



Comment Analysis Chrome Extension for YouTube

1*Gandharv Rudra, 2Aseem Saini,3Keshav ,4Sanjay Kumar Singh,

1 Final year BTech Student,2 Final Year BTech Student,3 Final Year BTech Student,4Professor, USAR

1Department of Artificial Intelligence and Machine Learning,

1 USAR, GGSIP University, Delhi, India

Abstract: This project focuses on the application of MLOps principles in developing a sentiment analysis system for YouTube comments, delivered through a Chrome Extension interface. The objective is to automate the process of collecting, analysing, and visualizing user sentiments from YouTube videos using a machine learning pipeline that integrates seamlessly with a real-world deployment setup. At the core, a pre-trained machine learning model is employed to classify comments into positive, negative, or neutral categories. The model, along with the TF-IDF vectorizer and preprocessing logic, is versioned and tracked using ML Flow, ensuring reproducibility and traceability throughout the development lifecycle. The dataset and model artifacts are managed using DVC (Data Version Control), enabling efficient storage and collaboration. For storage and deployment readiness, the model and metadata are synced to an AWS S3 bucket, allowing easy integration during inference, the backend is built using Flask, exposing RESTful API endpoints to receive comments, perform predictions, and return analytical insights including a sentiment pie chart, word cloud, and trend graph. These endpoints are consumed by the frontend Chrome Extension, which acts as a lightweight client to display insights in real time, While deployment is pending, the planned strategy involves containerizing the backend using Docker and deploying it on an EC2 instance, enabling scalable and accessible access to the model via API, this project is a practical demonstration of applying the MLOps lifecycle in a real-world application, covering model training, versioning, pipeline management, artifact tracking, and deployment preparation. It offers a blueprint for integrating machine learning models into production environments with reliability and scalability.

Key Words – Data versioning, Natural Language Processing, Sentiment Analysis, MLOps, Browser Extension, Flask.

1. INTRODUCTION

YouTube is one of the largest content-sharing platforms globally, engaging millions of users who actively participate through likes, shares, and especially comments on videos. These comments often contain a wealth of unstructured information that reflects the audience's sentiments, preferences, and reactions to the content. For content creators, businesses, and researchers, analyzing this data can offer significant insights that inform decision-making, strategy development, and content improvement. However, the volume of comments on popular videos can reach thousands or even millions, making manual analysis not only impractical but also inefficient and error-prone.

To address this challenge, the project proposes a YouTube Comment Analysis system in the form of a Chrome extension. The primary goal of this system is to automatically classify comments into three sentiment categories: positive, negative, or neutral. In addition to sentiment prediction, the extension will provide users with analytical visualizations such as sentiment distribution through pie charts, average comment length, unique commenter statistics, and trend graphs showing sentiment over time. A word cloud will also be generated to highlight the most frequently used terms in the comments.

The frontend of the extension is developed using JavaScript and HTML, enabling seamless integration with the YouTube interface and offering a user-friendly experience. The backend is powered by a Flask application that handles machine learning inference and serves dynamic visual outputs. Communication between the frontend and backend is facilitated through API calls, ensuring real-time interaction and response delivery.

What sets this project apart is the inclusion of a full MLOps (Machine Learning Operations) pipeline. The system is designed not just as a standalone tool but as a deployable, maintainable, and scalable machine learning application. Tools like DVC (Data Version Control) are used to track dataset and model versioning, ensuring reproducibility. MLflow is employed for tracking experiments, managing model lifecycle, and monitoring performance. The backend model and infrastructure will be containerized using Docker for consistent deployment across environments. The deployment plan involves pushing the Docker image to AWS ECR (Elastic Container Registry) and running the service on an EC2 instance, thereby simulating a production-ready deployment pipeline.

By combining machine learning with MLOps principles, the project emphasizes not just the predictive capabilities of the model but also the reliability, scalability, and maintainability of the entire application lifecycle. This approach ensures that future enhancements—such as retraining the model with new data or scaling the service to accommodate higher loads—can be implemented smoothly and systematically.

2. RESEARCH GAP

Despite significant advancements in artificial intelligence and sentiment analysis in the realm of social media platforms such as Reddit and YouTube. However, the availability of easy-to-use tools for non-technical users that analyse YouTube comments directly in-browser remains limited. Our proposed extension fills this gap by providing an accessible, browser-native tool for comment analysis tailored to YouTube.

- 1) **Time-Consuming and Inefficient:** Manually reading and categorizing thousands of comments on a video is impractical and slows down decision-making for content creators and marketers.
- 2) **Subjectivity and Inconsistency:** Human-based sentiment analysis is prone to bias and inconsistency, leading to inaccurate insights about audience perception.
- 3) **Lack of Visualization:** Raw comments do not provide structured insights, making it difficult to analyse sentiment trends and engagement levels over time.
- 4) **Scalability Issues:** As video content grows, so does the volume of comments, making it impossible to process them efficiently without automation.
- 5) **Limited Accessibility to Insights:** Non-technical users lack tools to extract meaningful information from comments, restricting data-driven content strategies.

Trend Tracking: Monitors how sentiment changes over time, helping influencers identify how different content affects audience perception.

