The same approach is used in static_reddit_scrapper.py except that it operates on a local CSV dataset. Both scripts rely on the KafkaProducer class from the kafka-python package, which handles low-level details of connecting to the Kafka broker, serializing objects, and sending them to the topic named "investing" by default or named by user. Proper usage of Kafka in such a scenario involves ensuring that the broker is running locally or on a dedicated server, that the "investing" is created or configured to auto-create, and that any required replication or partitioning is set appropriately.

On the consumption side, Spark’s Structured Streaming is responsible for reading from the Kafka source. This is configured using the spark.readStream .format("kafka") method, which yields a DataFrame-like object. The option subscribe=test_topic directs the consumer to read messages from the designated channel, while other options (like startingOffsets) determine whether to begin reading from the earliest or latest available message. The raw binary data in the value column is then cast to a string and parsed into a structured schema that matches the JSON fields produced by the scrapers. Once data is mapped to typed columns, the pipeline processes it in micro-batches whose interval is determined by Spark’s configuration. Instead of a purely streaming environment like Kafka Streams, Spark uses these micro-batches to perform transformations on small chunks of data that accumulate over short time windows.

This consumer also employs a multi-stage methodology that combines event streaming, named entity recognition, domain-specific sentiment analysis, weighting, intermediate aggregation, and send to another topic.The methodology is outlined as follows:

5.1 Kafka Topic Partition

In Apache Kafka, a topic is split into one or more partitions, which serve as a concurrency and scalability mechanism by allowing large streams of messages to be distributed across brokers. Each partition is an ordered, immutable sequence of records that is appended to sequentially. By specifying which partition(s) to read, an application can control how data is distributed and consumed, facilitating parallel processing across multiple consumers or consumer instances. This design also ensures fault tolerance and data replication, because each partition can be replicated and reassigned as needed to maintain availability. Instead of subscribing to an entire Kafka topic by default, we provide an option to read from specific partitions via the `KafkaPartitionQuery` class. This design allows for targeted analysis or load balancing, especially if certain partitions contain high-volume data. Users specify the partition(s) through an argument, and the script constructs JSON strings (e.g., `{"investing":[0]}`) for the `assign` option in Spark, enabling a more fine-grained subscription when needed.

5.2 Pipeline

We adopted the Hugging Face pipeline from transformer library to streamline sentiment analysis tasks by bundling together tokenization, model inference, and post-processing in a single interface. When the pipeline is instantiated with the specified task and model, it automatically converts raw text into tokenized inputs, passes these inputs through the FinBERT model, and processes the resulting outputs to generate the final sentiment labels (“positive,” “neutral,” or “negative”). To specify, when sentiment_pipeline is constructed with the FinBERT model and a particular `batch_size`, it automatically divides the input texts into suitably sized batches, runs each batch through FinBERT on the specified device (in this case, `device=0` for a GPU), and returns a list of sentiment predictions. Internally, the pipeline loads the model weights and tokenizer behind the scenes, ensures that the input length does not exceed the model’s capacity (`max_length=512`), and truncates if necessary. This abstraction allows developers to focus on high-level task definition rather than the underlying complexity of NLP workflows, making it straightforward to incorporate advanced language models into applications without needing to manually handle every step

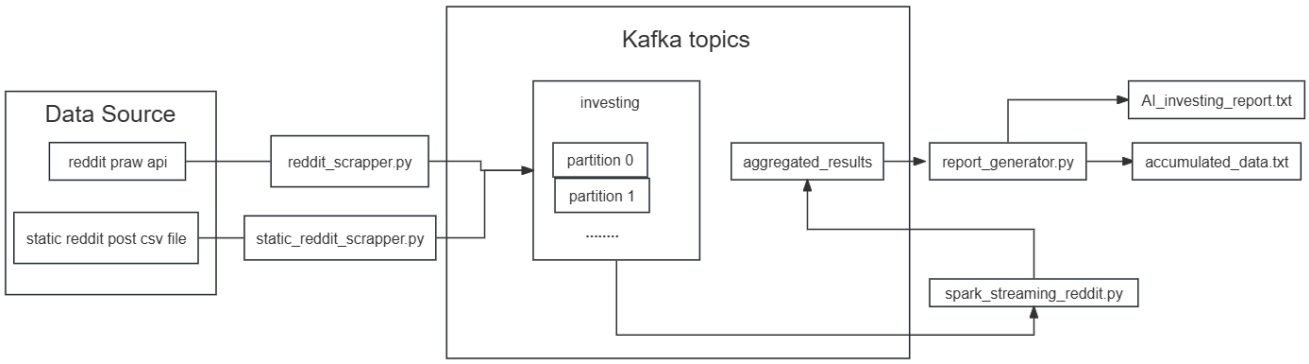

Figure 1: System architecture for the whole data streaming system

of the inference pipeline, including how to tokenize, batch, or feed data into GPU memory.

