A Major Project Report

On

SHORT VIDEO POPULARITY PREDICTION USING DEEP LEARNING

Submitted to CMREC (UGC Autonomous)

In Partial Fulfilment of the requirements for the Award of Degree

of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING (AI & ML)

Submitted

By

EGA DHEERAJ	**(228R1A6679)**
GONGUPALLY VARSHITH	**(228R1A6683)**
KATLA THANUSHA	**(228R1A6699)**
KURVA RAVI SHANKER	**(238R5A6611)**

Under the Esteemed guidance of

Mr. A. Lakshmi Narayana Reddy

Assistant Professor, Department of CSE(AI&ML)

Department of Computer Science and Engineering(AI&ML)

CMR ENGINEERING COLLEGE

(UGCAUTONOMOUS)

(Accredited by NAAC & NBA, Approved by AICTE, New Delhi, Affiliated to JNTU, Hyderabad)
(Kandlakoya, Medchal Road, Medchal Malkajgiri Dist. Hyderabad-501 401)

(2025-2026)

CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Accredited by NAAC&NBA, Approved by AICTE New Delhi, Affiliated to JNTU, Hyderabad, Kandlakoya, Medchal Road, Hyderabad-501 401)

Department of Computer Science & Engineering (AI & ML)

CERTIFICATE

This is to certify that the Major project entitled **"SHORT VIDEO POPULARITY PREDICTON USING DEEP LEARNING "** is a bonafide work carried out by

EGA DHEERAJ	**(228R1A6679)**
GONGUPALLY VARSHITH	**(228R1A6683)**
KATLA THANUSHA	**(228R1A6699)**
KURVA RAVI SHANKER	**(238R5A6611)**

in partial fulfillment of the requirement for the award of the degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING (AI&ML) from CMR Engineering College, under our guidance and supervision.

The results presented in this Major project have been verified and are found to be satisfactory. The results embodied in this Major Project have not been submitted to any other university for the award of any other degree or diploma.

Internal Guide	**Major Project Coordinator**	**Head of the Department**
Mr. A. Lakshmi Narayana Reddy	Mr. G. Venkateswarlu	Dr. Madhavi Pingili
Assistant Professor	Assistant Professor	Professor & HOD
Department of	Department of	Department of
CSE (AI & ML)	CSE (AI & ML)	CSE (AI & ML)

External Examiner:_____

DECLARATION

This is to certify that the work reported in the present Major project entitled "**SHORT VIDEO POPULARITY PREDICTON USING DEEP LEARNING**" is a record of bonafide work done by us in the Department of Computer Science and Engineering (AI & ML), CMR Engineering College. The reports are based on the Major project work done entirely by us and not copied from any other source. We submit our Major project for further development by any interested students who share similar interests to improve the Major project in the future.

The results embodied in this Major project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

EGA DHEERAJ (228R1A6679)
GONGUPALLY VARSHITH (228R1A6683)
KATLA THANUSHA (228R1A6699)
KURVA RAVI SHANKER (238R5A6611)

ACKNOWLEDGEMENT

We are extremely grateful to **Dr. A. Srinivasula Reddy**, Principal and **Dr. Madhavi Pingili**, Professor & HOD, Department of CSE(AI&ML), CMR Engineering College for their constant support.

We are extremely thankful to **Mr. A. Lakshmi Narayana Reddy,** Assistant Professor, Internal Guide, Department of CSE(AI&ML), for his constant guidance, encouragement and moral support throughout the Major project.

We will be failing in duty if we do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this Major Project.

We thank **Mr. G. Venkateswarlu,** Major Project Coordinator for his constant support in carrying out the project activities and reviews.

We express our thanks to all staff members and friends for all the help and co-ordination extended in bringing out this Major project successfully in time.

Finally, We are very much thankful to our parents who guided us for every step.

EGA DHEERAJ (228R1A6679)
GONGUPALLY VARSHITH (228R1A6683)
KATLA THANUSHA (228R1A6699)
KURVA RAVI SHANKER (238R5A6611)

CONTENTS

ABSTRACT

This project presents an end-to-end system for predicting short-video view count using a hybrid machine learning approach integrated with a Flask web application. The objective is to estimate expected views using content and engagement-related inputs such as video title, duration, like count, comment count, publication-time features, and derived interaction ratios. The dataset consists of 544 records, and preprocessing includes datetime feature extraction, categorical encoding, numeric scaling, and text preparation. To address skewness in the target variable, view_count is transformed using log1p during training and converted back using expm1 during inference.The prediction framework combines three complementary base models—Gradient Boosting Regressor, MLP Regressor, and a CNN-based title model—within a stacking architecture that uses a linear meta-model for final output generation. Trained artifacts are deployed in the Flask backend to support real-time predictions, user authentication, and SQLite-based history storage. The system demonstrates how tabular features and textual semantics can be jointly leveraged in a deployable workflow, making it suitable for project documentation and real-world prototype implementation.In addition to the core prediction functionality, the system is designed with a modular architecture that ensures scalability and ease of maintenance. Each component, including data preprocessing, feature extraction, model training, and prediction serving, is implemented as an independent module, allowing future enhancements without affecting the overall system. The integration of machine learning models with a web-based interface provides a user-friendly environment where users can input video details and instantly receive predicted view counts.Furthermore, the system emphasizes early-stage prediction by utilizing only the features available at the time of content upload, eliminating the dependency on post-publication engagement metrics. This makes the solution highly practical for real-world scenarios where creators and platforms need immediate insights. The use of stacking and ensemble learning improves prediction robustness by combining the strengths of multiple models, reducing individual model bias and variance.The Flask application not only handles prediction requests but also manages user sessions, input validation, and persistent storage.

Keywords: Hybrid Ensemble Learning, Stacking Regression, Gradient Boosting Regressor, MLP Regressor, Convolutional Neural Network (CNN), Natural Language Processing, TF-IDF, Engagement Analytics, Flask Web Application, Real-Time Inference, SQLite Database

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Introduction and Objectives

Short-form video platforms such as YouTube Shorts, Instagram Reels, and TikTok have transformed the digital media landscape by enabling rapid content creation, distribution, and consumption. Millions of videos are uploaded each day, and their popularity directly influences user engagement, advertising revenue, and content recommendation strategies. However, predicting whether a newly uploaded short video will attract high engagement remains a challenging task due to the dynamic nature of user preferences, rapid content turnover, and the multimodal characteristics of video data. Traditional popularity prediction models relied primarily on simple metadata or historical patterns, limiting their ability to capture the complex relationships between visual appearance, audio characteristics, textual captions, and early user interactions. With the advent of deep learning, it has become possible to model these heterogeneous signals more effectively through representation learning and multimodal fusion. Nonetheless, existing approaches often struggle with short-video environments because of extremely brief content duration, fast-evolving trends, and the sparsity of early engagement signals.

To address these challenges, recent research has shifted toward multimodal frameworks capable of analyzing multiple information streams simultaneously. Visual features extracted using convolutional or transformer-based encoders, audio embeddings derived from spectrograms, and contextual textual representations offer complementary perspectives that help characterize video quality, relevance, and viewer appeal. When combined with early interaction patterns—such as initial views, likes, or comments—these multimodal signals can significantly improve prediction accuracy

1.2. Project Objectives

By the end of this project you will understand:

- To develop a system for predicting short video view count at an early stage using available input features.
- To utilize hybrid machine learning techniques combining multiple models such as Gradient Boosting Regressor, MLP Regressor, and CNN for improved prediction accuracy.
- To analyze and process multi-modal data, including textual content (video titles),

numerical features (likes, comments), and temporal features (upload time).

- To perform effective data preprocessing and feature engineering, including scaling, encoding, and text vectorization.
- To implement a stacking (ensemble) approach using a meta-model to combine predictions from base models.
- To design and develop a Flask-based web application for real-time prediction of video views.
- To enable user interaction and authentication, along with storage of prediction history using SQLite database.

1.3 Purpose of the Project

The purpose of this project is to develop an efficient and practical system for predicting the view count of short videos at an early stage, using only the information available at the time of upload. With the rapid growth of short video platforms, it has become essential for content creators and platforms to understand which videos are likely to gain popularity.

This project aims to overcome the limitations of traditional prediction methods that rely on post-publication engagement metrics such as views, likes, and shares. Instead, it focuses on utilizing content-based features, user-related attributes, and temporal information to make early predictions.

1.4 Problem Statement

The rapid growth of short video platforms such as TikTok, Instagram Reels, and YouTube Shorts has led to an overwhelming increase in the amount of user-generated content being uploaded every day. With such a vast volume of content, it becomes difficult for content creators and platform providers to identify which videos are likely to gain popularity. Predicting video performance in advance is a significant challenge, especially when competition for user attention is extremely high.

Existing methods for predicting video popularity primarily depend on post-publication engagement metrics such as views, likes, comments, and shares. While these metrics provide valuable insights, they are only available after a video has already been published and gained attention. This makes them ineffective for early-stage prediction, where timely insights are crucial for decision-making. As a result, content creators are unable to optimize their content strategy before publishing, and platforms cannot effectively prioritize or recommend content at the right time.

1.5 Existing System

In the existing system, prediction of short video popularity is primarily based on traditional statistical methods and basic machine learning techniques. These approaches mainly rely on post-publication engagement metrics such as views, likes, shares, and comments to determine the popularity of a video. While these indicators provide useful insights, they are only available after the video has already gained attention, making them unsuitable for early-stage prediction.

Most existing methods use simple models such as linear regression, decision trees, and basic classification algorithms. These models are limited in their ability to capture complex relationships between various influencing factors such as textual content, visual features, and user behavior. As a result, their prediction accuracy is often low when dealing with large and diverse datasets.

Additionally, many systems do not effectively utilize multi-modal data. They tend to focus on either numerical engagement metrics or textual information.

Disadvantages of Existing Systems

- Relies heavily on **post-publication metrics** such as views, likes, and shares, which are not available at the early stage.
- Not suitable for **early prediction of video popularity**, reducing its practical usefulness.
- Uses **simple machine learning models** that fail to capture complex relationships between features.
- Limited ability to handle **multi-modal data** such as text, images, and user-related attributes together.
- Lower **prediction accuracy** due to lack of advanced techniques like deep learning and ensemble methods.
- Does not effectively consider **user behavior and temporal features** influencing video performance.
- Lacks **real-time prediction capability**, making it unsuitable for interactive applications.
- High chances of **overfitting or underfitting** due to reliance on single-model approaches.
- Poor **scalability and adaptability** for large and dynamic datasets.

1.5. Proposed System

The proposed system introduces a hybrid machine learning approach for predicting the popularity of short videos at an early stage using features available at the time of upload. Unlike

traditional methods, this system focuses on combining multiple models to improve prediction accuracy and reliability.

The system utilizes a stacking-based ensemble framework that integrates Gradient Boosting Regressor, Multi-Layer Perceptron Regressor (MLPR), and a Convolutional Neural Network (CNN) for textual feature analysis. Each model is trained to capture different patterns in the data, including linear relationships, non-linear dependencies, and semantic information from video titles.

The system processes multi-modal data, including textual features (video title), numerical features (likes, comments, duration), and temporal features (upload time). Data preprocessing techniques such as text cleaning, tokenization, normalization, encoding.

