News Analysis and Stock Trend Prediction Using PySpark and LLM

COMPSCI 532 - Systems for Data Science

Final Project

Eunbi Yoon Derek Liu Gayathri Akkinapalli Avinash Amballa

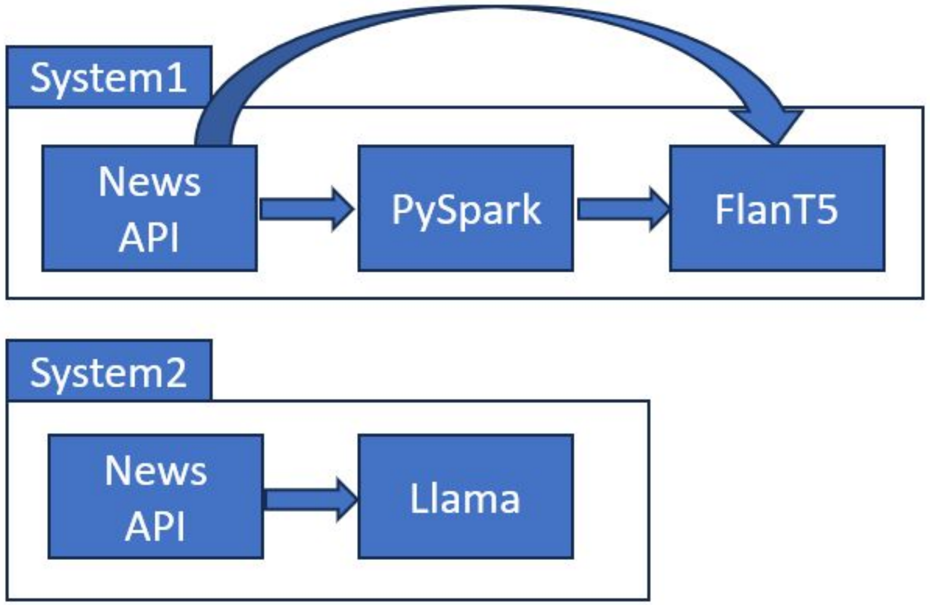
Outline

- **Design Description** — 3-5 slides, including diagrams, describing the overall program design and design decisions considered and made. Identify trade-offs.
  - We're looking for high-level design, such as files, classes, interesting methods, and maybe a couple of code snippets if you're particularly proud of them, plus anything you had difficulty with or didn't manage to accomplish.
  - In this section explain who did what work.

  -> Diagram
- **Tests** — On a separate slide or two, a description of the tests you ran on your program to convince yourself that it works correctly. Also describe any cases for which your program is known not to work correctly.

  -> Long, Short term News input and compared 3 evaluation metrics
- **Experimental Results** — On another separate ~2 slides, include the experimental results and your analysis of them. Include additional research questions and hypotheses about the system that these experimental results don't sufficiently answer.
- **Possible Improvements** — In one slide, describe possible improvements and extensions to your program (and sketch with words and/or diagrams how they might be made).
- **Goals** — 1 slide with your project goals and clear explanation of whether you achieved each of them or not.