5.3 Named Entity Recognition and Sentiment Analysis

Once messages from Kafka arrive in Spark micro-batches, each record is parsed into columns such as title, score, comments count, and creation time. We use spaCy to identify organization names (`ENT.label_ == "ORG"`). For each entity discovered, the text is sent to FinBERT for sentiment classification. We specifically exploit the `pipeline` interface to process multiple texts in batches, reducing overhead compared to per-item inference.

5.4 Weighted Sentiment Computation

We combine each post's FinBERT label (negative, neutral, or positive) with a weighting factor that prioritizes fresh and well-received content. Formally, the weight ω is defined as:

$$\omega = 2^{-\frac{\Delta t}{T_{\text{half}}}} \times \left(1 + \frac{\min(s,500)}{500}\right) \times \left(1 + \frac{\min(c,100)}{100}\right),$$

The formula we adopted integrates multiple factors that collectively provide a dynamic weight to each post's sentiment contribution. It incorporates an exponential time decay term $2^{-\frac{\Delta t}{T_{\text{half}}}}$ where Δt represents the time elapsed since the post was created and T_{half} is the designated half-life (for example, 7 days). This term ensures that older posts gradually lose their influence, thereby emphasizing the importance of fresh information in shaping current sentiment. Additionally, the formula includes a term $1 + \frac{\min(s,500)}{500}$, which scales the post's score s (representing the upvote count) to a value between 1.0 and 2.0. This adjustment reflects the level of community approval while capping extreme values to prevent outliers from disproportionately skewing the results. Moreover, a similar term $1 + \frac{\min(c,100)}{100}$ accounts for the number of comments c, capturing the level of engagement and discussion surrounding the post. This term is likewise bounded between 1.0 and 2.0 to maintain balance in the overall weighting. Combined, these three factors effectively balance recency with popularity, ensuring that posts which are both recent and highly engaged exert a greater influence on the overall sentiment score. The resulting weighted score mirrors real-world observations in which current, high-activity discussions have a more significant impact on market sentiment than older, less active posts.

5.5 Global State and Aggregation

For each batch, the script extracts entities, accumulates the weighted sentiment, and updates a global in-memory dataset. Once aggregated, the system sends partial results to a second Kafka topic aggregated_results, where a separate reporting script can merge increments over time. This architecture preserves the decoupling principle by distributing computations and final reporting across multiple processes.

5.6 Report Generation

The final stage of our system involves aggregating partial sentiment updates from a second Kafka topic and

producing user-friendly investment reports. We implement this functionality in the `report_generator.py` script using the following approach:

5.6.1 ***Continuous Consumption of Aggregated Results.*** We subscribe to the "`aggregated_results`" topic with a KafkaConsumer, polling for new messages every 90 seconds. Each message is expected to carry partial sentiment data in the format:

{ "CompanyName": {"Comment numbers": N, "Sentiment counts": {negative: n, neutral: m, positive: p}} ...}.

As new items arrive, these partial results are added into a global dictionary structure (`global_state`), ensuring previously accumulated data is preserved and incrementally updated over time.

5.6.2 ***Threshold-Based Investment Advice.*** Once the aggregated sentiment totals reach at least ten postings per company, a threshold-based function generates a buy-sell-hold recommendation. Define:

$$\text{positive ratio} = \frac{p}{p+m+n}, \quad \text{negative ratio} = \frac{n}{p+m+n}$$

In our system, the threshold value plays a critical role in determining the final investment recommendation —whether to "Buy," "Sell," or "Hold." Essentially, the threshold is used to quantify the difference between the accumulated positive and negative sentiment ratios. When the difference between the positive sentiment ratio and the negative sentiment ratio exceeds the threshold, the system generates a "Buy" recommendation; if the negative sentiment dominates by more than the threshold, it recommends "Sell." Otherwise, the system advises to "Hold."