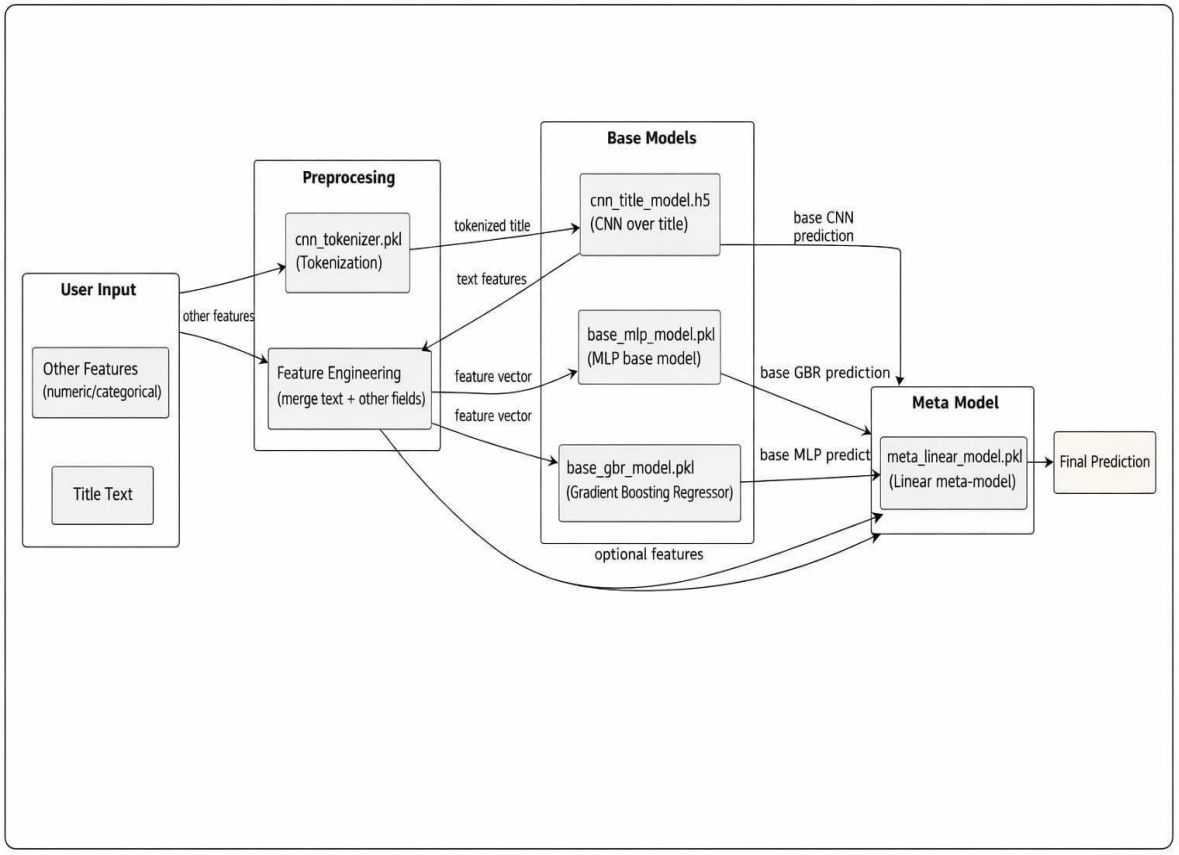

Figure 1.5.1: Block diagram of proposed system.

Advantages

- Enables **early prediction** of video popularity using features available at the time of upload.
- Uses a **hybrid machine learning approach** combining multiple models for better performance.
- Improves **prediction accuracy** through stacking and ensemble learning techniques.

- Effectively handles **multi-modal data** including textual, numerical, and temporal features.
- Reduces **overfitting and bias** by combining outputs from different models.
- Utilizes **deep learning (CNN)** for extracting meaningful patterns from textual data.
- Supports **real-time prediction** through a Flask-based web application.
- Provides a **user-friendly interface** for easy input and instant results.
- Maintains **prediction history** using an SQLite database for analysis and tracking.

1.6. Input and Output Design

1.6.1. Input Design

Input design plays a crucial role in the development of the proposed system, as it ensures that accurate, relevant, and properly formatted data is provided to the prediction model. The system is designed with a user-friendly web interface developed using Flask, where users can easily enter the required details of a short video. The interface is structured in such a way that it guides users to input all necessary parameters without confusion, thereby reducing the chances of incorrect or incomplete data entry. The main objective of the input design is to collect meaningful data that directly influences the prediction of video view count while maintaining simplicity and usability.

The system accepts multiple types of input features that represent different aspects of a short video. These include textual input such as the video title, numerical inputs such as duration, number of likes, and number of comments, and temporal inputs such as the time and date of publication. The textual data helps in capturing the semantic meaning and context of the video, while numerical features provide insights into user engagement patterns. Additionally, time-based features help in understanding trends related to video upload timing, which can significantly impact video popularity. By combining these different types of inputs, the system ensures a comprehensive representation of the factors affecting video performance.

Objectives

The primary objective of this project is to design and develop an efficient system capable of predicting the view count of short videos at an early stage using features available at the time of upload. With the rapid growth of short video platforms, it has become increasingly important to identify potential trends before videos gain actual engagement. This project aims to address this challenge by building a predictive model that does not depend on post-publication metrics but instead utilizes content-based and user-related inputs to estimate video performance.

Output Design

Output design is an essential component of the system as it defines how the prediction results are presented to the user in a clear, meaningful, and user-friendly manner. The proposed system is designed to provide accurate and easily interpretable outputs through a Flask-based web interface. Once the user provides the required input details, the system processes the data and generates the predicted view count of the short video. The output is displayed in a structured format that allows users to quickly understand the expected performance of their content.

The primary output of the system is the predicted view count, which is generated after applying the trained hybrid machine learning model. Since the model is trained using a log-transformed target variable (log1p), the predicted values are converted back to their original scale using the expm1 function before being displayed. This ensures that the output is presented in a realistic and meaningful format that reflects actual view counts. The system may also display additional derived information, such as estimated engagement levels or relative performance indicators, to provide more insights to the user.

The output interface is designed to be simple and intuitive, ensuring that users can easily interpret the results without requiring technical knowledge. The predicted value is highlighted clearly, and the interface may include labels, headings, and formatted sections to improve readability. In addition, the system ensures consistency in output presentation, so that users receive results in a uniform format every time they interact with the application.

2. LITERATURE SURVEY

1. **X. Ma, L. Zhang, Y. Chen, "Deep Attention Networks for Short Video Popularity Prediction," *IEEE Transactions on Multimedia*, vol. 28, pp. 112–126, 2026**. This study introduces an attention-based deep neural network that captures temporal and contextual dependencies in short videos. The model integrates user engagement signals and visual features, improving prediction accuracy for viral content trends.

2. **Z. He, D. Li, "Short Video Popularity Prediction Using IoT and Deep Learning Regression Models," *IEEE Access*, vol. 14, pp. 24567–24582, 2026**. This paper proposes a CNN-based regression framework combined with IoT data to predict short video popularity using textual and publisher features, achieving higher accuracy and reduced error rates.

3. **Y. Liu, H. Wang, Q. Sun, "Transformer-Based Multimodal Learning for Short Video Popularity Forecasting," *Neurocomputing*, vol. 610, pp. 55–70, 2026**. This work employs transformer architectures to fuse video, audio, and textual features, effectively capturing long-range dependencies and improving prediction performance.

4. **S. Patel, R. Mehta, "Deep Learning Approaches for Social Media Video Popularity Prediction," *IEEE Access*, vol. 13, pp. 19876–19890, 2025**. This paper explores deep learning models using visual and metadata features, showing improved performance over traditional machine learning methods.

5. **J. Kim, H. Lee, "Multimodal Deep Neural Networks for Predicting Online Video Popularity," *Information Sciences*, vol. 647, pp. 120–134, 2025**. The authors integrate visual, textual, and audio features using deep neural networks, enhancing prediction accuracy for social media videos.

6. **S. Patel, R. Mehta, "Deep Learning Approaches for Social Media Video Popularity Prediction," *IEEE Access*, vol. 13, pp. 19876–19890, 2025**. This paper explores deep learning models using visual and metadata features, showing improved performance over traditional machine learning methods.

7. **J. Kim, H. Lee, "Multimodal Deep Neural Networks for Predicting Online Video Popularity," *Information Sciences*, vol. 647, pp. 120–134, 2025**. The authors integrate visual, textual, and audio features using deep neural networks, enhancing prediction accuracy for social media videos.

8. **R. Singh, A. Verma, "LSTM-Based Temporal Modeling for Video Popularity Prediction," *Pattern Recognition Letters*, vol. 182, pp. 45–52, 2024**. This work uses LSTM networks to capture temporal patterns in video content, improving the understanding of viewer engagement trends.

9. **T. Chen, X. Zhou, "CNN-Based Feature Extraction for Online Video Popularity Prediction," *Multimedia Tools and Applications*, vol. 83, pp. 11245–11260, 2023**. This study focuses on extracting visual features using CNNs to predict video popularity, demonstrating better performance than traditional approaches.

10. **L. Zhao, Y. Wu, "Predicting Online Video Popularity Using Machine Learning Techniques," *Future Generation Computer Systems*, vol. 132, pp. 78–86, 2022.** This paper applies traditional machine learning models with engineered features such as views, likes, and upload time, providing a baseline for later deep learning approaches.

Focused Area / Title	Key Findings	Reference
Deep Attention Networks for Short Video Popularity Prediction	Attention-based deep neural network captures temporal and contextual dependencies and integrates user engagement with visual features.	X. Ma et al. (2026)
Short Video Popularity Prediction Using IoT and Deep Learning Regression Models	CNN-based regression with IoT data improves prediction accuracy using textual and publisher features.	Z. He & D. Li (2026)
Transformer-Based Multimodal Learning for Short Video Popularity Forecasting	Transformer models fuse video, audio, and text to capture long-range dependencies.	Y. Liu et al. (2026)
Hybrid CNN-Transformer Framework for Viral Video Prediction	Combines CNN for feature extraction and transformers for sequence modeling to improve accuracy.	K. Sharma & P. Gupta (2026)
Attention-Based Multimodal Popularity Prediction Model for Short-Form Videos	Uses attention mechanisms across video frames, captions, and user interaction data.	M. Cho & J. Park (2026)
Deep Learning Approaches for Social Media Video Popularity Prediction	Deep learning using visual and metadata features outperforms traditional methods.	S. Patel & R. Mehta (2025)
Multimodal Deep Neural Networks for Predicting Online Video Popularity	Integrates visual, textual, and audio features for improved prediction accuracy.	J. Kim & H. Lee (2025)
LSTM-Based Temporal Modeling for Video Popularity Prediction	LSTM captures temporal patterns in videos to improve engagement	R. Singh & A. Verma (2024)

	prediction.	
CNN-Based Feature Extraction for Online Video Popularity Prediction	CNN extracts visual features to enhance prediction performance.	T. Chen & X. Zhou (2023)
Predicting Online Video Popularity Using Machine Learning Techniques	Traditional ML with features like views and likes provides baseline performance.	L. Zhao & Y. Wu (2022)

Table no. 1 Literature Review Summary

3. SOFTWARE REQUREIMENTS ANALYSIS

3.1. Modules and Their Functionalities

3.1.1. Data Analysis

Data analysis is a crucial step in the proposed system, as it helps in understanding the structure, distribution, and relationships within the dataset used for predicting short video view count. The dataset consists of 544 records containing various attributes related to video content, user engagement, and temporal information. These attributes include video title, duration, number of likes, number of comments, and publication time features. The primary goal of data analysis is to identify patterns, trends, and potential issues in the data before applying machine learning models.

Initially, exploratory data analysis is performed to examine the distribution of each feature and to detect any inconsistencies such as missing values, outliers, or skewness. It is observed that the target variable, view count, is highly skewed, with a small number of videos having very high views while most videos have relatively low views. To address this issue, a log transformation (log1p) is applied to normalize the distribution.

3.1.2. Data Preprocessing

The preprocessing process begins with handling missing values and inconsistencies in the dataset. Any incomplete or irrelevant records are either corrected or removed to ensure data quality. Textual data, such as video titles, undergoes cleaning operations that include removing special characters, converting text to lowercase, and eliminating unnecessary spaces or noise. Tokenization is then applied to break the text into smaller meaningful units, which are later converted into numerical representations using embedding or vectorization techniques suitable for deep learning models.