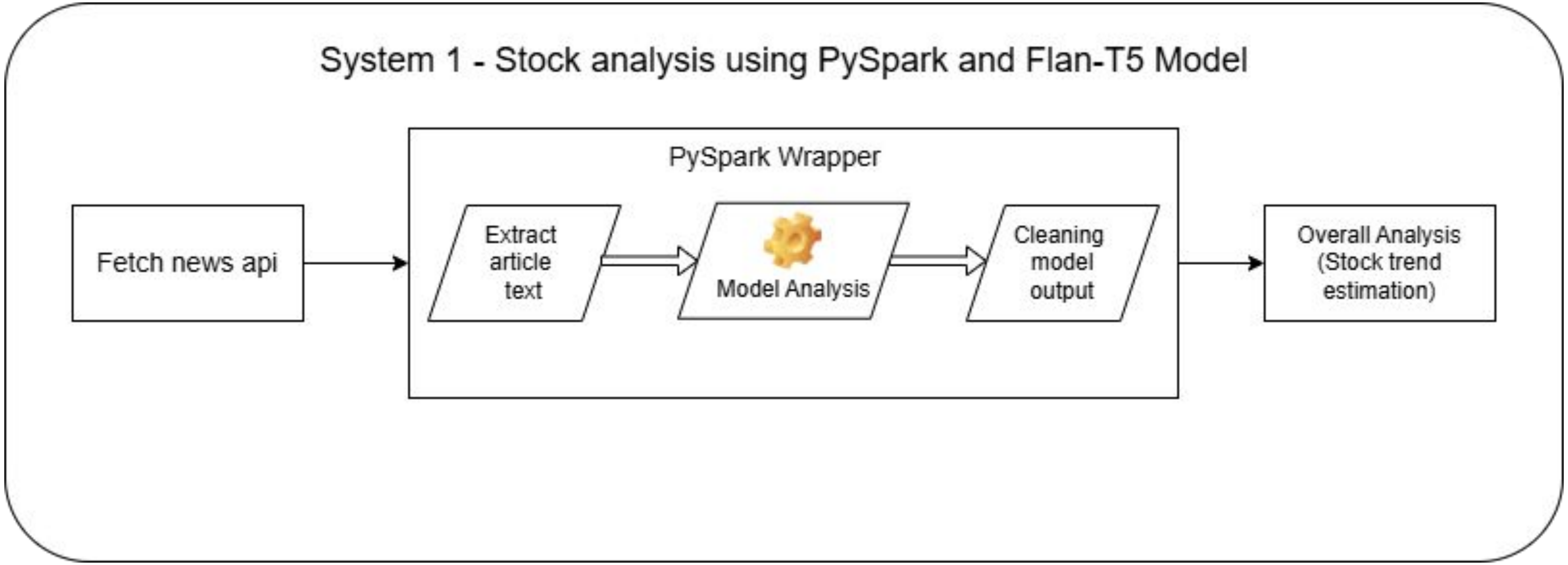
System Diagram



System 1

Stock Analysis Using PySpark and Flan-T5 model

Block Diagram and System configuration


System 1 - Stock analysis using PySpark and Flan-T5 Model
PySpark Wrapper
Fetch news api
Extract article text
Model Analysis
Cleaning model output
Overall Analysis (Stock trend estimation)

GPU : A100 of Unity Cluster
Framework : Pyspark
Model : Flan-T5
Model resource : Hugging Face

Program Design

Fetch News API:

- Used the News API to fetch articles within a given date range using the keyword "apple stock."

PySpark Wrapper:

- The PySpark wrapper combines preprocessing, model analysis, and post-processing functions using PySpark's user-defined functions (UDFs) and applies them to a PySpark DataFrame.
- This approach enables distributed analysis of all article data independently.

Functions used in the PySpark Wrapper:

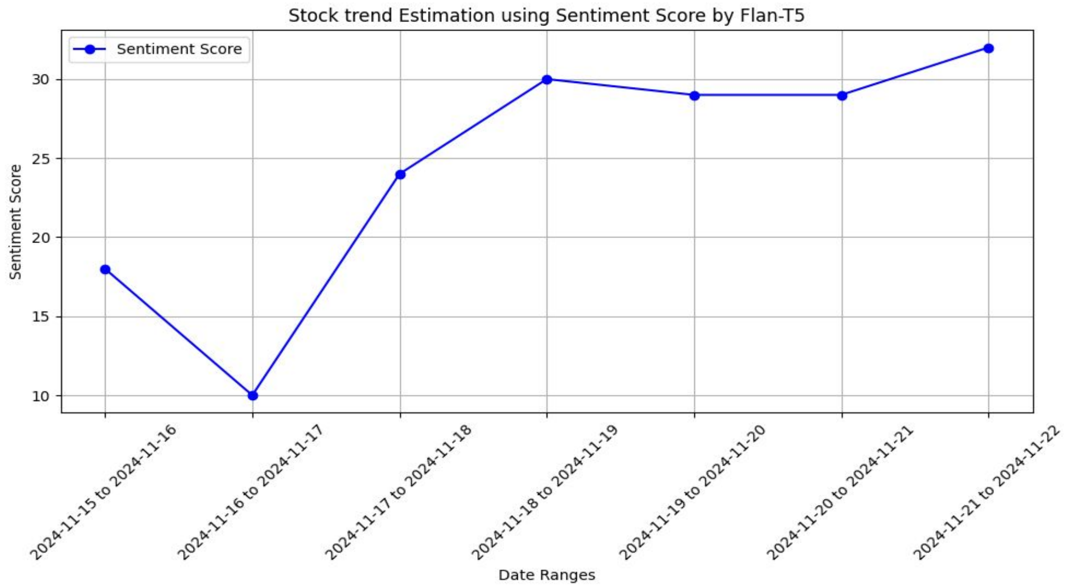
❖ Extract_news_data: Combines the title, description, and content fields of each article into a single text, which serves as the input prompt for the model analysis.
❖ Analyze_news: Performs sentiment analysis on all articles using the task specific prompt (to perform sentiment analysis) combined with article text. The Flan-T5 model from Hugging Face is used for sentiment analysis.
❖ Clean_sentiment: Cleans the model output to ensure only the sentiment labels ("Positive," "Negative," or "Neutral") are retained as the final output.