Additional Comment Analysis Features:

Word Cloud Visualization: Generates a word cloud showcasing the most frequently used words and phrases in the comments. Helps quickly identify trending topics, keywords, or recurring themes.

Average Comment Length: Calculates and displays the average length of comments, indicating the depth of audience engagement.

Average Comment Score: Calculates and displays the average comment score by summing up all the comments.

3. REVIEW OF RELATED WORK

Several sentiment analysis of social networks are performed such as YouTube and Reddit.

These researches affect comments, Reddit and other metadata collected from social networking sites from profile of users of public events and analysed to get significant and interesting insights about usage of social network by mass of individuals. The work most closely associated with ours is by Siersdorfer et al. They analyzed quite 6 million comments collected from 67,000 YouTube videos to identify the connection between comments, views, comment ratings and topic categories. The authors show promising leads to predicting the comment ratings of latest unrated comments by building prediction models using the already rated comments. Pang, Lee and Vaithyanathan perform sentiment analysis on 2053 movie reviews collected from the web Movie Database (IMDb). They examined the hypothesis that sentiment analysis are often treated as a special case of topic-based text classification. Their work depicted that standard machine learning techniques such as Naive Bayes or Support Vector Machines (SVMs) outperform manual classification techniques that involve human intervention.

However, the accuracy of sentiment classification falls in need of the accuracy of ordinary topic-based text categorization that uses such machine learning techniques. They reported that the simultaneous presence of positive and negative expressions (thwarted expectations) within the reviews make it difficult for the machine learning techniques to accurately predict the emotions.

Content Acquisition and Web Scraping

Data is collected using two primary sources: the YouTube Data API and web scraping techniques. The YouTube API provides structured access to video comments, while additional contextual data is enriched by Reddit dataset enhancing diversity and depth of training dataset.

ML Models for Content Processing

In machine learning we often combine different algorithms to get better and optimize results. Our main goal is to minimize loss function for which, one of the famous algorithm is XGBoost (Extreme boosting) technique which works by building an ensemble of decision trees sequentially where each new tree corrects the errors made by the previous one. It uses advanced optimization techniques and regularization methods that reduce overfitting and improve model performance.

LightGBM is an open-source high-performance framework developed by Microsoft. It is an ensemble learning framework that uses gradient boosting method which constructs a strong learner by sequentially adding weak learners in a gradient descent manner.

It's designed for efficiency, scalability and high accuracy particularly with large datasets. It uses decision trees that grow efficiently by minimizing memory usage and optimizing training time.

This enables LightGBM to outperform other frameworks in both speed and accuracy.

4. System Architecture

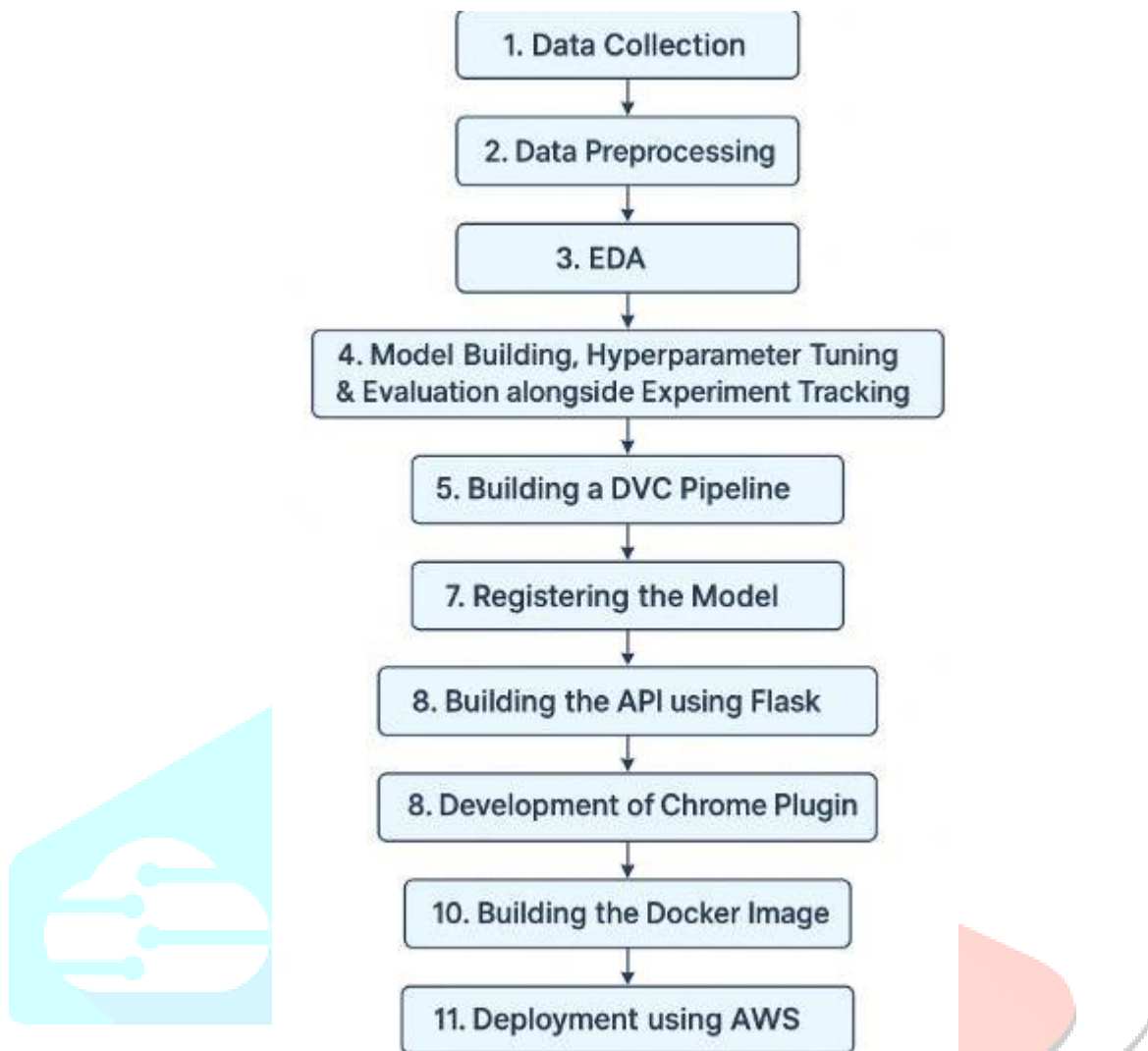
The architecture of the Chrome extension is modular, ensuring scalability and ease of integration.