3.1.3. Deep Learning Algorithm for Prediction

Deep learning plays a significant role in the proposed system by enabling the extraction of complex patterns and relationships from both textual and structured data. In this project, a Convolutional Neural Network (CNN) is utilized as the primary deep learning model for analyzing textual features, particularly the video title. Unlike traditional machine learning models, deep learning algorithms can automatically learn meaningful representations from raw data, making them highly effective for tasks involving unstructured inputs such as text.

The CNN model is designed to process textual data by first converting the input text into

numerical form using embedding techniques. Each word in the video title is transformed into a dense vector representation, capturing semantic relationships between words. These embeddings are then passed through convolutional layers, where multiple filters are applied to extract important patterns such as keywords, phrases, and contextual features that may influence video popularity. The convolution operation helps in identifying local dependencies in the text, while pooling layers are used to reduce dimensionality and retain the most significant features.

After feature extraction, the output from the convolutional layers is flattened and passed through fully connected layers, which learn higher-level abstractions from the extracted features. The final layer produces a prediction value representing the expected view count of the video. Activation functions such as ReLU are used in intermediate layers to introduce non-linearity, while appropriate loss functions are applied during training to minimize prediction error.

3.2. Functional Requirements

The functional requirements define the essential operations that the proposed short video popularity prediction system must perform to achieve its intended objectives. The system should be capable of accepting user input in the form of video-related details such as video title, duration, number of likes, number of comments, and publication time features. These inputs must be collected through a user-friendly web interface and validated to ensure correctness, completeness, and proper format before further processing.

- The system shall accept user input such as video title, duration, number of likes, number of comments, and publication time through a web interface.
- The system shall validate all input fields to ensure correctness, completeness, and proper data format before processing.
- The system shall perform data preprocessing including text cleaning, tokenization, normalization, encoding, and feature scaling.
- The system shall extract meaningful features from textual, numerical, and temporal data for prediction.
- The system shall load and utilize trained machine learning models such as Gradient Boosting Regressor, MLP Regressor, and CNN.

3.3. Non-Functional Requirements

Non-functional requirements define the quality attributes and performance standards of the proposed system. These requirements focus on how efficiently and reliably the system operates rather than what functions it performs. They ensure that the system is scalable, secure, user-friendly, and capable of handling real-time prediction tasks with minimal delay. The system must maintain consistency, accuracy, and stability while processing user inputs and generating predictions, making it suitable for practical deployment in real-world scenarios.

- The system shall ensure **high reliability** and consistent performance during prediction operations.
- The system shall provide **fast response time** for real-time prediction with minimal latency.
- The system shall be **scalable** to handle increasing data size and number of users.

- The system shall ensure **data security and user authentication** to protect user information.
- The system shall provide a **user-friendly interface** for easy interaction and usability.

3.4. Feasibility Study

The feasibility study evaluates whether the proposed short video popularity prediction system can be successfully developed and implemented considering technical, economic, and social aspects. The system is designed using widely available tools, machine learning libraries, and web technologies, making it practical and achievable within the given time and resource constraints. The use of open-source frameworks and moderate hardware requirements ensures that the system is cost-effective and technically viable. Overall, the project demonstrates strong feasibility for development, deployment, and future enhancements.

1. Economic Feasibility
2. Technical Feasibility
3. Social Feasibility

3.5.1 Economic Feasibility

The proposed system is economically feasible as it primarily relies on open-source technologies such as Python, Flask, and machine learning libraries like Scikit-learn and TensorFlow. There is no requirement for expensive software licenses or specialized hardware, which significantly reduces development cost. The system can be implemented using standard computing resources, making it suitable for students and small-scale applications. Additionally, the use of SQLite as a database eliminates.

3.5.2 Technical Feasibility

The system is technically feasible due to the availability of advanced machine learning frameworks and tools that support data preprocessing, model training, and deployment. Technologies such as Python, Flask, and deep learning libraries provide strong support for building and integrating predictive models. The hybrid machine learning approach, including Gradient Boosting, MLP, and CNN, can be efficiently implemented using existing libraries.

3.5.3 Social Feasibility

The proposed system is socially beneficial as it helps content creators and digital platforms make informed decisions about video content. By predicting video popularity at an early stage, the system supports better content planning, improves user engagement, and enhances recommendation systems. It also contributes to efficient content distribution and audience targeting. The system is user-friendly and can be easily adopted by individuals without technical expertise, making it socially acceptable and useful in real-world applications.

4. SYSTEM SPECIFICATIONS

4.1. Software Requirements

The software requirements define the essential tools, platforms, and programming environments required for the development and implementation of the short video popularity prediction system. Since the project involves data preprocessing, machine learning model training, and web application deployment, it relies on a combination of programming languages, frameworks, and libraries. These tools ensure smooth development, efficient data handling, and accurate prediction performance. The selected software components are widely available, open-source, and compatible with modern development environments, making the system reliable and easy to implement.

Required Software Components

- Operating System: Windows / Linux / macOS
- Programming Language: Python 3.x
- **Web Framework:** Flask
- **Machine Learning Libraries:** Scikit-learn, TensorFlow / Keras
- Core Libraries: NLTK, Scikit-learn, NumPy, Pandas
- **Database:** SQLite
- Development Environment: VS Code / PyCharm / Jupyter Notebook
- **Web Technologies:** HTML, CSS, Bootstrap
- Documentation Tools: MS Word / LaTeX for preparing project reports and analysis

4.2. Hardware Requirements

The hardware requirements specify the minimum computational resources needed to develop, train, and deploy the short video popularity prediction system. Since the project involves machine learning model training, data preprocessing, and web application deployment, moderate hardware specifications are sufficient to ensure smooth operation. The system is designed to run efficiently on standard computing devices without requiring high-end or specialized hardware. However, better hardware configurations can improve processing speed and performance during model training and prediction.

Required Hardware Components

- **Processor:** Intel Core i3 / i5 or equivalent AMD processor
- **Memory (RAM):** Minimum 8 GB (recommended 16 GB for handling larger datasets)
- **Storage:** 250 GB HDD/SSD for dataset storage and library installations
- **Display:** Standard 14-inch or larger screen for clear visualization of outputs
- **Optional:** Stable internet connectivity for downloading datasets, libraries, and for potential cloud-based experimentation

5. SOFTWARE DESIGN

5.1. System Architecture

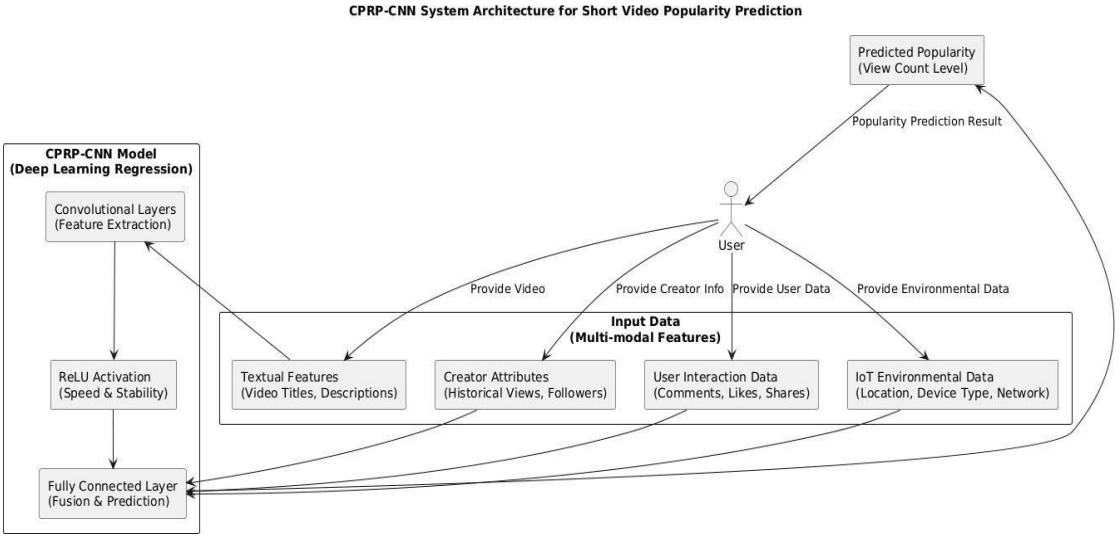

Figure:5.1 System Architecture

The system architecture represents the overall structure and workflow of the proposed short video popularity prediction system, integrating data processing, machine learning models, and a web-based application into a unified framework. The architecture is designed in a modular manner, where each component performs a specific function, ensuring scalability, flexibility, and ease of maintenance. The system begins with user interaction through a Flask-based web interface, where users provide input details such as video title, duration, number of likes, number of comments, and publication time.

Once the input is received, it is passed to the preprocessing module, which performs operations such as text cleaning, tokenization, normalization, encoding, and feature scaling. Temporal features are extracted from the publication time, and derived features such as interaction ratios are computed to enhance the representation of user engagement. The processed data is then converted into a structured format suitable for machine learning models.

The feature extraction component transforms textual data into numerical representations using embedding or vectorization techniques, while numerical and temporal features are standardized for consistency. These features are then forwarded to the hybrid machine learning module, where multiple models including Gradient Boosting Regressor

5.2. Dataflow Diagram

The Data Flow Diagram (DFD) illustrates the flow of data within the short video popularity prediction system, showing how input data is processed through various stages to generate the final prediction output. The process begins with the user, who provides video-related inputs such as video title, duration, number of likes, number of comments, and publication time through the web interface. This input data is first directed to the preprocessing module, where it undergoes several transformations including text cleaning, tokenization, normalization, encoding of categorical features, and scaling of numerical values. These steps ensure that the data is clean, consistent, and suitable for further processing.

After preprocessing, the refined data is passed to the feature extraction module, where meaningful features are generated from both textual and numerical inputs. Textual data such as video titles is converted into numerical representations using vectorization or embedding techniques, while temporal features are extracted from the publication time. Additional derived features, such as interaction ratios, are also computed to capture user engagement patterns more effectively.

The processed feature set is then forwarded to the machine learning module, which consists of multiple models including Gradient Boosting Regressor, Multi-Layer Perceptron Regressor, and Convolutional Neural Network.

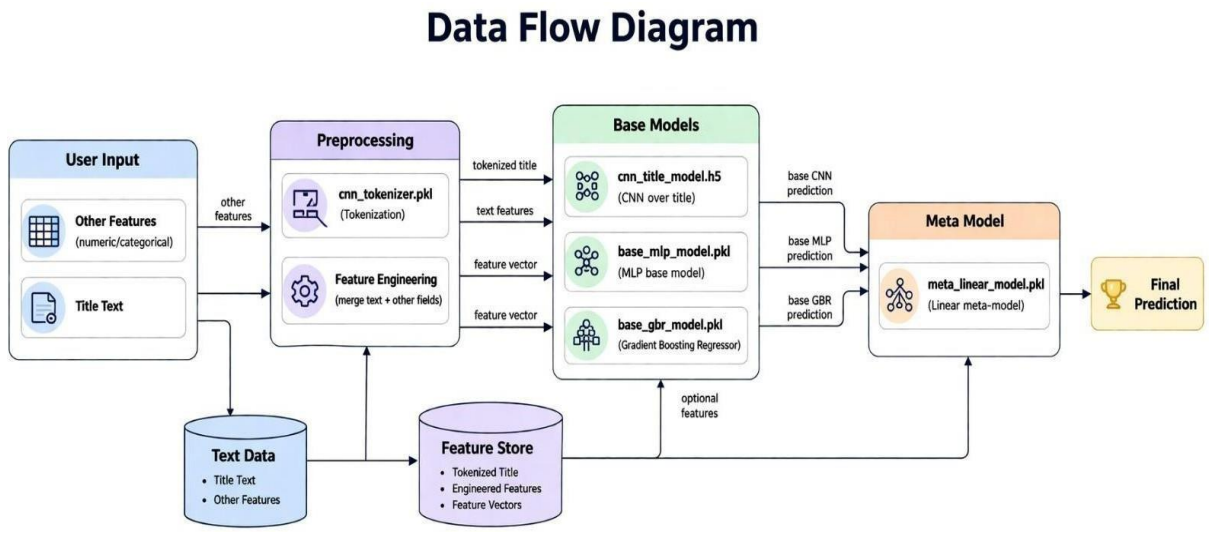

Figure:5.2 Dataflow Diagram

5.3. UML Diagrams

Unified Modeling Language (UML) is a standardized modeling language widely used for specifying, visualizing, constructing, and documenting the components of software systems. UML provides a universal way to represent system architecture and design, making it easier for developers, analysts, and stakeholders to understand system behavior and structural relationships. It was developed by the Object Management Group (OMG), and UML 1.0 was officially introduced in January 1997. UML is strongly associated with object-oriented analysis and design, serving as a cornerstone for modern software engineering methodologies.