Final output:

- The final model output is stored as a JSON file containing the sentiments of all articles within the specified date range.

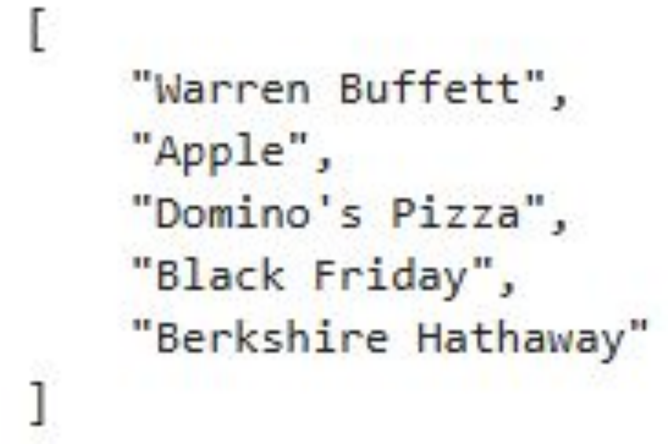
Model Analysis - Stock trend estimation

❖ Stock Trend estimation: We approximated the stock trend based on the sentiments of the articles. For this, we assigned the following values to each sentiment: Positive as +1, Negative as -1, and Neutral as 0. Using these values, we calculated the overall sentiment score for a given date range. This overall sentiment score was then used to plot the estimated stock trend graph. The resulting stock trend estimation plot is shown below.



Model Analysis - Top k most discussed topics

- We used the OpenAI model GPT-4-o-mini to identify the top 5 most discussed topics in the articles extracted within the date range of November 15, 2024, to November 16, 2024, using the keyword “apple stock.”
- The output is heavily influenced by the prompt used in this process. The identified top 5 most discussed topics are listed below:



```
[
    "Warren Buffett",
    "Apple",
    "Domino's Pizza",
    "Black Friday",
    "Berkshire Hathaway"
]
```

Possible Improvements

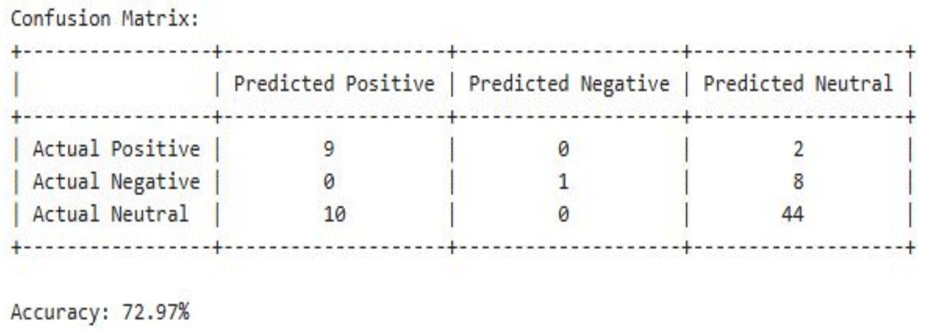
1. Integration of Live Stock Prices: Combine live Apple stock prices with sentiment scores for more accurate and dynamic trend predictions.
2. Fine-Tuned Model: Use a sentiment analysis model specifically fine-tuned for financial contexts to improve precision.
3. Optimize PySpark and GPU Utilization: Refine PySpark configurations and leverage models optimized for A100 GPUs to enhance performance and reduce memory usage. Additionally, implement GPU quantization techniques to improve efficiency and achieve smaller throughput.
4. Advanced Prediction Algorithms: Implement advanced methods like time-series analysis to better correlate sentiment with stock trends.
5. System Scalability: Enhance system scalability by optimizing resource allocation and parallel processing capabilities, allowing for faster processing of larger datasets.