The chosen threshold value essentially balances sensitivity and robustness. A higher threshold would mean that only significant differences in sentiment—indicating a clear market trend—trigger an active recommendation (buy or sell). Conversely, a lower threshold might lead to frequent changes in recommendations, potentially making the system overly reactive to minor fluctuations or noise in the data. Therefore, setting the threshold involves a trade-off: it must be low enough to detect meaningful shifts in sentiment promptly but high enough to filter out random noise or temporary anomalies.

In our implementation, we use a threshold value of 0.15 based on preliminary experiments and domain insights. This value appears to capture substantial sentiment differences while avoiding excessive signal volatility. Fine-tuning this parameter would typically require empirical evaluation and could vary depending on the specific characteristics of the dataset and the overall market context. where p, m, and n represent the accumulated positive, neutral, and negative weights, respectively. If positive ratio − negative ratio > 0.15, the advice is "Buy"; if the negative ratio outstrips the positive ratio by the same threshold, we recommend "Sell"; otherwise, the default suggestion is "Hold." This simple but transparent rule aligns with our target scenario, offering clear labels for users without external market data.

5.6.3 ***Report Generation and AI Summarization.*** Once all companies have been sorted by their positive sentiment (highest first), the script writes a human-readable summary to a text file, displaying each entity's overall comment count, weighted sentiment distribution, and resulting advice. In parallel, we prepare a prompt concatenating the full accumulated report and send it to a ChatGPT API endpoint. The large language model (e.g. GPT-3.5-turbo) then produces an extended financial analysis. This step underscores the system's flexibility: while a purely numeric report may suffice for programmatic integration or auditing, the AI-generated text offers a more explanatory narrative for business or academic stakeholders. By storing both outputs locally, we retain historical context for each batch of updates and deliver user-friendly information without manually reprocessing the data.

6 Evaluation

For the first producer-consumer combination, which is mainly used to capture data and do some specific data processing, we conducted a series of tests using both live Reddit data (fetched from the "investing" subreddit) and a static CSV file of WallStreetBets posts. In live mode, the `reddit_scrapper.py` script searched for posts containing the keyword 'invest', retrieving approximately 500 submissions. Our environment included an 8-core Windows machine with 16GB of memory, a local Kafka broker, and Spark in local mode. The average delay from when a Reddit post was scraped to its sentiment classification in Spark was about one to three seconds per post, which is acceptable for near-real-time consumption. When pushing five hundred posts into Kafka, we observed no noticeable congestion, and Spark consistently generated incremental results for each microbatch. We also switched to the

static data mode to verify static data streaming. The static_reddit_scrapper.py script read a pre-existing CSV file of 50 to 100 historical WallStreetBets entries, sending them into the same Kafka topic. Even with this alternate source, the pipeline functioned smoothly, indicating it does not rely exclusively on the Reddit API and can handle offline or archived data in a consistent manner.

To further verify the pipeline's robustness, we partition the topic used by the two producers mentioned above into two partitions, and then send data to the topic at the same time. In a time interval similar to the above, the data is sent to the two partitions relatively evenly without loss, that is, the number of data items in the two partitions is close and the sum is equal to the total number of data items sent by the two producers during this period.

In both scenarios,the first combined consumer spark_streaming_reddit.py can receive the data written to the two topics by the two producers in different ways, for example, it can use the subscribe method to obtain the data in all topics regardless of whether the topic has partitions, or it can use the assign method to process the data of each partition in a distributed partition only when the topic has partitions. In any case, the data will not be lost and will be processed in a reasonable time. Of course, partitioned processing will be faster. No matter how the data is obtained, spaCy's named entity recognition recognized typical company names like Apple, Tesla, or Microsoft, although it occasionally misidentified slang or ticker symbols. The FinBERT model, being domain-specific, mostly aligned with financial sentiment but became less certain when posts were vague or off-topic. Meanwhile, the weighting function performed as intended: heavily upvoted, recent posts contributed a larger share to final sentiment tallies, while old or low-engagement posts exerted minimal influence. Companies that attracted significantly negative discussions (e.g., Microsoft or MSTR) often showed a marked shift to a "Sell" recommendation, demonstrating that fresh negative signals dominated the sentiment distribution.

