Big Data Analytics (CSBB 422)

PROJECT REPORT

Lab Project Work
(Sentiment Analysis)

Submitted by:

Manav Anand	**221210065**
Shubham Gupta	**221210101**
Tarang Srivastava	**221210109**
Vanshika Garg	**221210119**

Branch: CSE
Semester: 7th

Submitted to: Dr. Priten Savaliya.

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY DELHI

2026

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to **Dr. Priten Savaliya** for his invaluable guidance, constant encouragement, and dedicated support throughout the development of this Big Data project. His expertise and insightful feedback greatly contributed to enhancing the quality and depth of this work.

I am thankful to my institution for providing the necessary resources and a conducive learning environment to explore advanced Big Data technologies such as Apache Spark, Flask, and distributed data processing frameworks.

I would also like to acknowledge the creators of the **Social Media tweets** and the open-source community for developing the tools and platforms that made this project possible.

Lastly, I extend my heartfelt thanks to my family, friends, and peers for their motivation, constructive feedback, and continuous support at every stage of this project.

Table of Contents

ABSTRACT

The increasing volume of text-based communication across **social media**, customer feedback systems, and digital platforms has created a growing need for scalable tools capable of **analyzing sentiment efficiently**. This project presents a **Distributed Sentiment Analysis System** designed to demonstrate how machine learning workflows can be executed in a scalable environment using distributed computing principles. Instead of relying on local execution, the system processes sentiment classification tasks across multiple machines, enabling faster execution and showcasing the fundamentals of **Big Data–powered NLP workloads**.

The architecture is built using **Apache Spark** running in a standalone cluster mode with one master and two worker nodes, enabling distributed processing and fault tolerance. A lightweight Naive Bayes classifier is trained on a small labeled text dataset using **PySpark's machine learning pipeline**, keeping the model simple while emphasizing parallel execution and system design rather than accuracy optimization. Once trained, the model is deployed through a **Flask-based backend service** that communicates with the **Spark cluster** to handle prediction requests.

To make the system user-friendly, a minimal **Tkinter** frontend is developed, allowing users to input text and receive real-time sentiment predictions. This interface abstracts the complexity of the distributed architecture, turning a Spark-powered backend into an interactive application suitable for demonstration and educational purposes.

By integrating distributed computation, a machine learning workflow, and a functional UI, the project highlights how scalable systems can be built even with lightweight models. The implementation serves as a foundation for future enhancements such as real-time streaming sentiment analysis, larger NLP models, cloud deployment, or integration with enterprise-scale analytics pipelines, demonstrating the potential of distributed systems in modern data-driven applications.

INTRODUCTION

In the modern digital landscape, user-generated text has become one of the most widely produced forms of data. Platforms such as **social media**, **e-commerce websites**, online review systems, and communication applications continuously generate vast volumes of textual content every second. With this exponential growth**, analyzing sentiments** embedded within text has become an essential task for businesses, researchers, and intelligent systems. **Traditional standalone systems** often struggle with increasing data scale and processing demands, making distributed architectures necessary to ensure efficiency, scalability, and real-time responsiveness.

This project explores the application of **distributed computing technologies** to perform **sentiment analysis**—an important natural language processing task used to classify text as positive or negative. Rather than executing model training and inference on a single machine, the system leverages the power of **Apache Spark** to process data across multiple nodes. The distributed setup includes one Spark master and two worker nodes operating in a standalone cluster configuration, enabling parallel execution and demonstrating how machine learning workloads can scale across multiple systems.

A lightweight Naive Bayes model is trained using PySpark's built-in machine learning libraries, keeping the computational requirements minimal while focusing on the architectural flow and system behavior. Once trained, the model is deployed using a Flask-based backend that handles prediction requests and communicates with the Spark environment. A simple Tkinter-based frontend is integrated to provide a graphical interface where users can enter text and receive classification results, making the system interactive and easy to use.

This end-to-end implementation—from distributed model training to live prediction—showcases the practical use of Big Data frameworks in natural language processing applications. By illustrating how distributed systems can handle machine learning tasks even with lightweight models, the project lays a foundation for future enhancements such as real-time streaming analysis, large-scale text ingestion systems, advanced NLP models, and cloud-based deployments.