Known Issues:

1. The Flan-T5 model has a limited context window, so it may fail to capture the full context of longer articles.
2. The News API can only fetch articles within a 1-month time range, limiting trend analysis to recent data.
3. Excessive unwanted symbols or tokens in the article text can hinder the model’s ability to identify the article’s context.
4. The system may encounter latency issues when processing large datasets due to suboptimal resource allocation or inefficient parallel processing configurations.

Tests

	Performance Metrics
System 1 Metrics Results	**Accuracy** (Ground Truth model - GPT4-o-mini)
	72.97%



```
Confusion Matrix:
+----------------+------------------+------------------+------------------+
|                | Predicted Positive | Predicted Negative | Predicted Neutral |
+----------------+------------------+------------------+------------------+
| Actual Positive |        9         |        0         |        2         |
| Actual Negative |        0         |        1         |        8         |
| Actual Neutral  |       10         |        0         |       44         |
+----------------+------------------+------------------+------------------+

Accuracy: 72.97%
```

Key observations from Accuracy:

1. The prompt provided to the model significantly influenced its performance.
2. As the Flan-T5 model is primarily fine-tuned for machine translation tasks, achieving this accuracy without additional fine-tuning and considering the model's size is likely the best outcome achievable.

instruction = """You are a sentiment analysis model. Given a {} article, determine the sentiment of the article as one of the following categories: "Positive", "Negative", or "Neutral".

Article: "{}"

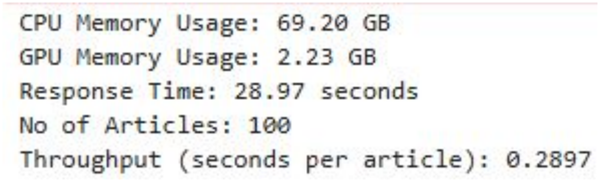
Output: Provide only the sentiment label: "Positive", "Negative", or "Neutral"
"""

Arguments: “apple stock”, news_content

Experimental Results

	System Metrics	
System 1 Metrics Results	Throughput (seconds per article)	Memory Usage
	0.2897 Response Time: 28.97 seconds No of Articles : 100	CPU : 69.20 GB GPU : 2.23 GB

Output:



```
CPU Memory Usage: 69.20 GB
GPU Memory Usage: 2.23 GB
Response Time: 28.97 seconds
No of Articles: 100
Throughput (seconds per article): 0.2897
```

Key observations:

The observed Throughput and response time is comparatively less because of the two main reasons