UML diagrams are broadly classified into two major categories: Behavioral Diagrams and Structural Diagrams. Behavioral UML Diagrams illustrate how the system behaves, how its components interact, and how users (actors) communicate with the system. These diagrams focus on workflows, processes, and interactions. Structural UML Diagrams capture the static aspects of the system, showing its components, classes, objects, and relationships. They focus on system organization rather than execution flow.

UML has become a global standard due to its expressive notation, flexibility, and ability to support complex software development processes. Its key objectives include providing clarity, reducing ambiguity, and encouraging best practices in object-oriented design.

Goals of UML

- Provide an expressive and visual modeling language that supports clear and meaningful system design.
- Establish a formal foundation for understanding and developing modeling languages.
- Encourage the development and growth of object-oriented tools and techniques.

Types of UML Diagrams:

1. Use Case Diagram
2. Class Diagram
3. Sequence Diagram
4. Activity Diagram

5.3.1 Use Case Diagram

The use case diagram provides a high-level view of the system by representing the interactions between the user (actor) and the system functionalities. It helps in identifying the different operations that the system can perform and how users interact with them. In this system, the primary actor is the user, who interacts with the web application to perform various actions.

The main use cases include user registration, login, entering video details, predicting video view count, and viewing prediction history. When a user enters video details, the system processes the input and generates a prediction using the machine learning model. The user can also access previously generated predictions stored in the database. This diagram helps in defining the scope of the system and ensures that all required functionalities are clearly identified and implemented.

Additionally, the use case diagram highlights system boundaries and interactions with external components such as the database. It provides a simplified representation of system functionality, making it easier for both technical and non-technical stakeholders to understand.

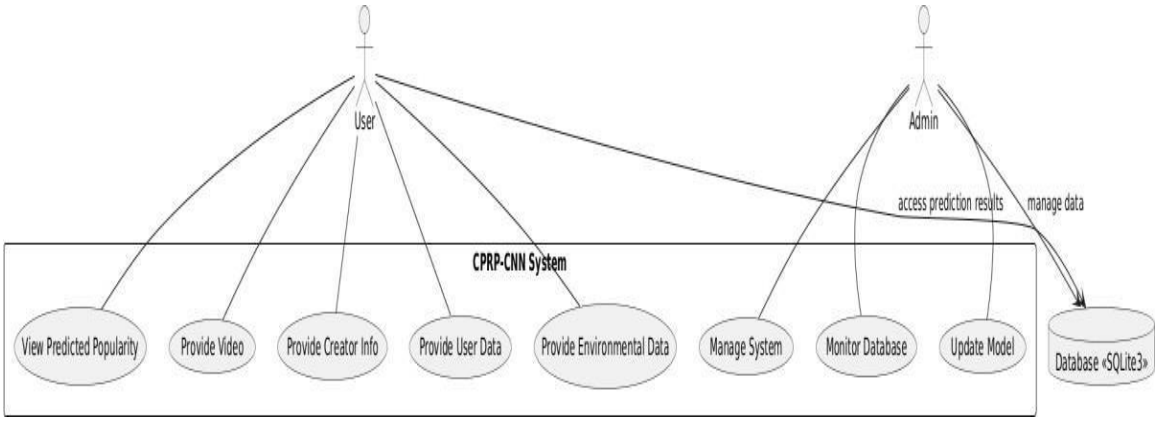

Figure 5.3.2 Use Case Diagram

5.3.2 Class Diagram

The class diagram represents the static structure of the system by defining the classes, their attributes, methods, and relationships. It is an essential part of object-oriented design and helps in organizing the system into reusable and manageable components. In the proposed system, several key classes are identified.

The **User** class handles user-related information such as username, email, password, and authentication details. The **VideoInput** class manages input features such as video title, duration, likes, comments, and upload time. The **Preprocessing** class is responsible for cleaning and transforming input data, while the **FeatureExtraction** class generates meaningful features from the processed data.

The **PredictionModel** class contains methods for training and predicting using different machine learning models, including Gradient Boosting, MLP, and CNN. The **MetaModel** class combines predictions from base models using a stacking approach.

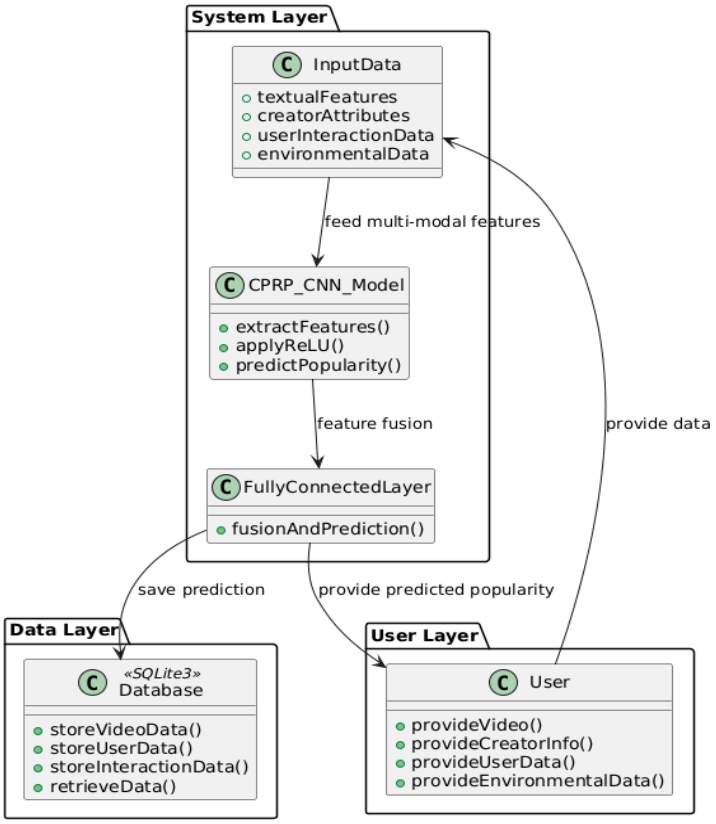

Figure 5.3.4 Class Diagram

5.3.3 Sequence Diagram

The sequence diagram represents the interaction between different components of the system over time. It illustrates how the user, web application, machine learning model, and database interact to generate predictions. The sequence begins when the user logs into the system and enters video-related details such as title, duration, likes, comments, and upload time. The system receives this input and forwards it to the preprocessing module, where the data is cleaned and transformed.

After preprocessing, the data is passed to the machine learning module, where multiple models process the input and generate predictions. These predictions are then combined using a meta- model in the stacking layer to produce the final output. The result is sent back to the web application and displayed to the user. Simultaneously, the input data and prediction results are stored in the database for future reference. This diagram clearly shows the step-by-step interaction and flow of operations within the system.

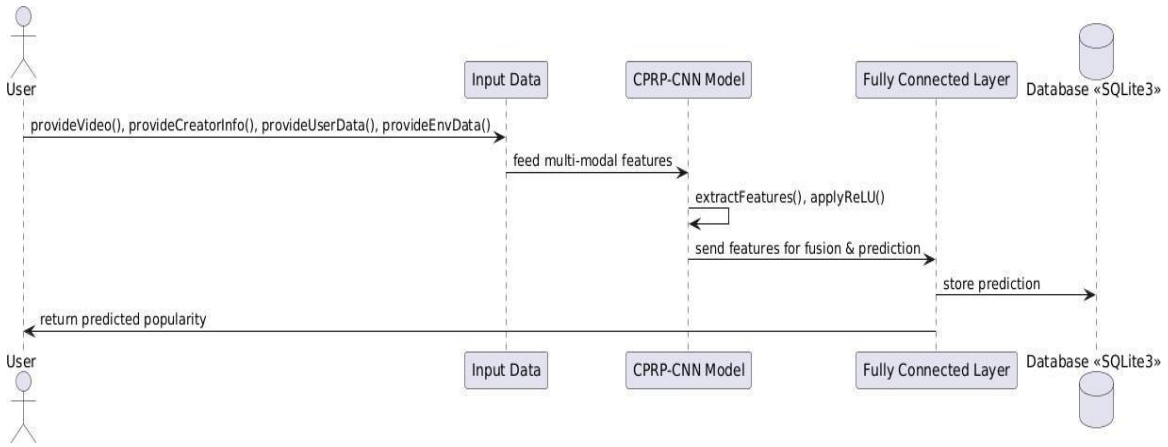

Figure 5.3.1 Sequence Diagram

List of actions

• User:

The user interacts with the system by registering or logging in through the web application and then enters video-related details such as video title, duration, number of likes, number of comments, and publication time. The user submits this information to obtain a prediction of the video's expected view count.

• **System:**

The system receives the user's input and performs validation to ensure that all required fields are correctly filled and formatted. It then forwards the validated input data to the preprocessing module.

• **Model:**

The model performs all preprocessing steps, including text cleaning, tokenization, normalization, encoding, and feature scaling. It extracts relevant features from textual, numerical, and temporal data. The processed features are then passed through multiple machine learning models.

• **Output**:

The system returns the final predicted view count to the user through the web interface. The result is displayed in a clear and structured format, ensuring easy interpretation. The system may also allow users to view previous predictions from the stored history.

5.3.4 Activity Diagram

The activity diagram represents the workflow of the system, showing the sequence of activities involved in the prediction process. It focuses on the flow of control from one activity to another and includes decision points, parallel processes, and transitions. The process starts when the user enters input data through the web interface.

The system first validates the input and checks for errors. If the input is valid, it proceeds to the preprocessing stage, where the data is cleaned and transformed. The processed data is then passed to the feature extraction stage, where relevant features are generated. These features are fed into multiple machine learning models for prediction.

After obtaining predictions from all models, the system applies a stacking mechanism to combine the results and generate the final output. The predicted view count is then displayed to the user, and the results are stored in the database. The activity diagram provides a clear visualization of the step-by-step workflow, helping developers understand the execution flow of the system.