The key components are:

- **Frontend Interface:** Embedded within the YouTube UI, this interface provides buttons and visual elements to trigger comment analysis and display results.
- **Background Script:** Manages communication between the extension's components and external APIs.

This is the core component of the Chrome extension responsible for coordinating and managing communication between the various parts of the extension and external services. Specifically, it acts as an intermediary that listens for events or actions triggered by the user interface (like a button click), and then performs tasks such as fetching data from the YouTube API or sending preprocessed comments to the sentiment analysis model. It also handles messaging between content scripts (which interact with the YouTube page directly)

- **Data Processor:** Handles all stages of textual preprocessing including data cleaning, tokenization, stop-word removal, stemming, and lemmatization. It also manages essential Natural Language Processing (NLP) tasks such as language detection, part-of-speech tagging, and syntactic parsing. This module is responsible for running the sentiment classification pipeline using two advanced gradient boosting algorithms—XGBoost and LightGBM. XGBoost is used for its robustness and high accuracy on sparse data, while LightGBM offers faster training speed and lower memory consumption. Together, they ensure the system can classify comments into categories like Positive, Negative, Neutral, or Spam with high efficiency and reliability across large-scale datasets.
- **Categorisation:** Identifying the comments on YouTube video through extension and categorizing the comments into Positive, Neutral and Negative with scores 1,0, -1 respectively.
- **Visualization Module:** Generates real-time word clouds, bar graphs, and sentiment distribution pie charts.
- **Storage Module:** Utilizes browser local storage or optional cloud storage to maintain a history of comment data and analysis.
- **Web Interface:** Browser extension offers a control panel. Separate sections for different visualisations and comments sentiment analysis, which enables user to engage on board.



5. METHODOLOGY

The analytical pipeline implemented in the extension includes:

MODEL BUILDING

Content Acquisition and Web Scraping: Comments are collected using two primary sources: the YouTube Data API and web scraping techniques. The YouTube API provides structured access to video comments, while additional contextual data is enriched by scraping Reddit discussions related to specific YouTube videos. Reddit threads often offer deeper discourse, humor, and critiques that complement YouTube comment data, making them valuable for improving model training and overall sentiment diversity. This multi-source approach ensures that our model generalizes better and captures a wider array of opinions and linguistic styles.

Text Preprocessing: Includes cleaning, tokenization, lowercasing, stop-word removal, stemming, and lemmatization.

- **Cleaning:** Removing unwanted characters, symbols, punctuation, or formatting issues.
- **Tokenization:** Splitting text into individual words or meaningful chunks (tokens).
- **Lowercasing:** Converting all characters to lowercase to maintain consistency and avoid duplicate tokens.
- **Stop-word Removal:** Eliminating common words (like “the”, “is”, “and”) that carry little semantic meaning, special care is taken to retain crucial negation words such as “not,” “no,” and “neither,” which are essential for accurately identifying negative sentiment. This ensures that the sentiment analysis process does not misinterpret the tone of comments that rely heavily on negation

- **Stemming:** Reducing words to their root forms (e.g., “running” → “run”), often by chopping off suffixes.
- **Lemmatization:** Similar to stemming but more accurate, as it considers the word’s meaning and part of speech to find the dictionary form (e.g., “better” → “good”).

These steps help standardize and simplify text data, making it easier for machine learning models to interpret and process it effectively.

Sentiment Classification: Comments are fed into trained XGBoost and LightGBM models that classify them as Positive, Negative, Neutral, or Spam. These gradient boosting models are known for their high performance in classification tasks.

XGBoost uses a regularized learning objective, which helps prevent overfitting and handles sparse data well. It builds additive decision trees sequentially and is highly customizable.