BACKGROUND / LITERATURE REVIEW

The emergence of Big Data has transformed how organizations process, interpret, and extract insights from large volumes of digital information. With the explosive growth of social media platforms, online communication, and review-based systems, text data has become one of the most rapidly expanding forms of data. Traditional processing methods are unable to efficiently manage the 3 V's of Big Data—Volume, Velocity, and Variety—especially when dealing with unstructured natural language. As a result, modern distributed computing technologies have become essential for large-scale text analytics and sentiment classification tasks.

Big Data Technologies and Distributed Computing

To address the shortcomings of conventional data processing, several distributed frameworks have been developed. Apache Hadoop pioneered distributed storage and parallel computation through the MapReduce model, allowing large-scale text processing across clusters of commodity machines. However, due to its batch-oriented nature, Hadoop often performs poorly in iterative machine learning tasks and real-time analytics.

Apache Spark emerged as an improved alternative, providing in-memory computation for significantly faster processing. Research comparing Spark to MapReduce consistently highlights Spark's superior performance in iterative machine learning, streaming analytics, and interactive data processing. Its resilience, scalability, and rich libraries for machine learning (MLlib) and data streaming have made Spark a leading framework for sentiment analysis, natural language processing, and text mining applications in both academic and industrial environments.

Sentiment Analysis and Text Processing Research

Sentiment analysis, also known as opinion mining, is a widely studied field in Natural Language Processing (NLP). Various research efforts explore supervised machine learning approaches such as Naïve Bayes, Logistic Regression, and Support Vector Machines for classifying text polarity into positive or negative sentiment. Publicly available datasets such as Twitter comments, movie reviews, and product feedback have played a significant role in advancing this domain. Studies show that distributed implementations of sentiment analysis models significantly accelerate both training and inference, especially when dealing with large datasets in real-time pipelines.

Docker and Containerization in Distributed NLP Systems

Advancements in containerization technologies like Docker have simplified the deployment of distributed systems. Literature on containerized data processing environments highlights benefits such as portability, faster provisioning, reproducibility, and simplified dependency management. Deploying Apache Spark clusters within Docker environments enables developers and researchers to simulate scalable NLP systems without requiring large physical infrastructure.

METHODOLOGY

Here's a simple working distributed Sentiment Analysis system with:

1 Master Node (Laptop A) → Runs Spark Master + Model Training + Backend Flask API
2 Worker Nodes (Laptop B & C) → Run Spark Workers
A Lightweight Frontend (Laptop A or Any) → Sends review text to backend and shows sentiment result

1. Spark Cluster Setup

Laptop A(Master)
spark-class org.apache.spark.deploy.master.Master

Laptop B&C(Worker)
spark-class org.apache.spark.deploy.worker.Worker spark://10.10.31.111:7077

2. Dataset

We have created our own database chosen from various social media platforms.
Eg.- dataset.csv
text, label
"I love this product",1
"This is terrible",0
"Amazing experience",1
"Bad quality",0