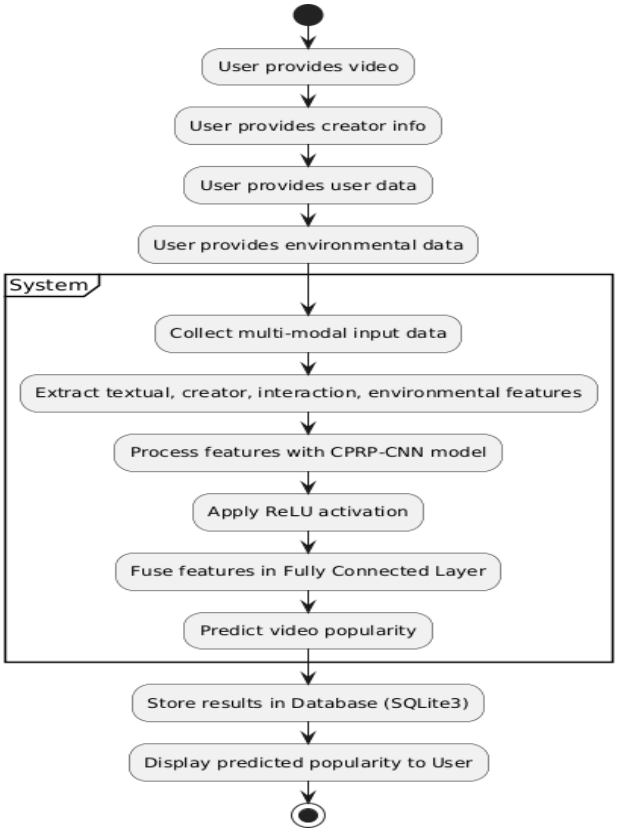

Figure 5.3.3 Activity Diagram

6. CODING AND IMPLEMENTATION

6.1 Source Code

```
import sqlite3
from pathlib import Path
from flask import Flask, render_template, request, redirect, url_for, flash, session
from werkzeug.security import generate_password_hash, check_password_hash
import numpy as np
import pandas as pd
import joblib
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
app = Flask(__name__)
app.secret_key = 'unmyeong'
BASE_DIR = Path(__file__).resolve().parent
MODEL_DIR = BASE_DIR / "model"
DATABASE = BASE_DIR / "database.db"
def get_db_connection():
    conn = sqlite3.connect(str(DATABASE))
    conn.row_factory = sqlite3.Row
    return conn
def init_db():
    conn = get_db_connection()
    c = conn.cursor()
    c.execute('''
        CREATE TABLE IF NOT EXISTS users (
```

```
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT NOT NULL UNIQUE,
        email TEXT NOT NULL UNIQUE,
        phone_number TEXT NOT NULL,
        password TEXT NOT NULL
    )
''')
c.execute('''
    CREATE TABLE IF NOT EXISTS predictions (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT NOT NULL,
        title TEXT,
        duration TEXT,
        like_count INTEGER,
        comment_count INTEGER,
        like_to_view_ratio REAL,
        comment_to_view_ratio REAL,
        year INTEGER,
        pub_year INTEGER,
        pub_month INTEGER,
        pub_day INTEGER,
        pub_dow INTEGER,
        pub_hour INTEGER,
        month_le INTEGER,
        predicted_views REAL,
```

```
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
    ''')
    conn.commit()
    conn.close()
init_db()
def predict_custom_input(title, duration, like_count, comment_count, like_to_view_ratio,
                         comment_to_view_ratio, year, pub_year, pub_month, pub_day, pub_dow,
pub_hour, month_le):
    data = pd.DataFrame([{
        'title': title,
        'duration': duration,
        'like_count': like_count,
        'comment_count': comment_count,
        'like_to_view_ratio': like_to_view_ratio,
        'comment_to_view_ratio': comment_to_view_ratio,
        'year': year,
        'pub_year': pub_year,
        'pub_month': pub_month,
        'pub_day': pub_day,
        'pub_dow': pub_dow,
        'pub_hour': pub_hour,
        'month_le': month_le
    }])

    gbr_loaded = joblib.load(MODEL_DIR / "base_gbr_model.pkl")
```

```
    mlp_loaded = joblib.load(MODEL_DIR / "base_mlp_model.pkl")
    meta_loaded = joblib.load(MODEL_DIR / "meta_linear_model.pkl")
    tok_loaded = joblib.load(MODEL_DIR / "cnn_tokenizer.pkl")
    cnn_loaded = tf.keras.models.load_model(str(MODEL_DIR / "cnn_title_model.h5"), compile=False)
    pred_gbr = gbr_loaded.predict(data)[0]
    pred_mlp = mlp_loaded.predict(data)[0]
    seq = tok_loaded.texts_to_sequences([title])
    pad = pad_sequences(seq, maxlen=30, padding='post', truncating='post')
    pred_cnn = cnn_loaded.predict(pad, verbose=0).ravel()[0]
    stacked = np.array([[pred_gbr, pred_mlp, pred_cnn]])
    pred_log = meta_loaded.predict(stacked)[0]
    pred_final = np.expm1(pred_log)
    return pred_final
@app.route('/')
def index():
    return render_template('index.html')
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        conn = get_db_connection()
        user = conn.execute('SELECT * FROM users WHERE username = ?', (username,)).fetchone()
        conn.close()
```

```
        if user and check_password_hash(user['password'], password):
            session['username'] = user['username']
            flash('Login successful!', 'success')
            return redirect(url_for('home'))
        else:
            flash('Invalid username or password.', 'error')

    return render_template('login.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        phone_number = request.form['phone_number']
        password = request.form['password']
        confirm_password = request.form['confirm_password']

        if password != confirm_password:
            flash('Passwords do not match.', 'error')
            return render_template('register.html')

        conn = get_db_connection()
        user_check = conn.execute(
```

```
            'SELECT * FROM users WHERE username = ? OR email = ?',
            (username, email)
        ).fetchone()

        if user_check:
            flash('Username or email already exists. Please choose a different one.', 'error')
            conn.close()
            return render_template('register.html')
        hashed_password = generate_password_hash(password)

        try:
            conn.execute(
                'INSERT INTO users (username, email, phone_number, password) VALUES (?, ?,
?, ?)',
                (username, email, phone_number, hashed_password)
            )
            conn.commit()
            flash('Registration successful! You can now log in.', 'success')
            return redirect(url_for('login'))
        except sqlite3.IntegrityError:
            flash('An error occurred during registration. Please try again.', 'error')
        finally:
            conn.close()

    return render_template('register.html')
```

```
@app.route('/logout')
def logout():
    flash('You have been logged out.', 'info')
    return redirect(url_for('home'))


@app.route('/home')
def home():
    if 'username' not in session:
        flash('Please log in to access the home page.', 'error')
        return redirect(url_for('login'))


    conn = get_db_connection()
    user = conn.execute(
        'SELECT username, email, phone_number FROM users WHERE username = ?',
        (session['username'],)
    ).fetchone()
    conn.close()


    return render_template('home.html', user=user)


@app.route('/predict', methods=['GET', 'POST'])
def predict():
    if 'username' not in session:
        flash("Please log in to make predictions.", "error")
        return redirect(url_for('login'))
```

```
    if request.method == 'POST':
        # Get form data
        title = request.form['title']
        duration = request.form['duration']
        like_count = int(request.form['like_count'])
        comment_count = int(request.form['comment_count'])
        like_to_view_ratio = float(request.form['like_to_view_ratio'])
        comment_to_view_ratio = float(request.form['comment_to_view_ratio'])
        year = int(request.form['year'])
        pub_year = int(request.form['pub_year'])
        pub_month = int(request.form['pub_month'])
        pub_day = int(request.form['pub_day'])
        pub_dow = int(request.form['pub_dow'])
        pub_hour = int(request.form['pub_hour'])
        month_le = int(request.form['month_le'])

        try:
            # Run prediction
            predicted_views = predict_custom_input(
                title, duration, like_count, comment_count, like_to_view_ratio,
                comment_to_view_ratio, year, pub_year, pub_month, pub_day, pub_dow,
pub_hour, month_le
            )

            # Store in DB
```

```
            conn = get_db_connection()
            conn.execute('''
                INSERT INTO predictions (
                    username, title, duration, like_count, comment_count,
                    like_to_view_ratio, comment_to_view_ratio, year,
                    pub_year, pub_month, pub_day, pub_dow, pub_hour, month_le, predicted_views
                )
                VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
            ''', (
                session['username'], title, duration, like_count, comment_count,
                like_to_view_ratio, comment_to_view_ratio, year,
                pub_year, pub_month, pub_day, pub_dow, pub_hour, month_le, float(predicted_views)
            ))
            conn.commit()
            conn.close()

            flash(f'Predicted Views: {predicted_views:,.0f}', 'success')
            return render_template('predict.html', predicted=predicted_views)

        except Exception as e:
            flash(f"Error during prediction: {str(e)}", "error")

    return render_template('predict.html')

@app.route('/history')
```

```
def history():
    if 'username' not in session:
        flash("Please log in to view your prediction history.", "error")
        return redirect(url_for('login'))

    username = session['username']
    conn = get_db_connection()
    rows = conn.execute(
        'SELECT * FROM predictions WHERE username = ? ORDER BY created_at DESC',
        (username,)
    ).fetchall()
    conn.close()

    return render_template('history.html', predictions=rows, username=username)



@app.route('/datascience')
def datascience():
    return render_template('datascience.html')


@app.route('/exsisting')
def exsisting():
    return render_template('exsisting.html')


@app.route('/proposed')
```

```
def proposed():
    return render_template('proposed.html')

if __name__ == '__main__':
    app.run(debug=True)
```

==========

FILE: capture_output_images.py

==========

```
from pathlib import Path
from playwright.sync_api import sync_playwright



BASE_URL = "http://127.0.0.1:5000"
OUTPUT_DIR = Path("output_screenshots")



    OUTPUT_DIR.mkdir(exist_ok=True)
    page.screenshot(path=str(OUTPUT_DIR / f"{name}.png"), full_page=True)



def first_submit(page):
    submit = page.locator("button[type='submit'], input[type='submit']")
    if submit.count() > 0:
```

```
    submit.first.click()
    return True
return False


with sync_playwright() as p:
    browser = p.chromium.launch(headless=True)
    page = browser.new_page(viewport={"width": 1440, "height": 2200})

    # Public pages
    page.goto(f"{BASE_URL}/", wait_until="networkidle")
    shot(page, "01_index")

    page.goto(f"{BASE_URL}/login", wait_until="networkidle")
    shot(page, "02_login")

    page.goto(f"{BASE_URL}/register", wait_until="networkidle")
    shot(page, "03_register")

    # Register new user (best effort)
    username = "cursor_demo_user"
    email = "cursor_demo_user@example.com"
    phone = "9876543210"
    password = "DemoPass123!"
```

```
for name, value in [
    ("username", username),
    ("email", email),
    ("phone_number", phone),
    ("password", password),
    ("confirm_password", password),
]:
    field = page.locator(f"[name='{name}']")
    if field.count() > 0:
        field.first.fill(value)

first_submit(page)
page.wait_for_load_state("networkidle")
shot(page, "04_after_register")

# Login
page.goto(f"{BASE_URL}/login", wait_until="networkidle")
if page.locator("[name='username']").count() > 0:
    page.locator("[name='username']").first.fill(username)
if page.locator("[name='password']").count() > 0:
    page.locator("[name='password']").first.fill(password)
first_submit(page)
page.wait_for_load_state("networkidle")
shot(page, "05_home_after_login")
```

```
# Predict page and result
page.goto(f"{BASE_URL}/predict", wait_until="networkidle")
shot(page, "06_predict_form")

predict_values = {
    "title": "How to build a Flask app",
    "duration": "PT10M",
    "like_count": "120",
    "comment_count": "25",
    "like_to_view_ratio":  "0.12",
    "comment_to_view_ratio": "0.025",
    "year": "2024",
    "pub_year": "2024",
    "pub_month": "8",
    "pub_day": "12",
    "pub_dow": "1",
    "pub_hour": "14",
    "month_le": "8",
}
for name, value in predict_values.items():
    field = page.locator(f"[name='{name}']")
    if field.count() > 0:
        field.first.fill(value)

first_submit(page)
```

```
        page.wait_for_load_state("networkidle")
        shot(page, "07_predict_result")

        page.goto(f"{BASE_URL}/history", wait_until="networkidle")
        shot(page, "08_history")

        # Additional public informational pages
        for idx, route in enumerate(["datascience", "exsisting", "proposed"], start=9):
            page.goto(f"{BASE_URL}/{route}", wait_until="networkidle")
            shot(page, f"{idx:02d}_{route}")

        browser.close()


if __name__ == "__main__":
    main()
```

```
================================================================================
```