LightGBM, in contrast, utilizes a histogram-based algorithm and leaf-wise tree growth strategy, making it faster and more memory-efficient while maintaining high accuracy, especially on large-scale datasets. It can handle large datasets with a lower memory footprint and has faster training speed than other boosting algorithms.

Topic and Keyword Extraction: Frequent keywords and phrases are extracted for display via dynamic word clouds. These visualizations automatically update as new comment data is analysed, offering an intuitive way to explore user interest and trending topics. Word clouds serve as a powerful method to highlight prominent themes, helping users instantly grasp dominant sentiments and discussion areas.

Relevance Filtering: Machine learning classifiers filter out spam or irrelevant content, ensuring that only meaningful comments are visualized and analysed. The relevance filter is built upon supervised models trained on labelled datasets comprising both legitimate and spam comments, enabling high-precision filtering. This preprocessing phase ensures that the visualizations and sentiment analyses reflect authentic user feedback, thereby increasing the reliability of interpretations and reducing noise in data-driven decision-making processes.

Correction Of Discrepancies: Including empty entries, duplicates, and inconsistent labels.

Experimentation with Preprocessing Techniques using MLflow

To identify the most effective preprocessing strategy, MLflow was utilized for experiment tracking and model optimization. Several experiments were logged using MLflow to systematically evaluate combinations of tokenization strategies, n-gram ranges, and vocabulary sizes.

Key parameters evaluated included:

Text Vectorization Technique: Multiple tokenization methods were compared, and TF- IDF (Term Frequency-Inverse Document Frequency) emerged as the most effective in capturing contextual importance and word relevance within the comments.

Vocabulary Size (max_ features): Experimentation was conducted with various feature sizes, and it was determined that limiting the vocabulary to 10,000 words provided the optimal trade-off between model complexity and performance.

N-gram Range: To incorporate local context and word sequences, different n-gram configurations were tested. The trigram model (capturing sequences of three words) performed best, particularly in identifying nuanced sentiment expressions.

Also, the data had less number of data points for negative class. To tackle that oversampling was performed using Adasyn.

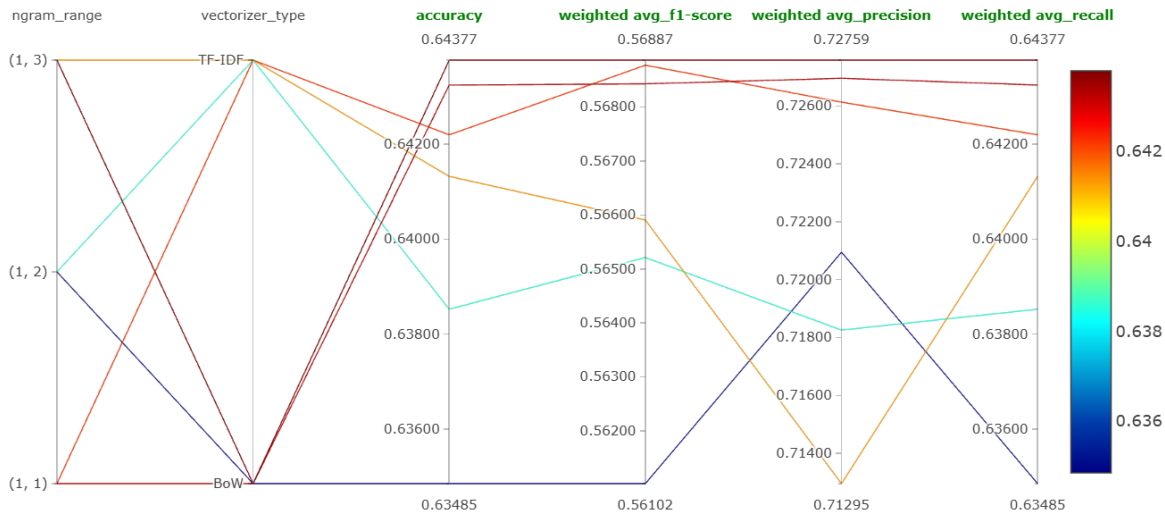


Fig 1: Compares the three n-grams with TD-IDF and Count Vectorizer

Model Selection and Hyperparameter Optimization

Following the identification of the most effective text preprocessing pipeline (TF-IDF vectorization with a trigram configuration and a vocabulary size of 10,000), the next step was to determine the best-performing machine learning model for the sentiment classification task.

Three advanced ensemble-based classifiers were selected:

XGBoost (Extreme Gradient Boosting)

LightGBM (Light Gradient Boosting Machine)

Each of these models is well-known for its performance on structured/tabular data and has proven effectiveness in text classification tasks when paired with engineered features like TF-IDF.

To ensure a fair and thorough comparison, hyperparameter optimization was carried out using Optuna, a powerful Bayesian optimization library. Hyperopt allowed for an efficient and intelligent search across the hyperparameter space, reducing the time and computational resources required compared to traditional grid search or random search techniques. The best run for each model was logged and compared.

Key Hyperparameters Tuned:

For XGBoost: learning rate, maximum depth, number of estimators, max depth, and regularization parameters.

For LightGBM: number of leaves, learning rate, max depth.