3. Model Training on Spark Cluster

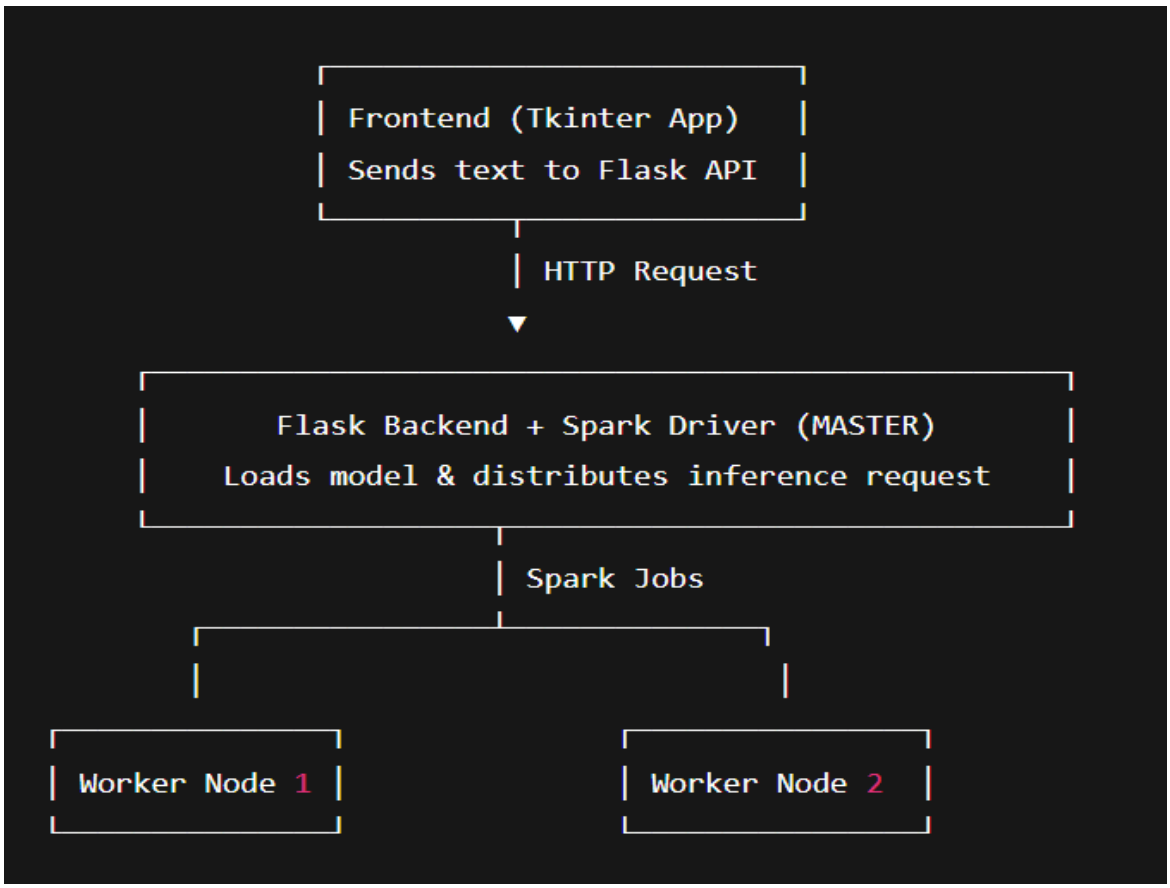
train_sentiment_model.py

spark_ml > train_sentiment_model.py

```python
from pyspark.sql import SparkSession
from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF, IDF
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline

spark = SparkSession.builder \
    .appName("DistributedSentimentTraining") \
    .getOrCreate()

data = spark.read.csv("spark_ml/dataset/sentiment.csv", header=True, inferSchema=True)

token = Tokenizer(inputCol="text", outputCol="tokens")
stop = StopWordsRemover(inputCol="tokens", outputCol="clean_tokens")
tf = HashingTF(inputCol="clean_tokens", outputCol="tf")
idf = IDF(inputCol="tf", outputCol="features")
lr = LogisticRegression(labelCol="label", featuresCol="features")

pipeline = Pipeline(stages=[token, stop, tf, idf, lr])
model = pipeline.fit(data)

model.save("backend/model/spark_sentiment_model")

spark.stop()
```