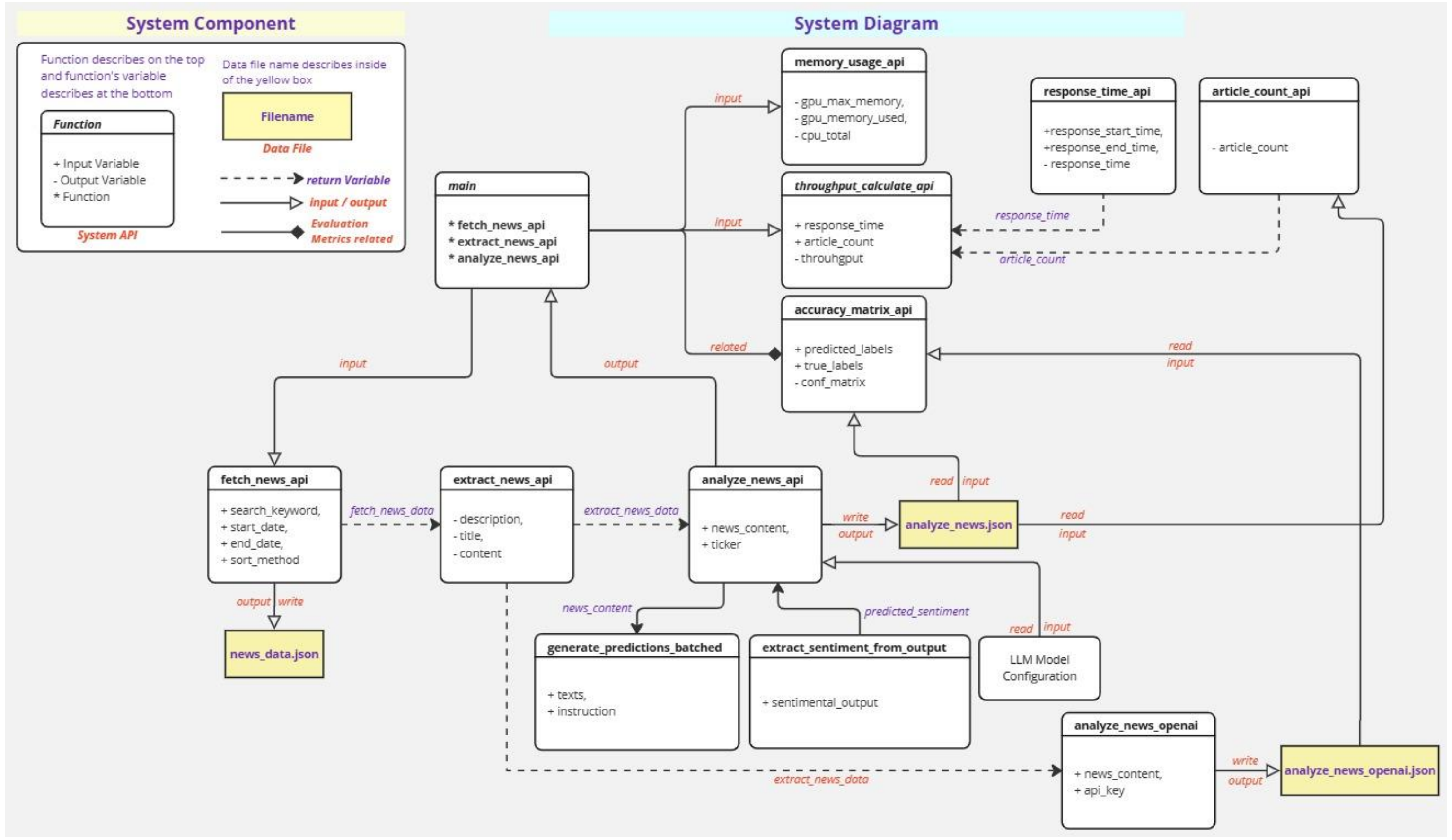
1. The use of PySpark, which reduces throughput and response time due to its distributed nature.
2. The A100 GPU's architecture, optimized for batch performance, leading to higher memory utilization.

System 2

Meta

Llama 3.1

Design Description - Diagram



Design Description - Overall Program Design

Architecture Overview:

- **Data Pipeline**
  - Fetch news articles → Preprocess → Sentiment Analysis → Save Outputs.
- **Sentiment Analysis**:
  - **Model Used:** Llama-3.2-1B with tokenized inputs.
  - **Alternative Model for Accuracy Comparison**: GPT-4o-mini.
- **Performance Metrics**: Memory usage, latency, throughput, and accuracy (Ran on T4 GPU).

Key Functions:

1. **fetch_news_api**:
   - Fetch news articles using NewsAPI.
   - Filters irrelevant content.
2. **extract_news_api**:
   - Prepares and cleans article data for sentiment analysis.
3. **generate_predictions_batched**:
   - Generates predictions using Llama-3.2-1B.
   - wrapper around huggingface transformers.GenerationMixin.generate
4. **analyze_news_openai**:
   - Uses GPT-4o-mini for comparison.

Design Description - Design Decisions and Tradeoffs

```
with torch.no_grad():
    outputs = model.generate(
        **inputs,
        max_new_tokens=1,
        num_beams=1,
        do_sample=False
    )
```

```
for news in news_content:
…
# implicit batch size = 1
output = generate_predictions_batched([news], [instruction])
```

- Batch size
  - Tradeoff: lower batch size is simpler and low latency, but need higher batch size for higher throughput
  - Design Decision: set batch size to 1
- Max new tokens
  - Tradeoff: low max new token has much lower latency, but does not allow chain of thought so lower accuracy
  - Design Decision: set max_new_tokens to 1

Design Description - Design Decisions and Tradeoffs

```
instruction="What is the sentiment of this news? Please choose an
answer from {negative, neutral, positive}."
my_prompt = "sentiment:"

combined_inputs = [f"News:\n{text}\n{instruction}\n{my_prompt}"
for text, instruction in zip(texts, instructions)]

 inputs = tokenizer(combined_inputs, return_tensors="pt",
padding=True, truncation=True, max_length=2048)
```