MLOps Workflow Integration

To ensure the scalability, reliability, and continuous improvement of the YouTube Comment Analysis Chrome Extension, the integration of a robust MLOps (Machine Learning Operations) workflow is essential. MLOps bridges the gap between machine learning development and operations, enabling seamless collaboration, automation, and deployment.

The following components are integrated into the MLOps pipeline:

Continuous Integration and Continuous Deployment (CI/CD) Pipelines for Model Updates

CI/CD pipelines automate the entire lifecycle of the machine learning models, from code integration to deployment. Whenever a new model version or improvement is committed to the repository, automated testing and validation processes are triggered. These pipelines ensure that changes do not introduce bugs or performance regressions. Tools such as GitHub Actions, Jenkins, or GitLab CI are employed to implement these pipelines, streamlining development and ensuring consistency across environments.

Automation of Model Training, Testing, and Deployment

The model training process is scheduled or triggered by events such as new data arrival or performance degradation detection. Automated scripts preprocess the data, retrain models (XGBoost and LightGBM), evaluate them against benchmarks, and deploy the best-performing model into production. This eliminates the need for manual intervention, reduces errors, and enables rapid iteration. Automated testing includes unit tests for preprocessing steps, validation on hold-out datasets, and model performance comparison using metrics like accuracy, precision, recall, and F1-score.

Monitoring and Version Control for Continuous Improvement

Post deployment monitoring ensures that the model maintains its performance in real-world scenarios. Tools like Prometheus, Grafana, and custom logging systems track key performance indicators (KPIs) such as prediction latency, model confidence scores, drift detection, and user feedback trends. Version control of both the code and model artifacts (using platforms like DVC or MLflow) enables traceability and rollback in case of performance issues. This also facilitates reproducibility and auditing, which are critical for maintaining trust and compliance.

Data and Pipeline Versioning with DVC

To manage data dependencies, pipeline stages, and outputs, DVC (Data Version Control) was used. DVC allowed version control for datasets and models, enabling reproducible ML pipelines that could be tracked and reverted easily.

The pipeline included the following stages:

Data Partitioning: Split the dataset into training, validation, and test sets, ensuring balanced class distributions.

Data Preprocessing: Applied text cleaning and TF-IDF vectorization using the best-found configuration (trigram n-grams, max features = 10,000).

Model Building: Trained selected models (Random Forest, XGBoost, LightGBM) using hyperparameter tuning via Hyperopt.

Model Evaluation: Generated evaluation metrics (accuracy, F1-score) and visual artifacts like confusion matrices.

Model Registry: Registered finalized models for deployment using MLflow and version-controlled storage.

Experiment Tracking with MLflow

MLflow was integrated across all stages to enable comprehensive tracking of:

- Experiments and model run.
- Hyperparameters and preprocessing configurations.
- Performance metrics and model artifacts.
- Model registry.

By implementing this MLOps framework, the YouTube Comment Analysis system becomes not only more robust and scalable but also easier to maintain and enhance over time.

Building the Chrome Plugin

1. Develop a Chrome extension to fetch YouTube comments

This involves creating a browser extension using HTML, CSS, and JavaScript that integrates directly into the YouTube video page. The extension uses YouTube's DOM structure or YouTube Data API to extract visible comments in real-time. It adds a user-friendly interface (such as a button or side panel) for users to initiate the analysis process without leaving the page. Permissions and manifest files are configured to ensure secure and smooth browser integration.

2. Connect the extension to a Flask backend for sentiment analysis

Once comments are fetched, the extension sends them to a Flask backend via HTTP requests (usually POST). The backend handles all data preprocessing and runs the sentiment analysis using trained XGBoost and LightGBM models. The server returns categorized sentiment data (e.g., Positive, Negative, Neutral, Spam) and other relevant metrics. This decoupled architecture allows for scalable updates and model improvements without changing the extension code.

3. Display insights using charts, sentiment breakdowns, and comment statistics

The returned sentiment data is rendered visually on the extension UI using JavaScript libraries like Chart.js or D3.js. Charts include pie charts for sentiment distribution, bar graphs for comment frequency, and word clouds for keyword trends. Additional statistics such as total comments analyzed, number of spam comments, or most frequent words help users interpret engagement patterns quickly and intuitively. This makes complex sentiment data accessible and actionable for creators, researchers, and analysts.

DEPLOYMENT OF PLUGIN

1. Deploy the Flask API on cloud servers

The Flask-based backend, responsible for handling comment processing and sentiment analysis, is hosted on a cloud platform such as AWS. This deployment ensures high availability, scalability, and secure access to the API endpoints. Containerization tools like Docker can be used for consistent deployment environments, and auto-scaling options can handle varying user loads effectively.

2. Ensure seamless communication between the extension and the backend

The Chrome extension communicates with the cloud-hosted Flask API via secure HTTP requests (e.g., RESTful POST/GET). CORS (Cross-Origin Resource Sharing) policies are configured properly to allow the browser extension to interact with the server without security issues. Consistent response structures and error-handling mechanisms are implemented to ensure smooth and reliable data exchange between the frontend and backend components.

3. Dockerizing the Application

The first step in our deployment plan is to containerize the Flask backend using Docker. This ensures that the application, along with its dependencies and environment, can run uniformly across various systems. A Dockerfile will be created to define the build environment and dependencies.