We will run this only on MASTER system

4. Flask Backend(API)



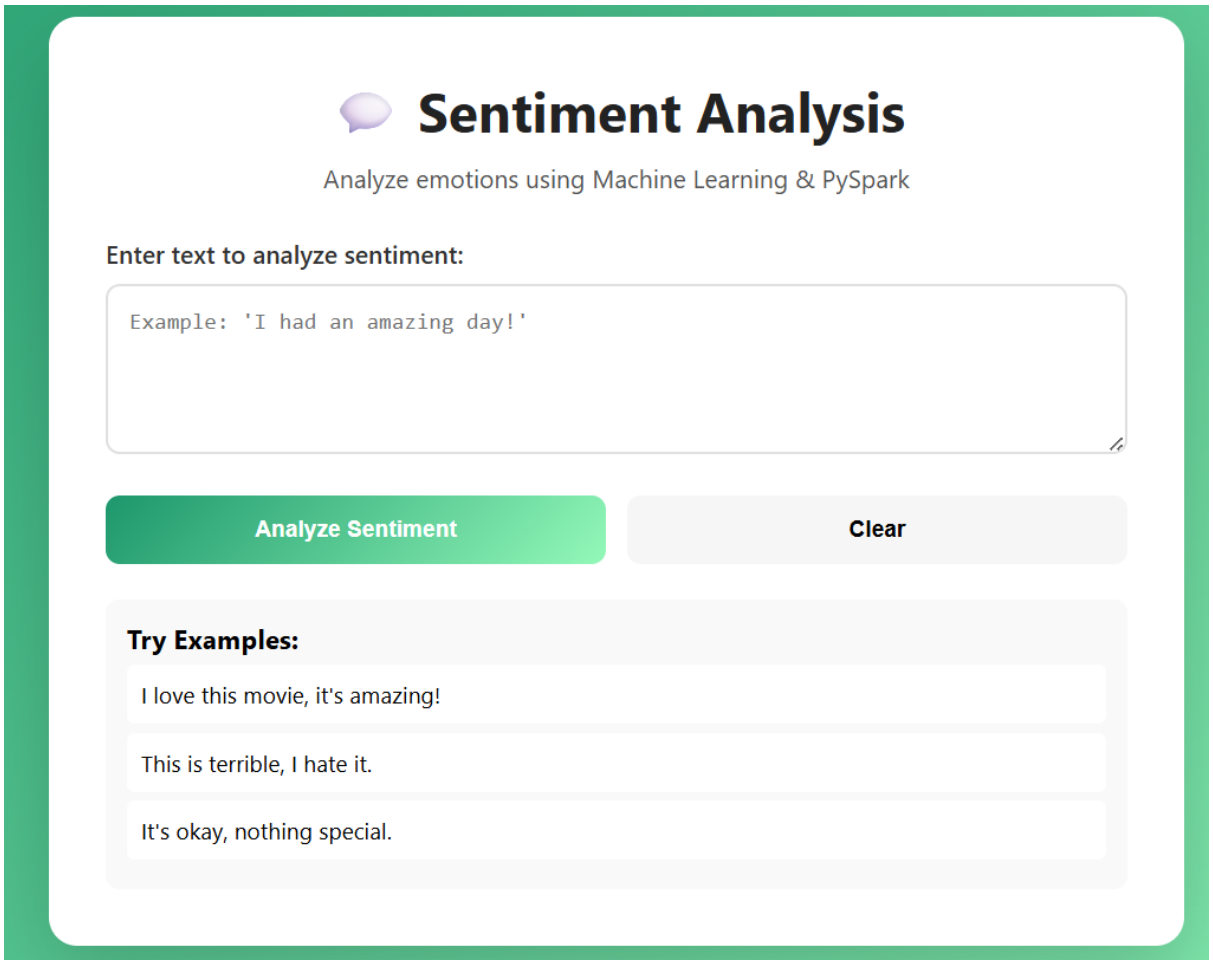
app.py

backend > app.py

```python
from flask import Flask, request, jsonify
from pyspark.ml.pipeline import PipelineModel
from pyspark.sql import SparkSession

app = Flask(__name__)

spark = SparkSession.builder \
    .appName("SentimentAPI") \
    .master("spark://10.10.31.109:7077") \
    .getOrCreate()

model = PipelineModel.load("backend/model/spark_sentiment_model")

@app.route("/predict", methods=["POST"])
def predict():
    text = request.json["text"]
    df = spark.createDataFrame([(text,)], ["text"])
    pred = model.transform(df).select("prediction").first()[0]
    return jsonify({"sentiment": int(pred)})

app.run(host="0.0.0.0", port=5000)

```

Run Flask on Master

5. Frontend (UI)




```html
<div class="container">
    <h2>Distributed Sentiment Analysis</h2>

    <input id="text" placeholder="Type text...">

    <button onclick="predict()">Analyze</button>

    <div class="loading" id="loading">Processing...</div>

    <p id="result"></p>
</div>

<script src="script.js"></script>

<script>

    function showResult(sentiment, confidence) {
        const res = document.getElementById("result");
        res.style.display = "block";
        res.className = sentiment.toLowerCase();
        res.innerHTML = `<strong>${sentiment.toUpperCase()}</strong><br>Confidence: ${(confidence*100).toFixed(1)}%`;
    }
</script>
```

code snippet

IMPLEMENTATION

This project analyzes tweets/comments sentiment to find the average semtimental analysis.