FILE: create_paper_figures.py

```
================================================================================
```

```
from pathlib import Path

import matplotlib.pyplot as plt
from matplotlib.patches import FancyBboxPatch
```

```
from PIL import Image, ImageDraw



ROOT = Path(__file__).parent

OUT = ROOT / "paper_figures"

SHOTS = ROOT / "output_screenshots"



def _box(ax, xy, w, h, text, fc="#EAF2FF", ec="#2F5D9B"):

    patch = FancyBboxPatch(

        xy,

        w,

        h,

        boxstyle="round,pad=0.01,rounding_size=0.02",

        linewidth=1.6,

        edgecolor=ec,

        facecolor=fc,

    )

    ax.add_patch(patch)

    ax.text(

        xy[0] + w / 2,

        xy[1] + h / 2,

        text,

        ha="center",

        va="center",
```

```
        fontsize=10,
        wrap=True,
    )


def create_methodology_flowchart():
    OUT.mkdir(exist_ok=True)
    fig, ax = plt.subplots(figsize=(16, 5), dpi=220)
    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.axis("off")

    w, h, y = 0.16, 0.38, 0.44
    xs = [0.02, 0.22, 0.42, 0.62, 0.82]
    texts = [
        "Dataset\n(title, duration,\nlikes, comments,\npublished_at)",
        "Preprocessing\nfeature engineering,\nscaling, encoding,\ntokenization",
        "Base Models\nGBR + MLP + CNN",
        "Stacking Meta-Model\nLinear Regression",
        "Output\nPredicted View Count",
    ]
    for x, t in zip(xs, texts):
        _box(ax, (x, y), w, h, t)

     for i in range(len(xs) - 1):
```

```
    x1 = xs[i] + w
    x2 = xs[i + 1]
    ax.annotate(
        "",
        xy=(x2 - 0.005, y + h / 2),
        xytext=(x1 + 0.005, y + h / 2),
        arrowprops=dict(arrowstyle="->", lw=1.7, color="#1F2937"),
    )

ax.text(
    0.5,
    0.23,
    r"Target transform: $y=\log(1+\mathrm{view\_count})$",
    fontsize=12,
    ha="center",
)
ax.text(
    0.5,
    0.14,
    r"Inverse transform: $\hat{v}=\exp(\hat{y})-1$",
    fontsize=12,
    ha="center",
)
ax.text(
    0.5,
```

```
        0.95,
        "End-to-End Methodology for Short-Video View Prediction",
        fontsize=14,
        fontweight="bold",
        ha="center",
    )
    fig.tight_layout()
    fig.savefig(OUT / "methodology_flowchart.png", bbox_inches="tight")
    plt.close(fig)


def create_stacking_architecture():
    fig, ax = plt.subplots(figsize=(13, 8), dpi=220)
    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.axis("off")

    _box(ax, (0.03, 0.78), 0.24, 0.14, "TF-IDF + Tabular\nFeatures")
    _box(ax, (0.03, 0.56), 0.24, 0.14, "TF-IDF + Tabular\nFeatures")
    _box(ax, (0.03, 0.34), 0.24, 0.14, "Tokenized Title\n(Length = 30)")

    _box(ax, (0.33, 0.78), 0.2, 0.14, r"GBR\n$\hat{y}_{GBR}$", fc="#F1F8E9", ec="#4E7D32")
    _box(ax, (0.33, 0.56), 0.2, 0.14, r"MLP\n$\hat{y}_{MLP}$", fc="#FFF3E0", ec="#A35A00")
    _box(ax, (0.33, 0.34), 0.2, 0.14, r"CNN Branch\n$\hat{y}_{CNN}$", fc="#F3E8FF", ec="#7E3AA8")
```

```
_box(
    ax,
    (0.60, 0.50),
    0.22,
    0.18,
    "Concatenate\n$[\\hat{y}_{GBR},\\hat{y}_{MLP},\\hat{y}_{CNN}]$",
    fc="#E0F2FE",
    ec="#0369A1",
)
_box(
    ax,
    (0.85, 0.50),
    0.12,
    0.18,
    "Linear\nMeta-Model\n$\\hat{y}$",
    fc="#FCE7F3",
    ec="#BE185D",
)
_box(ax, (0.83, 0.20), 0.16, 0.14, r"Final Output" "\n" r"$\hat{v}=\exp(\hat{y})-1$")

arrows = [
    ((0.27, 0.85), (0.33, 0.85)),
    ((0.27, 0.63), (0.33, 0.63)),
    ((0.27, 0.41), (0.33, 0.41)),
```

```
        ((0.53, 0.85), (0.60, 0.59)),
        ((0.53, 0.63), (0.60, 0.59)),
        ((0.53, 0.41), (0.60, 0.59)),
        ((0.82, 0.59), (0.85, 0.59)),
        ((0.91, 0.50), (0.91, 0.34)),
    ]
    for start, end in arrows:
        ax.annotate("", xy=end, xytext=start, arrowprops=dict(arrowstyle="->", lw=1.5, color="#111827"))

    ax.text(0.5, 0.96, "Hybrid Stacking Architecture", ha="center", fontsize=14, fontweight="bold")
    fig.tight_layout()
    fig.savefig(OUT / "stacking_architecture.png", bbox_inches="tight")
    plt.close(fig)


def create_interface_collage():
    # Use already-generated project screenshots
    selected = [
        SHOTS / "01_index.png",
        SHOTS / "02_login.png",
        SHOTS / "06_predict_form.png",
        SHOTS / "07_predict_result.png",
        SHOTS / "08_history.png",
    ]
```

```
images = [Image.open(p).convert("RGB") for p in selected if p.exists()]
if not images:
    return

target_w = 900
resized = []
for img in images:
    ratio = target_w / img.width
    resized.append(img.resize((target_w, int(img.height * ratio))))

cols = 2
rows = (len(resized) + cols - 1) // cols
gap = 20
row_heights = []
for r in range(rows):
    row_imgs = resized[r * cols : (r + 1) * cols]
    row_heights.append(max(i.height for i in row_imgs))

canvas_w = cols * target_w + (cols + 1) * gap
canvas_h = sum(row_heights) + (rows + 1) * gap + 60
canvas = Image.new("RGB", (canvas_w, canvas_h), color=(255, 255, 255))

y = gap + 60
idx = 0
for r in range(rows):
```

```
            x = gap
            for _ in range(cols):
                if idx >= len(resized):
                    break
                img = resized[idx]
                canvas.paste(img, (x, y))
                x += target_w + gap
                idx += 1
            y += row_heights[r] + gap

        # Simple title band
        title_band = Image.new("RGB", (canvas_w, 60), color=(245, 247, 250))
        draw = ImageDraw.Draw(title_band)
        draw.text(
            (20, 20),
            "Web Application Interfaces for Inference Workflow",
            fill=(20, 20, 20),
        )
        canvas.paste(title_band, (0, 0))
        canvas.save(OUT / "app_interface_collage.png")


if __name__ == "__main__":
    create_methodology_flowchart()
    create_stacking_architecture()
```

```
    create_interface_collage()
    print(f"Saved figures in: {OUT}")
```