After the above processing, spark_streaming_reddit.py will also serve as the producer of the second producer-consumer combination. The processed sentiment data will be sent to another topic. According to the data acquisition method of the previous topic, the sending method will change subtly. If the assign method is used to obtain data and the previous topic has partition settings, the data here will be partitioned and sent to the second topic separately. If the subscribe method is used to obtain the data of the previous partition, all the data will be sent together to the next topic. After testing, no data will be lost in either way. Of course, the partition method will be faster when sending, but it needs to be merged on the consumer side of the second combination, that is, report_generator.py. After inspection, it was found that there were no duplicate company names in the generated accumulated_data.txt file and that a readable and reasonable investment report would be written into the file AI_investing_report.txt under the premise that there was no failure in the third-party ai api. The above test results can prove that the second producer-consumer combination can run stably and there are no abnormalities in the data processing process.

Overall, the system consistently demonstrated robust performance across both live and static data ingestion scenarios, handling partitioned or non-partitioned topics with no data loss and minimal processing delays. Whether the producers sent data concurrently or the consumer used the subscribe or assign method, Spark was able to quickly retrieve, parse, and apply its weighting logic to all relevant posts in near real time. The spaCy-based named entity recognition and FinBERT-driven sentiment analysis behaved reliably, reflecting intended recommendations—even in more dynamic setups where partitions were used to optimize throughput. In the second producer-consumer phase, the downstream sentiment data passed into another Kafka topic was likewise delivered without duplication or omission, and the final reporting script showed no anomalies in aggregated reports, producing clear and consistent investment summaries in a stable, fault-tolerant manner. However, several limitations remain in this system despite its near real-time data processing capabilities. The threshold-based buy-sell-hold logic relies on estimates and lacks adaptive tuning, so it might not accurately reflect changing market conditions or outlier cases where standard sentiment ratios do not hold. The FinBERT model, while specialized for finance-related text, still depends on preexisting training data and may misinterpret uncommon jargon or rapidly emerging trends. The current approach to entity extraction depends on spaCy's default named entity recognition, leaving room for custom rules or models that might better capture

colloquial references or ticker symbols. Moreover, the pipeline's performance optimizations revolve around partitioning and micro-batching, but the system offers only partial fault tolerance for stateful operations if the Spark driver or other components need to restart. As a result, while workable in moderate data volumes and classroom contexts, the architecture may require more sophisticated strategies to handle large-scale production loads or deeper analytical needs.

7 Presentation of Results

Our pipelines continuously prints and stores incremental sentiment statistics as new batches of Reddit posts are processed. These console outputs track how many posts and which entities were found per batch, along with weighted scores for negative, neutral, and positive classes. Based on these aggregated values, the system proposes buy, hold, or sell advice for each company. Older data never vanishes entirely, though the time-decay factor diminishes its influence in the final tallies, reflecting real-world scenarios where recent market sentiment tends to be more impactful.

We have also verified that the system meets its main requirement of real-time ingestion and sentiment classification. We tested two data sources, observed acceptable latency, and confirmed that the weighting function and FinBERT sentiment model yielded rational recommendations. Under heavy negative sentiment, the pipeline shifted to "Sell," while strongly positive content pushed the recommendation to "Buy." For uncertain or moderate signals, "Hold" was advised. This matches our primary objective: a continuously running pipeline that processes live or static streams, accumulates entity-based sentiment scores, and provides actionable or interpretable advice. In circumstances where the data was insufficient or ambiguous, we explained why "Hold" was advised, thereby clarifying the system's limitations and setting the stage for further enhancements such as adding labeled training data or refining the classification logic.According to the above ideas, the data format generated at the first pipeline consumer end or each partition of the first pipeline consumer end is as follows:

```
== ACCUMULATED REPORT (ALL BATCHES) WITH WEIGHTING ==
Company: NVDA
  Comment numbers: 60
  Weighted Sentiment counts: {'negative': 0.0,
                        'neutral': 30.820685223057886,
                        'positive': 45.63886900942217}
  Investment Advice: Suggestions: Buy

Company: MSTR
  Comment numbers: 40
 Weighted Sentiment counts: {'negative': 40.67913580871482,
                             'neutral': 0.0,
                             'positive': 0.0}
  Investment Advice: Suggestions: Sell

...
```