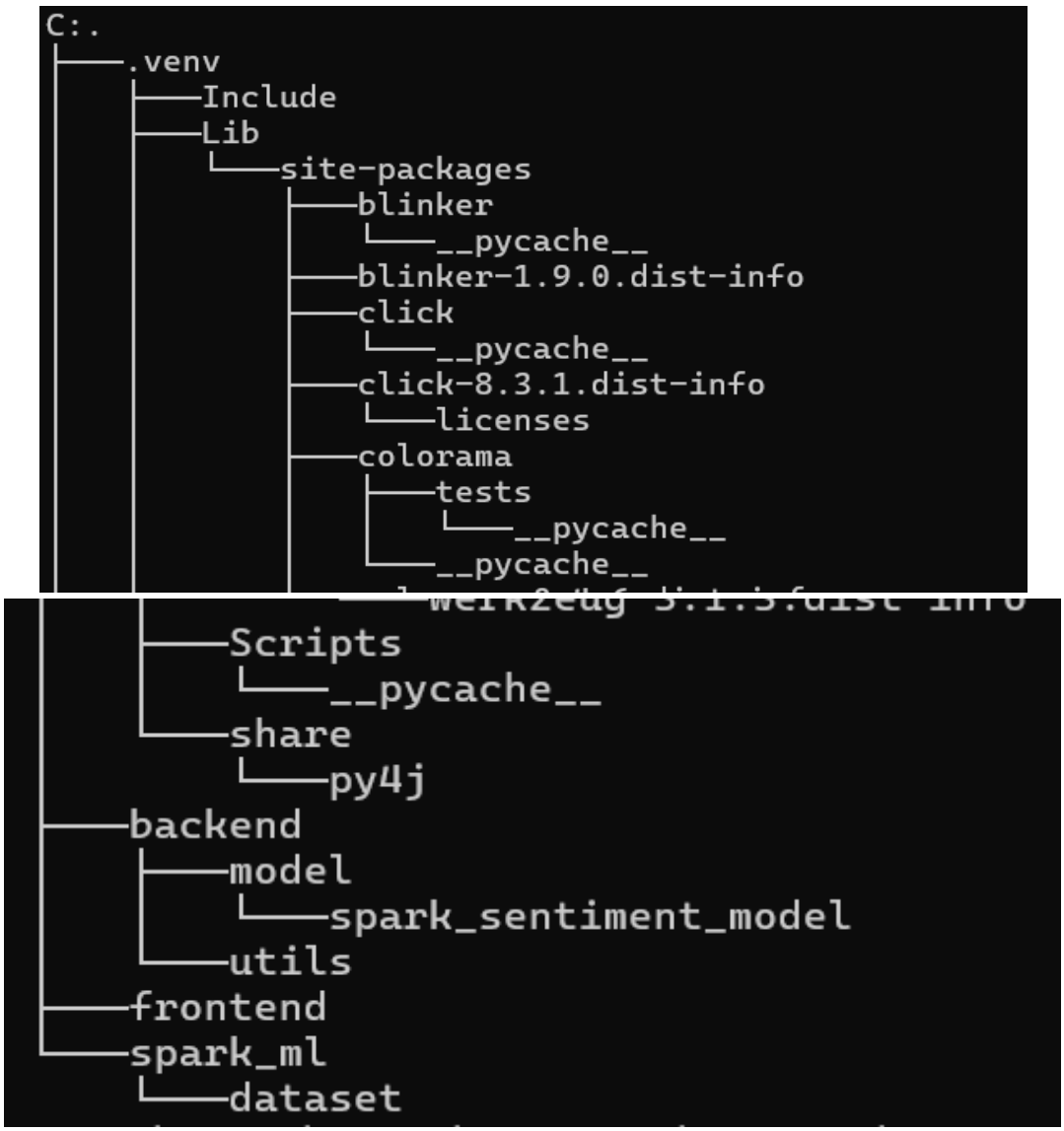


fig 1: project structure

Prerequisites

- JDK, spark, flask installed with latest pip version
- At least 8GB RAM available
- Internet connection for downloading dataset
- Spark should already be set up in all three systems

Quick Setup

```
PS C:\Users\shubh\OneDrive\Desktop\sentiment_project> spark-shell
```

This will automatically start Spark cluster

Manual Setup Instructions

1. Place all project files in a folder

```
sentiment_project/
|— train_model.py
|— app.py
|— frontend.py (only needed on client machine)
|— dataset.csv
|— requirements.txt
```