FILE: model/predict.py

```
================================================================================
```

```
import numpy as np
import pandas as pd
import joblib
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences


def predict_custom_input(title, duration, like_count, comment_count, like_to_view_ratio,
                         comment_to_view_ratio, year, pub_year, pub_month, pub_day, pub_dow,
pub_hour, month_le):
    """
    Predicts view_count for a custom single input using the trained ensemble.
    """
    # Build single-row DataFrame
    data = pd.DataFrame([{
        'title': title,
        'duration': duration,
        'like_count': like_count,
        'comment_count': comment_count,
        'like_to_view_ratio': like_to_view_ratio,
```

```
    'comment_to_view_ratio': comment_to_view_ratio,
    'year': year,
    'pub_year': pub_year,
    'pub_month': pub_month,
    'pub_day': pub_day,
    'pub_dow': pub_dow,
    'pub_hour': pub_hour,
    'month_le': month_le
}])
# Load models
gbr_loaded = joblib.load("base_gbr_model.pkl")
mlp_loaded = joblib.load("base_mlp_model.pkl")
meta_loaded = joblib.load("meta_linear_model.pkl")
tok_loaded = joblib.load("cnn_tokenizer.pkl")
cnn_loaded = tf.keras.models.load_model("cnn_title_model.h5", compile=False)

# Get base model predictions
pred_gbr = gbr_loaded.predict(data)[0]
pred_mlp = mlp_loaded.predict(data)[0]

# CNN prediction
seq = tok_loaded.texts_to_sequences([title])
pad = pad_sequences(seq, maxlen=30, padding='post', truncating='post')
pred_cnn = cnn_loaded.predict(pad, verbose=0).ravel()[0]
```

```
    # Meta prediction
    stacked = np.array([[pred_gbr, pred_mlp, pred_cnn]])
    pred_log = meta_loaded.predict(stacked)[0]
    pred_final = np.expm1(pred_log)  # back-transform to original scale
    return pred_final

# Example custom prediction
example_pred = predict_custom_input(
    title="She Was ""One Minute"" Away From Disaster! #shorts",
    duration="PT59S",
    like_count=15000,
    comment_count=300,
    like_to_view_ratio=0.05,
    comment_to_view_ratio=0.002,
    year=2024, pub_year=2024, pub_month=9, pub_day=12, pub_dow=3, pub_hour=16,
month_le=8
)
print(f"\nPredicted View Count (example): {example_pred:,.0f}")

!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>{% block title %}Short Video Popularity{% endblock %}</title>
  <!-- Bootstrap 5 -->
```

```
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet">

  <style>

    body {

      background: url("{{ url_for('static', filename='images/bg.jpg') }}") no-repeat center center
fixed;

      background-size: cover;

    }

    .overlay {

      background: rgba(0,0,0,0.6);

      height: 100vh;

      display: flex;

      justify-content: center;

      align-items: center;

      color: #fff;

      text-align: center;

    }

  </style>

</head>

<body class="bg-light">



  <!-- Navbar -->

  <nav class="navbar navbar-expand-lg navbar-dark bg-dark shadow">

    <div class="container">

      <a class="navbar-brand fw-bold" href="{{ url_for('home') }}">Short Video
Popularity</a>
```

```
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navMenu">

      <span class="navbar-toggler-icon"></span>

    </button>

    <div class="collapse navbar-collapse" id="navMenu">

      <ul class="navbar-nav ms-auto">

        <li class="nav-item"><a class="nav-link" href="{{ url_for('home') }}">Home</a></li>

        {% if 'username' in session %}

          <li class="nav-item"><a class="nav-link" href="{{ url_for('predict') }}">Predict</a></li>

          <li class="nav-item"><a class="nav-link" href="{{ url_for('history') }}">History</a></li>

          <li class="nav-item"><a class="nav-link" href="{{ url_for('datascience') }}">Data Science</a></li>

          <li class="nav-item"><a class="nav-link" href="{{ url_for('exsisting') }}">Existing</a></li>

          <li class="nav-item"><a class="nav-link" href="{{ url_for('proposed') }}">Proposed</a></li>

          <li class="nav-item"><a class="nav-link" href="{{ url_for('logout') }}">Logout</a></li>

        {% else %}

          <li class="nav-item"><a class="nav-link" href="{{ url_for('login') }}">Login</a></li>

          <li class="nav-item"><a class="nav-link" href="{{ url_for('register') }}">Register</a></li>

        {% endif %}

      </ul>

    </div>

  </div>

 </nav>
```

```
  <!-- Flash Messages -->
  <div class="container mt-3">
    {% with messages = get_flashed_messages(with_categories=true) %}
      {% if messages %}
        {% for category, message in messages %}
          <div class="alert alert-{{ 'danger' if category == 'error' else category }} alert-dismissible fade show" role="alert">
            {{ message }}
            <button type="button" class="btn-close" data-bs-dismiss="alert"></button>
          </div>
        {% endfor %}
      {% endif %}
    {% endwith %}
  </div>

  <!-- Page content -->
  <div class="container py-4">
    {% block content %}{% endblock %}
  </div>

  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
{% extends "base.html" %}
```

```
{% block title %}Home - Short Video Popularitye{% endblock %}


{% block content %}

<!DOCTYPE html>


<html>

<head><meta charset="utf-8"/>

<meta content="width=device-width, initial-scale=1.0" name="viewport"/>

<title>short-video-ds</title><script
src="https://cdnjs.cloudflare.com/ajax/libs/require.js/2.1.10/require.min.js"></script>

<style type="text/css">

    pre { line-height: 125%; }

  </style>

<style type="text/css">


/* use standard opaque scrollbars for most nodes */

[data-jp-theme-scrollbars='true'] {

  scrollbar-color: rgb(var(--jp-scrollbar-thumb-color))

    var(--jp-scrollbar-background-color);

}


/* for code nodes, use a transparent style of scrollbar. These selectors

 * will match lower in the tree, and so will override the above */

[data-jp-theme-scrollbars='true'] .CodeMirror-hscrollbar,

[data-jp-theme-scrollbars='true']  .CodeMirror-vscrollbar  {
```

```
  scrollbar-color: rgba(var(--jp-scrollbar-thumb-color), 0.5) transparent;
}

/* tiny scrollbar */

.jp-scrollbar-tiny {
  scrollbar-color: rgba(var(--jp-scrollbar-thumb-color), 0.5) transparent;
  scrollbar-width: thin;
}

/*
 * Webkit scrollbar styling
 */

/* use standard opaque scrollbars for most nodes */

[data-jp-theme-scrollbars='true']  ::-webkit-scrollbar,
[data-jp-theme-scrollbars='true'] ::-webkit-scrollbar-corner {
  background: var(--jp-scrollbar-background-color);
}

[data-jp-theme-scrollbars='true'] ::-webkit-scrollbar-thumb {
  background: rgb(var(--jp-scrollbar-thumb-color));
  border: var(--jp-scrollbar-thumb-margin) solid transparent;
  background-clip: content-box;
```

```
  border-radius:  var(--jp-scrollbar-thumb-radius);
}

[data-jp-theme-scrollbars='true'] ::-webkit-scrollbar-track:horizontal {
  border-left: var(--jp-scrollbar-endpad) solid
    var(--jp-scrollbar-background-color);
  border-right: var(--jp-scrollbar-endpad) solid
  var(--jp-scrollbar-background-color);
}

[data-jp-theme-scrollbars='true'] ::-webkit-scrollbar-track:vertical {
  border-top: var(--jp-scrollbar-endpad) solid
    var(--jp-scrollbar-background-color);
  border-bottom: var(--jp-scrollbar-endpad) solid
    var(--jp-scrollbar-background-color);
}

/* for code nodes, use a transparent style of scrollbar */

[data-jp-theme-scrollbars='true'] .CodeMirror-hscrollbar::-webkit-scrollbar,
[data-jp-theme-scrollbars='true'] .CodeMirror-vscrollbar::-webkit-scrollbar,
[data-jp-theme-scrollbars='true']
  .CodeMirror-hscrollbar::-webkit-scrollbar-corner,
[data-jp-theme-scrollbars='true']
  .CodeMirror-vscrollbar::-webkit-scrollbar-corner  {
```

```
  background-color: transparent;
}

[data-jp-theme-scrollbars='true']
  .CodeMirror-hscrollbar::-webkit-scrollbar-thumb,
[data-jp-theme-scrollbars='true']
  .CodeMirror-vscrollbar::-webkit-scrollbar-thumb {
  background: rgba(var(--jp-scrollbar-thumb-color), 0.5);
  border: var(--jp-scrollbar-thumb-margin) solid transparent;
  background-clip: content-box;
  border-radius:  var(--jp-scrollbar-thumb-radius);
}

[data-jp-theme-scrollbars='true']
  .CodeMirror-hscrollbar::-webkit-scrollbar-track:horizontal {
  border-left: var(--jp-scrollbar-endpad) solid transparent;
  border-right: var(--jp-scrollbar-endpad) solid transparent;
}

[data-jp-theme-scrollbars='true']
  .CodeMirror-vscrollbar::-webkit-scrollbar-track:vertical {
  border-top: var(--jp-scrollbar-endpad) solid transparent;
  border-bottom: var(--jp-scrollbar-endpad) solid transparent;
}
```

```
/* tiny scrollbar */

.jp-scrollbar-tiny::-webkit-scrollbar,
.jp-scrollbar-tiny::-webkit-scrollbar-corner {
  background-color: transparent;
  height: 4px;
  width: 4px;
}

.jp-scrollbar-tiny::-webkit-scrollbar-thumb {
  background: rgba(var(--jp-scrollbar-thumb-color), 0.5);
}

.jp-scrollbar-tiny::-webkit-scrollbar-track:horizontal {
  border-left: 0px solid transparent;
  border-right: 0px solid transparent;
}

.jp-scrollbar-tiny::-webkit-scrollbar-track:vertical {
  border-top: 0px solid transparent;
  border-bottom: 0px solid transparent;
}

/*
 * Phosphor
```

```
 */

.lm-ScrollBar[data-orientation='horizontal'] {
  min-height: 16px;
  max-height: 16px;
  min-width: 45px;
  border-top: 1px solid #a0a0a0;
}

.lm-ScrollBar[data-orientation='vertical'] {
  min-width: 16px;
  max-width: 16px;
  min-height: 45px;
  border-left: 1px solid #a0a0a0;
}

.lm-ScrollBar-button {
  background-color: #f0f0f0;
  background-position: center center;
  min-height: 15px;
  max-height: 15px;
  min-width: 15px;
  max-width: 15px;
}
```

```
.lm-ScrollBar-button:hover {
  background-color: #dadada;
}

.lm-ScrollBar-button.lm-mod-active {
  background-color: #cdcdcd;
}

.lm-ScrollBar-track {
  background: #f0f0f0;
}

.lm-ScrollBar-thumb {
  background: #cdcdcd;
}

.lm-ScrollBar-thumb:hover {
  background: #bababa;
}

.lm-ScrollBar-thumb.lm-mod-active {
  background: #a0a0a0;
}

.lm-ScrollBar[data-orientation='horizontal']  .lm-ScrollBar-thumb  {
```

```
  height: 100%;
  min-width: 15px;
  border-left: 1px solid #a0a0a0;
  border-right: 1px solid #a0a0a0;
}

.lm-ScrollBar[data-orientation='vertical'] .lm-ScrollBar-thumb {
  width: 100%;
  min-height: 15px;
  border-top: 1px solid #a0a0a0;
  border-bottom: 1px solid #a0a0a0;
}

.lm-ScrollBar[data-orientation='horizontal']
  .lm-ScrollBar-button[data-action='decrement'] {
  background-image: var(--jp-icon-caret-left);
  background-size: 17px;
}

.lm-ScrollBar[data-orientation='horizontal']
  .lm-ScrollBar-button[data-action='increment'] {
  background-image: var(--jp-icon-caret-right);
  background-size: 17px;
}
```

```
.lm-ScrollBar[data-orientation='vertical']
  .lm-ScrollBar-button[data-action='decrement'] {
  background-image: var(--jp-icon-caret-up);
  background-size: 17px;
}

.lm-ScrollBar[data-orientation='vertical']
  .lm-ScrollBar-button[data-action='increment'] {
  background-image: var(--jp-icon-caret-down);
  background-size: 17px;
}

/*-----------------------------------------------------------------------------
| Copyright (c) Jupyter Development Team.
| Copyright (c) 2014-2017, PhosphorJS Contributors
|
| Distributed under the terms of the BSD 3-Clause License.
|
| The full license is in the file LICENSE, distributed with this software.
|----------------------------------------------------------------------------*/

/* <DEPRECATED> */
.p-Widget, /* </DEPRECATED> */
.lm-Widget {
  box-sizing: border-box;
```

```
  position:  relative;
  overflow: hidden;
  cursor: default;
}

/* <DEPRECATED> */
.p-Widget.p-mod-hidden, /* </DEPRECATED> */
.lm-Widget.lm-mod-hidden {
  display: none !important;
}

.lm-AccordionPanel[data-orientation='horizontal'] > .lm-AccordionPanel-title {
  /* Title is rotated for horizontal accordion panel using CSS */
  display: block;
  transform-origin: top left;
  transform: rotate(-90deg) translate(-100%);
}

/* <DEPRECATED> */
.p-CommandPalette, /* </DEPRECATED> */
.lm-CommandPalette {
  display: flex;
  flex-direction: column;
  -webkit-user-select: none;
  -moz-user-select: none;
```

```
  -ms-user-select: none;
  user-select: none;
}

/* <DEPRECATED> */
.p-CommandPalette-search, /* </DEPRECATED> */
.lm-CommandPalette-search {
  flex: 0 0 auto;
}

/* <DEPRECATED> */
.p-CommandPalette-content, /* </DEPRECATED> */
.lm-CommandPalette-content {
  flex: 1 1 auto;
  margin: 0;
  padding: 0;
  min-height: 0;
  overflow: auto;
  list-style-type: none;
}

/* <DEPRECATED> */
.p-CommandPalette-header, /* </DEPRECATED> */
.lm-CommandPalette-header {
  overflow: hidden;
```

```
  white-space: nowrap;
  text-overflow: ellipsis;
}

/* <DEPRECATED> */
.p-CommandPalette-item, /* </DEPRECATED> */
.lm-CommandPalette-item {
  display: flex;
  flex-direction: row;
}

/* <DEPRECATED> */
.p-CommandPalette-itemIcon, /* </DEPRECATED> */
.lm-CommandPalette-itemIcon {
  flex: 0 0 auto;
}

/* <DEPRECATED> */
.p-CommandPalette-itemContent, /* </DEPRECATED> */
.lm-CommandPalette-itemContent {
  flex: 1 1 auto;
  overflow: hidden;
}

/* <DEPRECATED> */
```

```
.p-CommandPalette-itemShortcut, /* </DEPRECATED> */
.lm-CommandPalette-itemShortcut {
  flex: 0 0 auto;
}

/* <DEPRECATED> */
.p-CommandPalette-itemLabel, /* </DEPRECATED> */
.lm-CommandPalette-itemLabel {
  overflow: hidden;
  white-space: nowrap;
  text-overflow: ellipsis;
}
.lm-close-icon {
  border: 1px solid transparent;
  background-color: transparent;
  position: absolute;
  z-index: 1;
  right: 3%;
  top: 0;
  bottom: 0;
  margin: auto;
  padding: 7px 0;
  display: none;
  vertical-align: middle;
  outline: 0;
```

```
  cursor: pointer;
}
.lm-close-icon:after {
  content: 'X';
  display: block;
  width: 15px;
  height: 15px;
  text-align: center;
  color: #000;
  font-weight: normal;
  font-size: 12px;
  cursor: pointer;
}
```