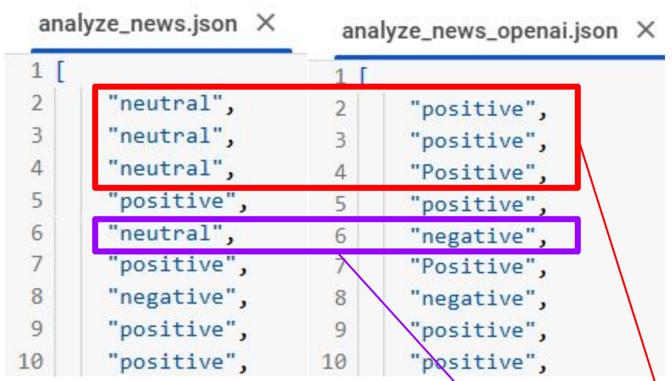
- combined_inputs = [f"News:\n{text}\n{instruction}\n{my_prompt}" for text, instruction in zip(texts, instructions)]
    - Trade off : There is a tradeoff between brevity and clarity. Shorter prompts allow faster response, while longer prompts which includes few-shot examples ensure precision.
    - Design Decision : Choose relatively short prompt, no few-shot examples for faster response
- Max_length = 2048
    - Trade off : Too long max_length preserves full context and short max_length faster processing, efficiency resource use, and reduced noise.
    - Design Decision : 2048 limit is a common maximum token length for transformer models like GPT, balancing sufficient context for tasks with manageable computational and memory requirements.

Tests

Accuracy on financial sentiment analysis on news articles

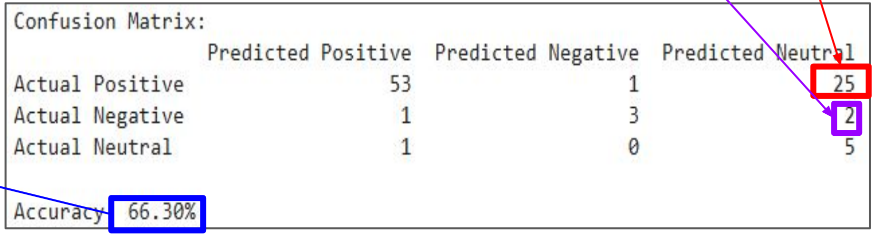
```
def fetch_news_api(search_keyword, start_date, end_date, sort_method):
    api_key = "fcf6368111ce48c3b234b0a479d1dca6"
    # url definition
    url = (
        "https://newsapi.org/v2/everything?"
        f"q={search_keyword}&"
        f"from={start_date}&"
        f"to={end_date}&"
        f"sortBy={sort_method}&"
        f"apiKey={api_key}"
    )
```

Output data file between Llama and GPT



```
[                    [
    "neutral",           "positive",
    "neutral",           "positive",
    "neutral",           "Positive",
    "positive",          "positive",
    "neutral",           "negative",
    "positive",          "Positive",
    "negative",          "negative",
    "positive",          "positive",
    "positive",          "positive",
```

Use confusion metrics to calculate accuracy



```
Confusion Matrix:
                 Predicted Positive  Predicted Negative  Predicted Neutral
Actual Positive                  53                   1                 25
Actual Negative                   1                   3                  2
Actual Neutral                    1                   0                  5

Accuracy  66.30%
```

Test Result

Accuracy between different input datasets is above 50%

□ **no issue with code, function properly!**

Test	search_keyword	(end_date) - (start_date)	Accuracy
#1	apple	1 day	66.30%
#2	apple	1 month	56%
#3	nvidia	1 day	52.58%

Experimental Results - Quantization (1/2)

```
# torchao int8wo
from torchao.quantization import (
quantize_,
int8_weight_only,
)
...
# quantize
quantize_(model =
AutoModelForCausalLM.from_pretrained(m
odel_id, torch_dtype=torch.bfloat16,
device_map="auto")
, int8_weight_only())
```

TorchAO INT8 Weight-Only

- Definition

  Optimizes PyTorch models for general use by quantizing weights-only.

- Slightly lower but not significant in Peak GPU memory usage

  Peak memory usage is dominated by activations, not weights, during forward pass. TorchAO only quantizes the weights, leaving activations in high precision, which leads to still high memory consumption.