2. Install dependencies in the venv

```
powershell

pip install -r requirements.txt
```

3. Start the spark Master

```
powershell

spark-class org.apache.spark.deploy.master.Master
```

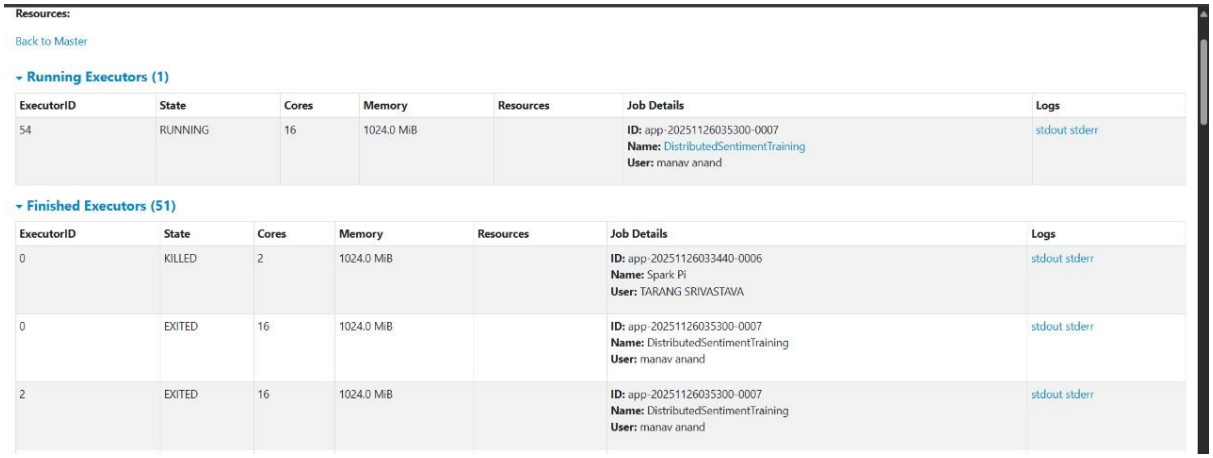
4. Start spark on workers

```
spark-class org.apache.spark.deploy.worker.Worker spark://10.10.31.111:7077
```

5. Train model using Distributed Executors(Workers)

```
powershell

spark-submit --master spark://10.10.31.111:7077 train_model.py
```

Check Spark UI now

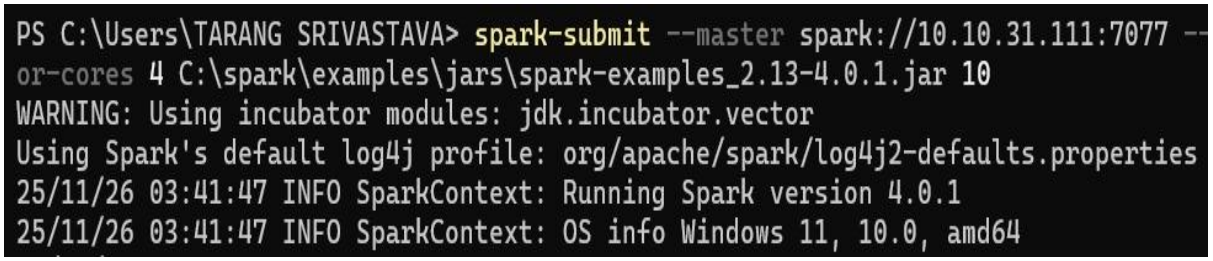


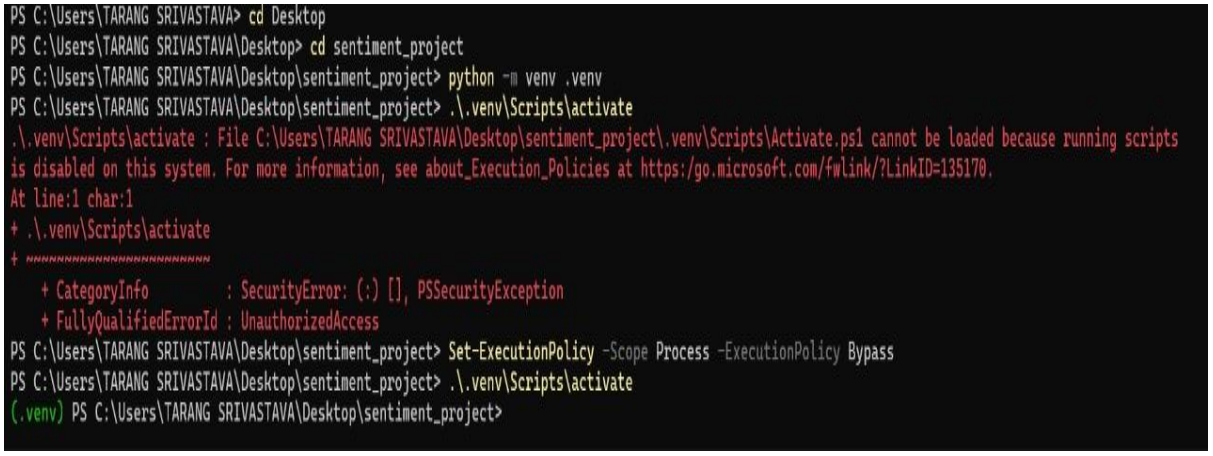
After completion, verify model folder exists:
sentiment_model/