The Flask application will be encapsulated in a Docker container and tested locally to confirm proper functionality.

4. Pushing to AWS Elastic Container Registry (ECR)

Once the backend is successfully containerized, the Docker image will be uploaded to Amazon Elastic Container Registry (ECR) — a secure, scalable container image repository provided by AWS.

- Using AWS CLI, we will authenticate Docker with AWS and push the image to a private ECR repository.
- This allows us to manage versioning and access controls for our application container. Running the Application on AWS EC2

After uploading the image to ECR, the next step will be to deploy the application on an Amazon EC2 instance.

- The EC2 instance will be configured with Docker.
- We will pull the Docker image from ECR and run the container, exposing the necessary port for API communication.

6. IMPLEMENTATION

Data Acquisition and Preprocessing

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

dataset = pd.read_csv('/content/reddit_preprocessing.csv')

# Drop rows with NaN values in 'clean_comment'
cleaned_dataset = dataset.dropna()

# Separate features and target
X_cleaned = cleaned_dataset['clean_comment']
y_cleaned = cleaned_dataset['category']

# Split the cleaned data into train and test sets (80-20 split)
X_train_cleaned, X_test_cleaned, y_train_cleaned, y_test_cleaned = train_test_split(X_cleaned, y_cleaned, test_size=0.2, random_state=42)

# Apply TfidfVectorizer with trigram setting and max_features=1000
tfidf_cleaned = TfidfVectorizer(ngram_range=(1, 3), max_features=1000)

# Fit the vectorizer on the training data and transform both train and test sets
X_train_tfidf_cleaned = tfidf_cleaned.fit_transform(X_train_cleaned)
X_test_tfidf_cleaned = tfidf_cleaned.transform(X_test_cleaned)
```

Content Processing Agent Implementation

For the processing agent, we implemented multiple summarization approaches

Extractive Summarization:

- Text Rank algorithm implementation for baseline summarization
- TF-IDF based extraction for comparison purposes

```

# Separate features and target
X_cleaned = cleaned_dataset['clean_comment']
y_cleaned = cleaned_dataset['category']

# Split the cleaned data into train and test sets (80-20 split)
X_train_cleaned, X_test_cleaned, y_train_cleaned, y_test_cleaned = train_test_split(X_cleaned, y_cleaned, test_size=0.2, random_state=42)

# Apply TfidfVectorizer with trigram setting and max_features=1000
tfidf_cleaned = TfidfVectorizer(ngram_range=(1, 3), max_features=1000)

# Fit the vectorizer on the training data and transform both train and test sets
X_train_tfidf_cleaned = tfidf_cleaned.fit_transform(X_train_cleaned)
X_test_tfidf_cleaned = tfidf_cleaned.transform(X_test_cleaned)

```

MODEL IMPLEMENTATION

```

] import lightgbm as lgb
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import GridSearchCV
import optuna

Show hidden output

] # Function to optimize LightGBM hyperparameters
def objective(trial):
    # Define hyperparameters to be tuned
    param = {
        "objective": "multiclass",
        "num_class": 3, # Assuming 3 categories (-1, 0, 1)
        "learning_rate": trial.suggest_float("learning_rate", 1e-3, 1e-1),
        "n_estimators": trial.suggest_int("n_estimators", 50, 500),
        "max_depth": trial.suggest_int("max_depth", 3, 20),
        "metric": "multi_logloss",
        "is_unbalance": True,
        "class_weight": "balanced",
    }

    # Define the LightGBM model with the trial parameters
    model = lgb.LGBMClassifier(**param)

    # Perform cross-validation
    scores = cross_val_score(model, X_train_tfidf_cleaned, y_train_cleaned, cv=3, scoring='accuracy')

    # Return the average score across folds
    return scores.mean()

] # Create an Optuna study to optimize the hyperparameters
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=50)

Show hidden output

] # Extract the best hyperparameters
best_params = study.best_params
best_params

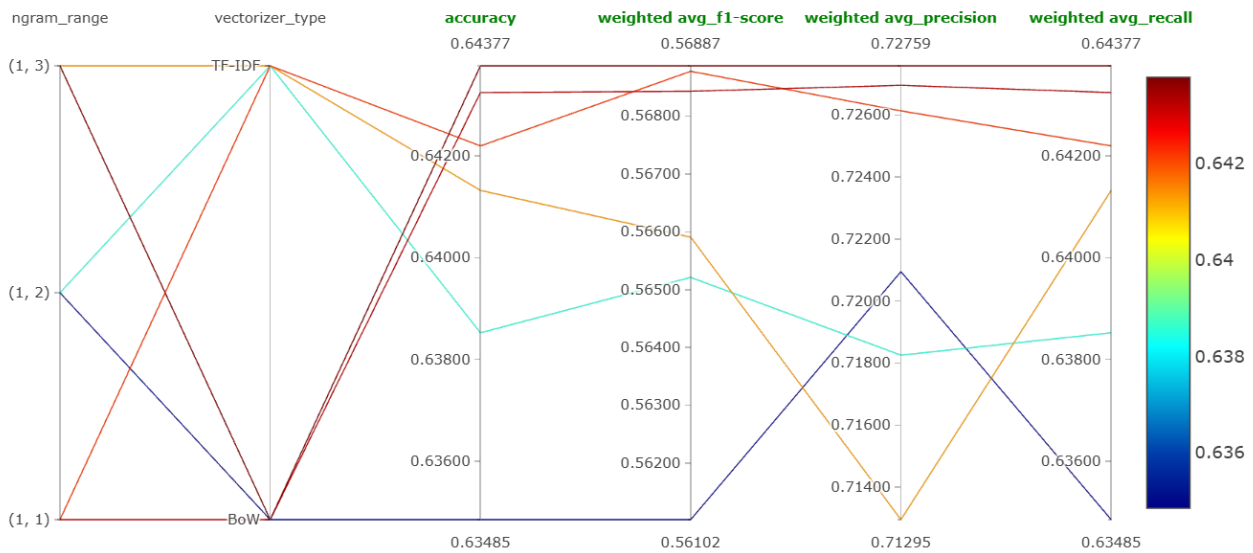
{'learning_rate': 0.08081298097796712, 'n_estimators': 367, 'max_depth': 20}