Note: Due to the large amount of output data, the data shown below is only part of the actual generated file, which is intended to help readers understand the code output format. If you want to view all the information in the generated file, you can go to the output folder in the GitHub repository.(same for the chatgpt part)

Once each micro-batch finishes, the pipeline appends the results to a local text file for historical reference. An optional feature then calls a large language model (ChatGPT) with the updated sentiment data, requesting a concise investment summary. Below is an example of the AI-driven output, shown in smaller text:

```
========== ChatGPT Investment Analysis ==========
Based on the accumulated sentiment analysis data,
here is a concise investment analysis and advice
for the listed companies:

Stocks to Buy:
1. NVDA (Nvidia) : Positive sentiment with a suggestion to buy.
2. LFG : Positive sentiment with a suggestion to buy.
...
Stocks to Sell:
1. MSTR : Negative sentiment with a suggestion to sell.
2. SPY : Negative sentiment with a suggestion to sell.
...
```

This two-pronged reporting strategy—detailed numeric output plus a summarized text report—caters to both technical and non-technical audiences. Users can inspect exact sentiment totals to see precisely how much weight or polarity was assigned, while at the same time receiving straightforward buy-sell-hold guidelines. If desired, the same data can be forwarded to a dashboard or time-series database for advanced visualization.

8 Conclusion & Future Works

This project demonstrates a near real-time data analytics pipeline that exploits Kafka partition-level control, Spark Structured Streaming's micro-batch model, domain-specific NLP via spaCy and FinBERT, and a final AI-based analysis layer. Kafka provides the fault-tolerant messaging backbone, Spark handles distributed streaming analytics, spaCy performs named entity recognition to detect relevant companies, and FinBERT fine-tunes the sentiment analysis for financial contexts. A weighting function, which accounts for recency, upvote count, and comment engagement, further refines the aggregated sentiment. By dividing the process into three stages, the first for obtaining original data from datasets, the second for initial sentiment aggregation in Spark and one for final reporting, the design remains flexible and extensible for multiple downstream consumers. The emphasis on weighting provides a practical approach for highlighting recent and heavily discussed posts in financial sentiment analyses. Although limited by the basic threshold-based buy-sell-hold rule and the partial coverage of named entity recognition for ticker symbols, the pipeline meets most real-world use cases that require distributed, low-latency data processing. Future work could focus on refining entity recognition by customizing spaCy or integrating domain-specific dictionaries that capture ticker symbols and financial abbreviations, while also bringing in real-time market data or fundamental metrics to correlate user sentiment with actual price movements. Additional steps might include moving beyond a simple threshold-based buy-hold-sell logic by incorporating risk measures and machine learning models trained on labeled financial data. Deeper integration with large language models could produce more nuanced thematic insights or user credibility assessments, and multi-lingual support would expand the pipeline's applicability to global finance communities, capturing sentiment in multiple languages and broadening the scope of its real-time investment analysis.

References

[1] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2012.

[2] Gpreda. Reddit WallStreetBets Posts Dataset. *Kaggle*, 2022. Available at: https://www.kaggle.com/datasets/gpreda/reddit-wallstreetsbets-posts/data

[3] Bohnet, J. PRAW: The Python Reddit API Wrapper. *Software Documentation*, 2021. Available at: https://praw.readthedocs.io/en/latest/

[4] Kreps, J., Narkhede, N., & Rao, J. Kafka: a Distributed Messaging System for Log Processing. In *NetDB*, 2011.

[5] Honnibal, M., & Montani, I. spaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks and Incremental Parsing. *To appear*, 2017.

[6] Yang, Y., & Yang, A. FinBERT: A Pretrained Language Model for Financial Communications. *arXiv preprint arXiv:2006.08097*, 2020.

[7] OpenAI. GPT-3.5: Language Models are Few-Shot Learners. *OpenAI Technical Report*, 2023.

[8] Niu, X., Zhu, H., Cao, T., & AlKhalifa, S. Real-Time Discussion Mining Using Distributed Streaming and Domain-Specific NLP. In *Proceedings of the IEEE International Conference on Big Data*, 2020.