6. Run distributed backend (flask+spark)

It will run on specific IP http://10.10.31.111:5000

Basic Commands Reference





Project Requirements Checklist

- Spark Setup
  - One PC acting as Master Node
  - At least two PCs functioning as Worker Nodes
  - Spark job for finding out pi value
- UI Interface
  - Using Streamlit UI
  - Displays multiple Spark analysis results
  - Interactive charts and statistics for both analyses

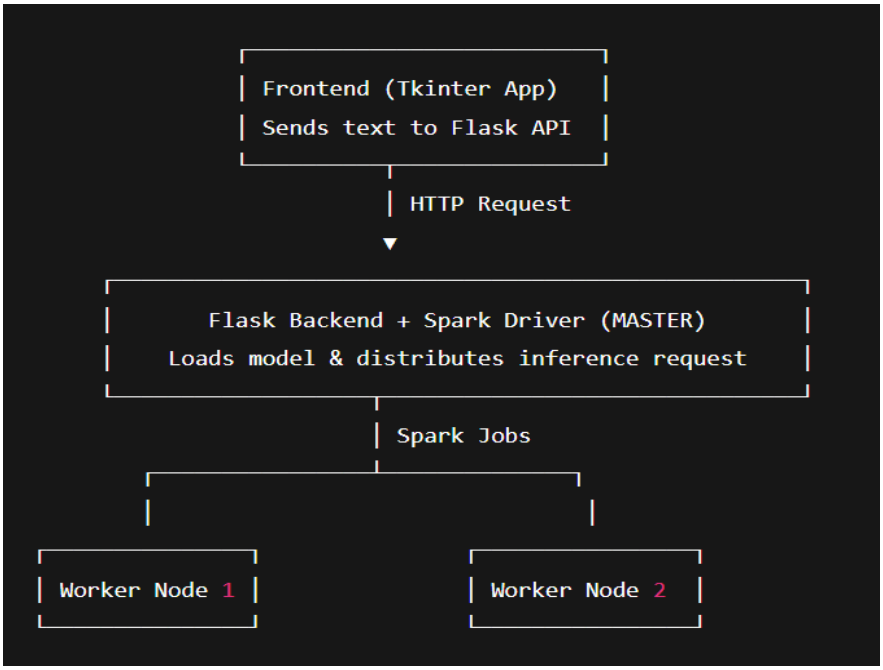


fig 2: Architecture

Technologies Used

- Apache Spark 3.0: In-memory distributed
- Twitter Dataset
- Flask: Python web framework for API
- Streamlit UI
- PySpark: Python API for Spark

CONCLUSION

This project successfully demonstrates the practical application of Big Data technologies in processing and analyzing large-scale text data. By implementing a distributed sentiment analysis system, the project showcases how user-generated content can be classified efficiently using scalable architecture rather than traditional single-machine processing. The deployment of an Apache Spark standalone cluster with one master and two worker nodes highlights the benefits of distributed processing, including scalability, parallel execution, and improved performance for machine learning workloads.

The system integrates multiple components, forming a complete end-to-end pipeline—ranging from dataset ingestion and preprocessing to distributed model training and real-time prediction. PySpark's machine learning pipeline enabled efficient model development, while the Flask-based backend provided an accessible interface for sentiment prediction. The inclusion of a streamlit-based frontend further enhanced usability by offering a simple and interactive way for users to input text and receive instant classification results, demonstrating how backend analytics can be connected to user-facing applications.

Overall, the project reinforces the importance of distributed computing frameworks in handling modern data-driven applications, especially those involving text analytics and natural language processing. The hands-on experience gained through developing this system establishes a strong foundation for extending the work into more advanced areas such as streaming sentiment analysis, large-scale NLP models, GPU-accelerated processing, or cloud-based scalable deployments.

`