] best_model = lgb.LGBMClassifier(

    objective='multiclass',
    num_class=3,
    metric="multi_logloss",
    is_unbalance=True,
    class_weight="balanced",
    reg_alpha=0.1, # L1 regularization
    reg_lambda=0.1, # L2 regularization
    learning_rate=0.08,
    max_depth=20,
    n_estimators=367
)

] # Fit the model on the resampled training data
best_model.fit(X_train_tfidf_cleaned, y_train_cleaned)

```



FRONTEND IMPLEMENTATION

```
1 #backend
2 import matplotlib
3 matplotlib.use('Agg')
4
5 from flask import Flask, request, jsonify, send_file
6 from flask_cors import CORS
7 import io
8 import matplotlib.pyplot as plt
9 from wordcloud import WordCloud
10 import mlflow
11 import numpy as np
12 import joblib
13 import re
14 import pandas as pd
15 from nltk.corpus import stopwords
16 from nltk.stem import WordNetLemmatizer
17 from mlflow.tracking import MlflowClient
18 import matplotlib.dates as mdates
19
20 app = Flask(__name__)
21 CORS(app) # Enable CORS for all routes
22
23 # Define the preprocessing function
24 def preprocess_comment(comment):
25     """Apply preprocessing transformations to a comment."""
26     try:
27         # Convert to lowercase
28         comment = comment.lower()
29
30         # Remove trailing and leading whitespaces
31         comment = comment.strip()
32
33         # Remove newline characters
34         comment = re.sub(r'\n', ' ', comment)
35
36         # Remove non-alphanumeric characters, except punctuation
```



```

def load_model_and_vectorizer(model_name, model_version, vectorizer_path):
    # Set MLflow tracking URI to your server
    mlflow.set_tracking_uri("http://ec2-13-229-222-187.ap-southeast-1.compute.amazonaws.com:5000/") # Replace with your MLflow tracking
    client = MLflowClient()
    model_uri = f"models://{model_name}/{model_version}"
    model = mlflow.sklearn.load_model(model_uri)
    vectorizer = joblib.load(vectorizer_path) # Load the vectorizer
    return model, vectorizer

# Initialize the model and vectorizer
model, vectorizer = load_model_and_vectorizer("my_model1", "2", "./tfidf_vectorizer.pkl") # Update paths and versions as needed

@app.route('/')
def home():
    return "Welcome to our flask api"

@app.route('/predict_with_timestamps', methods=['POST'])
def predict_with_timestamps():
    data = request.json
    comments_data = data.get('comments')

    if not comments_data:
        return jsonify({"error": "No comments provided"}), 400

    try:
        comments = [item['text'] for item in comments_data]
        timestamps = [item['timestamp'] for item in comments_data]

        # Preprocess each comment before vectorizing
        preprocessed_comments = [preprocess_comment(comment) for comment in comments]