- Worse Throughput

  Weights are stored in INT8 but are dequantized to bfloat16 during computation, introducing an additional dequantization overhead which affects worse throughput

Quantization technique	Code	Peak GPU memory usage	Throughput (seconds/articles)
Base Model	No Quantization	2.56 GB	0.22
TorchAO INT8 Weight-Only	int8_weight_only()	2.48 GB	0.33
LLM BNB 8-bit Quantization	bnb_config(load_in_8bit=True)	1.29 GB	0.23

Experimental Results - Quantization (2/2)

```
# LLM.int8()
from transformers import BitsAndBytesConfig
...
bnb_config = BitsAndBytesConfig(
load_in_8bit=True,
)

model =
AutoModelForCausalLM.from_pretrained(
"/content/Llama-3.2-1B-Instruct",
torch_dtype="auto",
device_map="auto",
quantization_config=bnb_config,
)
```

LLM BNB 8-bit quantization

- Definition

  Developed by the BitsAndBytes (BNB) library, designed to efficiently optimize Large Language Models (LLMs) for inference
- Lower Peak GPU memory usage

  No weight dequantization and activations are also quantized to INT8, so achieve half of base peak GPU memory usage
- Similar Throughput with base model(no quantization)

  Weights are pre-quantized to INT8 and modern GPUs, such as NVIDIA GPUs with Tensor Cores, provide dedicated hardware acceleration for INT8 operations. Also activations are quantized dynamically during computation which is lightweight and optimized to minimize any computational overhead, meaning it adds very little extra time compared to the base model.

Conclusion : Choose LLM BNB 8-bit Quantization Technique !

Quantization technique	Code	Peak GPU memory usage	Throughput (seconds/articles)
Base Model	No Quantization	2.56 GB	0.22
TorchAO INT8 Weight-Only	int8_weight_only()	2.48 GB	0.33
LLM BNB 8-bit Quantization	bnb_config(load_in_8bit=True)	1.29 GB	0.23

Possible Improvements

Enhancements to Accuracy:

- Fine-tune Llama-3.2-1B to reduce neutral label and classify to positive or negative.
- Ground truth from gpt4o-mini was not deterministic, got different ground truth each time we called openai api
- Find more reliable truth label which is not changeable in every execution

Additional Features:

- Real-time streaming for news article analysis.
- Broader sentiment categories (e.g., "very positive," "very negative").

Performance Optimizations:

- Explore more aggressive quantization techniques.
- Explore pruning and Deep Learning Compilers

Goals

Evaluation Metrics of System 1 & System 2

Search apple stock in 1 day	GPU memory usage	Throughput (seconds/articles)	Accuracy
System1	2.23 GB	0.2897	72.97%
System2	1.29 GB	0.23	63.03%

Conclusion:
System 1 is good in Accuracy
System 2 is good in GPU memory usage and Throughput.

From the Final Project Proposal Draft #2, all final project goals have been achieved!

- **4.1.1. System 1 - Analysis using Flan-T5 LLM with Pyspark** ✅
- **4.1.2. System 2 - Analysis Directly with Llama LLM without Pyspark** ✅
- **4.2. Overall results:** ✅
- We will estimate the stock market trend by aggregating sentiment scores over a given time frame. This net sentiment score will serve as a proxy for estimating stock market movement, and we will visualize this trend to provide a rough estimate of market behavior based on news sentiment. After we retain certain accuracy, we will focus on increasing performance in the aspects of the system as below visualizing plots and metrics.
- **4.3. Evaluation Metrics** ✅
    - Peak Memory Usage
    - Latency
    - Accuracy
- **4.4. Conclusion** ✅
- Visualize evaluation metrics with graph and table and identify which system offers the best overall performance in terms of system efficiency. This analysis will guide us in selecting the system that can best balance memory usage, latency, throughput, and accuracy. Ultimately, our goal is to find a more optimized system that can meet the demands of real-time stock trend prediction by efficiently processing high volumes of data while maintaining a high level of sentiment classification accuracy.

Contributions

System1

Gayathri - Pyspark framework integration, System metrics, Stock trend estimation

Avinash - Python functions and api integration, Accuracy, Top k most discussed topics

System 2

Eunbi & Derek - Quantization, Fetch Datasets, Accuracy, System metrics

PySpark
pyspark
THANK YOU!