        # Transform comments using the vectorizer
        transformed_comments = vectorizer.transform(preprocessed_comments)

```

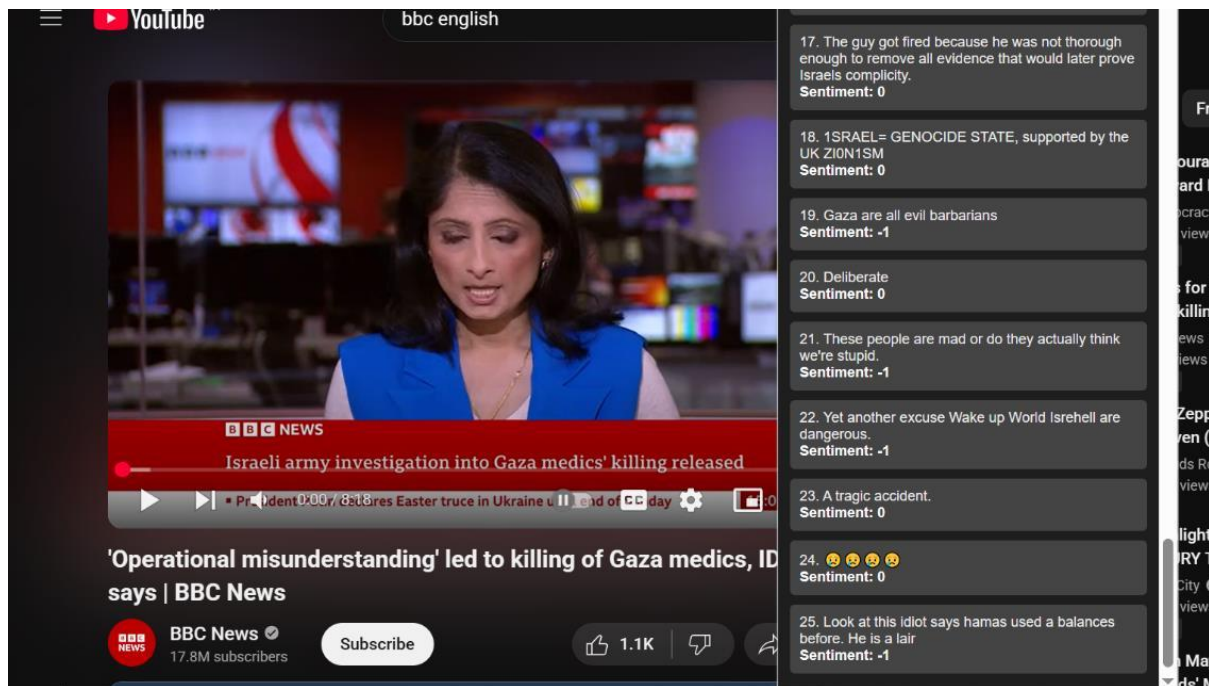
7. RESULTS

The implementation and evaluation of our Comment Analyser Extension yielded several notable findings across multiple dimensions

PERFORMANCE METRICS

DATASET	ACCURACY	PRECISION (-1,0,1)	RECALL (-1,0,1)	F1 SCORE (-1,0,1)
TRAINING	0.93	0.91,0.88,0.98	0.90,0.98,0.90	0.91,0.93,0.94
TEST	0.87	0.81,0.84,0.92	0.78,0.97,0.83	0.79,0.90,0.83

- Training Set shows high performance with an accuracy of 93%, indicating the model learns well on the data it was trained on.
- Test Set performance drops to 87%, which is still strong, but suggests slight overfitting.
- Class 0 (Neutral) consistently shows the highest recall, especially on the test set (0.97), meaning the model is very good at identifying neutral comments.
- Class -1 (Negative) has the lowest recall on the test set (0.78), indicating room for improvement in identifying negative sentiment.



The screenshot shows a YouTube video player for a BBC News video titled "Israeli army investigation into Gaza medics' killing released". The video player includes a progress bar, play/pause controls, and a 'Subscribe' button. To the right of the video player, a list of comments is displayed, each with its text and a sentiment score. The comments are numbered 17 through 25. The sentiment scores are: 17 (0), 18 (0), 19 (-1), 20 (0), 21 (-1), 22 (-1), 23 (0), 24 (0), and 25 (-1).

Comment ID	Comment Text	Sentiment
17	The guy got fired because he was not thorough enough to remove all evidence that would later prove Israels complicity.	0
18	1 ISRAEL= GENOCIDE STATE, supported by the UK ZIONISM	0
19	Gaza are all evil barbarians	-1
20	Deliberate	0
21	These people are mad or do they actually think we're stupid.	-1
22	Yet another excuse Wake up World Isrehell are dangerous.	-1
23	A tragic accident.	0
24	🤔🤔🤔🤔	0
25	Look at this idiot says hamas used a balances before. He is a lair	-1

The sentiment analysis model classified YouTube comments into positive, negative, or neutral categories with a high accuracy. It was integrated with a Flask backend and a Chrome extension, allowing real-time analysis and visualization through charts and word clouds. MLOps tools like DVC and MLflow were used for version control, pipeline management, and experiment tracking, ensuring reproducibility and scalability. Dockerization is complete, and deployment on AWS is planned. The project successfully showcases the effectiveness of combining ML and MLOps in a practical application.

8. CONCLUSION

This project provides an efficient, automated solution for analyzing YouTube comments, helping users gain insights into audience sentiment. By leveraging machine learning, the system eliminates the need for manual comment analysis, making it a valuable tool for content creators, marketers, and researchers.

The integration of MLOps ensures that the model remains up-to-date, scalable, and reliable, allowing for continuous improvements in accuracy and efficiency. The Chrome plugin makes sentiment analysis accessible and user-friendly, enabling real-time feedback analysis without requiring technical expertise. By combining sentiment analysis, data visualization, and MLOps, this project bridges the gap between raw audience feedback and actionable insights, empowering users to make informed decisions based on audience reactions.

REFERENCES

- [1] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.
- [2] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Proceedings of the 31st International Conference on Neural Information Processing Systems.
- [3] Hutto, C. J., & Gilbert, E. (2014). VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. Proceedings of the International AAAI Conference on Web and Social Media (ICWSM).
- [5] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of NAACL-HLT.
- [6] Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly Media.
- [7] Google Developers. YouTube Data API v3.

[8] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