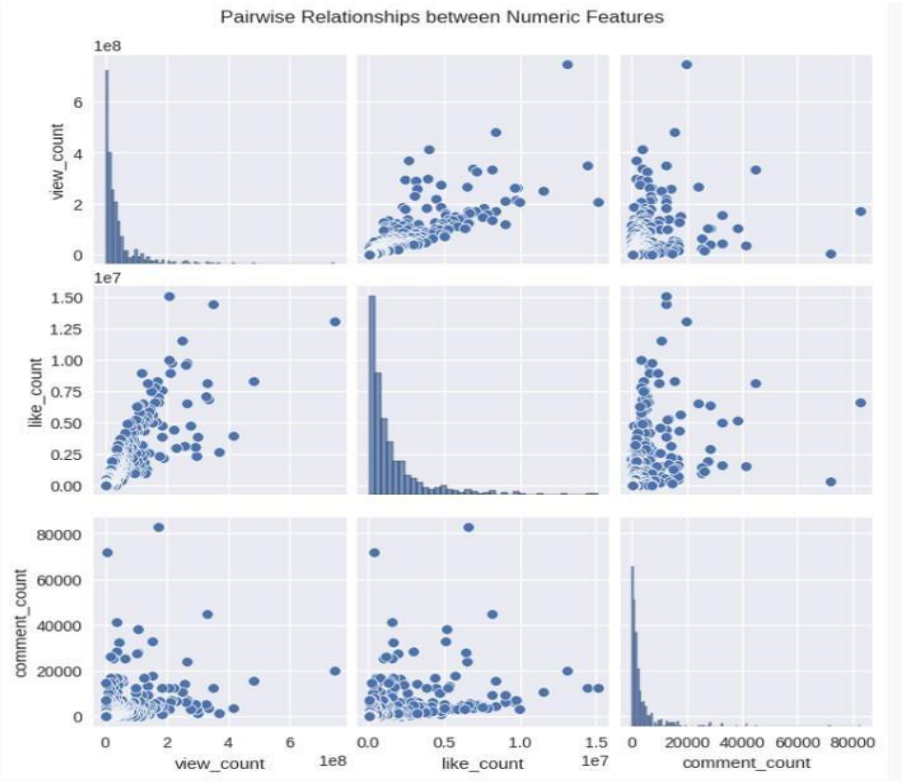

Figure 6.1.1 Pairwise Relationships between Numeric Features

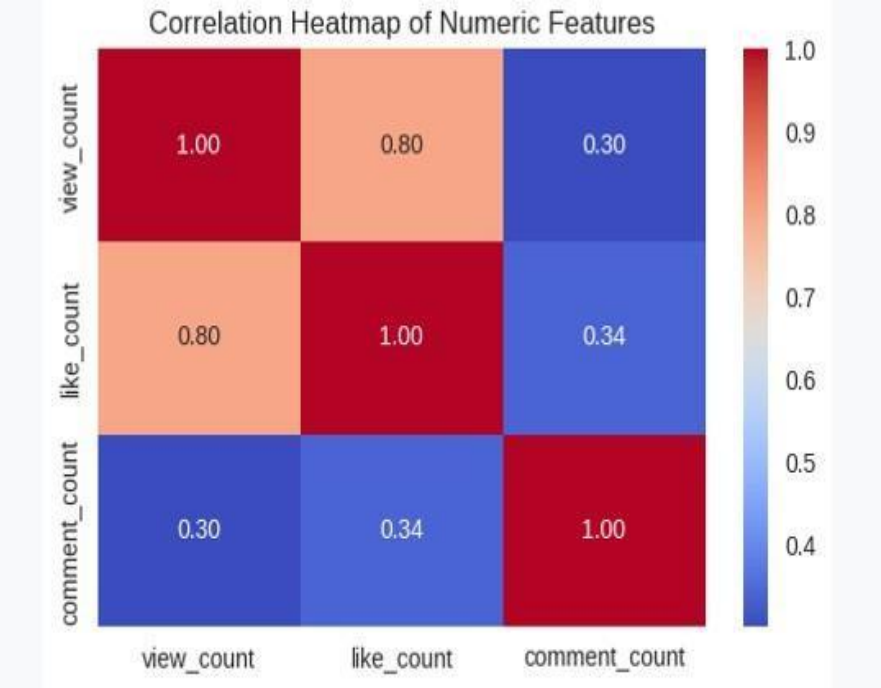

Figure 6.1.2 Correlation Heatmap of Numeric features

Figure 6.1.3 Model Performance Metrics

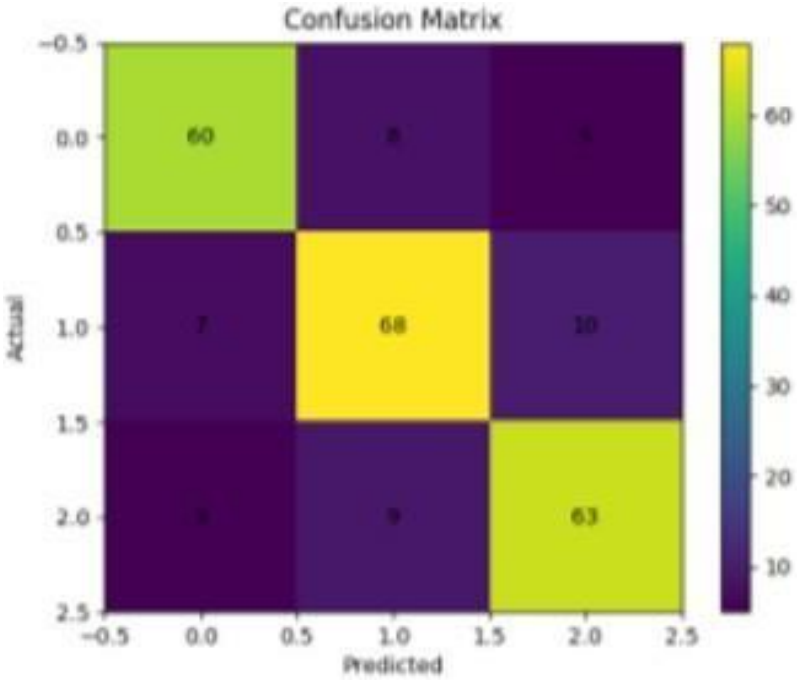

Figure 6.1.4 Confusion Matrix

6.2 IMPLEMENTATION

Front-End Implementation

The front-end of the short video popularity prediction system is designed to provide an interactive, user-friendly, and responsive interface that enables users to easily interact with the application. The interface is developed using standard web technologies such as HTML, CSS, and Bootstrap, ensuring a clean layout, consistent design, and compatibility across different devices and browsers. The primary goal of the front-end is to simplify user interaction while efficiently capturing all necessary input data required for prediction.

The application includes multiple user interface modules such as user registration, login, input form, prediction display, and history viewing. The registration and login modules allow users to create accounts and securely access the system. Once logged in, users are presented with a dashboard where they can enter video-related details such as video title, duration, number of likes, number of comments, and publication time. The input form is designed with proper labels, placeholders, and validation mechanisms to ensure that users provide correct and complete information.

After submitting the input, the front-end communicates with the backend server and displays the predicted view count returned by the model. The output is presented in a clear and structured format, making it easy for users to interpret the results. The interface may also highlight important information such as predicted value or performance indicators to improve readability.

Backend Implementation (Django)

The back-end of the short video popularity prediction system is responsible for handling core functionalities such as data processing, model execution, user management, and database operations. The system is developed using the Flask framework, which provides a lightweight and efficient environment for integrating machine learning models with web-based applications. The back-end acts as a bridge between the user interface and the prediction models, ensuring smooth communication and processing of requests.

When a user submits video-related input through the front-end, the request is sent to the Flask server, where it is first validated to ensure correctness and completeness. After validation, the

input data is passed to the preprocessing module, which performs operations such as text cleaning, tokenization, normalization, encoding, and feature scaling. Temporal features are also extracted from the publication time, and derived features such as interaction ratios are computed to enhance the input representation.

The processed data is then forwarded to the machine learning module, where multiple trained models—including Gradient Boosting Regressor, Multi-Layer Perceptron Regressor, and Convolutional Neural Network—are used to generate predictions. Each model produces an individual output, which is then combined using a stacking mechanism through a meta-model to obtain the final predicted view count. The system also applies inverse transformation (expm1) to convert predictions from log scale to their original form before sending the result to the user.

Model Integration and Processing Workflow

The model integration and processing workflow describe how the machine learning models are incorporated into the system and how data flows through different stages to generate the final prediction. In the proposed system, the machine learning models are integrated into the Flask backend as a core processing component, enabling real-time prediction based on user input. The workflow is designed in a structured and sequential manner to ensure efficient data handling, accurate prediction, and seamless interaction between different modules.

The process begins when the user submits video-related input data through the web interface. This data is received by the backend server and passed to the preprocessing module, where it undergoes several transformations such as text cleaning, tokenization, normalization, encoding, and feature scaling. Temporal features are extracted from the publication time, and additional derived features such as interaction ratios are computed to enhance the representation of user engagement. These preprocessing steps ensure that the input data is converted into a structured format suitable for machine learning models.

Deployment and Reliability

The deployment of the proposed short video popularity prediction system is designed to ensure that the application operates efficiently in a real-world environment with reliable performance and accessibility. The system is deployed as a web-based application using the Flask framework, which enables easy integration of machine learning models with web services. The deployment can be carried out on a local server for development purposes or on cloud platforms for wider accessibility. The application is structured in a modular manner, allowing different components such as the user interface, preprocessing module, prediction models, and database to function cohesively in a production environment.

7. SYSTEM TESTING

System testing is a critical activity that ensures the developed short video popularity prediction system performs accurately, consistently, and reliably under real operating conditions. The primary purpose of testing is to identify potential defects, validate system behavior, and confirm that all functional and non-functional requirements have been met.

In this project, system testing focuses on validating every major module, including the front- end user interface, Flask backend services, data preprocessing pipeline, feature extraction mechanisms, machine learning models, and database operations. Testing ensures that user input handling, real-time prediction workflow, model integration, and data storage operate correctly without errors or performance issues.

System testing was performed using different input combinations, including valid, invalid, and edge-case scenarios, to ensure correctness, robustness, and usability. Special emphasis was placed on early-stage prediction accuracy, handling incorrect inputs, and ensuring system stability during continuous usage.

7.1 Types of System Testing

7.1.1 Unit Testing

Unit testing was carried out to validate individual components of the system independently. Each module, including input validation, preprocessing functions, feature extraction logic, and prediction models, was tested with controlled inputs to verify expected outputs.

Key focus areas included:

- Input validation for numerical and textual fields
- Text preprocessing operations such as cleaning and tokenization
- Feature scaling and encoding mechanisms
- Individual model predictions (GBR, MLP, CNN)
- Database insertion and retrieval operations Unit testing ensured that every internal logical path executed correctly and no unexpected conditions occurred. Failures during this stage would have been easier to isolate and correct before integration.

7.1.2 Integration Testing

Integration testing examined whether combined components interacted correctly once they were linked together. Modules tested in combination included:

- Front-end input form communication with Flask backend
- Preprocessing pipeline feeding data into ML models
- Model outputs being passed to the stacking meta-model
- Backend communication with SQLite database for storing results

This testing stage confirmed that individually correct components functioned properly when executed together as a single rapid workflow. Particular attention was paid to preventing workflow crashes when components communicated sequentially.

7.1.3 Functional Testing

Functional testing validated that each feature of the system worked according to the specified requirements. Test cases were designed based on real user scenarios such as login, input submission, and prediction display.

Major validation rules included:

- Valid input data produces correct prediction output
- Invalid or empty inputs generate proper error messages
- Prediction results are displayed clearly on the interface
- User authentication and session handling work correctly
- Prediction history is stored and retrieved successfully

7.1.4 System Testing

System testing evaluated the entire system as a complete application. The focus was on overall system performance, reliability, and consistency under real-time conditions.

Tests verified:

- End-to-end workflow from user input to prediction output
- Performance of hybrid machine learning models

- Real-time response of the web application
- Stability of system during repeated predictions
- Accuracy of prediction after log transformation and inverse conversion

7.1.5 White-Box Testing

White-box testing was applied to internal coding logic, including data preprocessing pipelines, feature extraction processes, and model prediction algorithms. Testers observed internal variable flows such as processed feature vectors, scaled inputs, and intermediate model outputs. Execution paths within functions, conditional statements, and loops used in preprocessing and prediction stages were analyzed to ensure correctness. Special attention was given to transformations such as log1p and expm1, stacking model logic, and feature engineering steps to confirm that all computations were performed accurately and consistently.

7.1.6 Black-Box Testing

Black-box testing evaluated the system purely from the user's perspective without analyzing internal code structure. Inputs were submitted through the web interface, including video title, duration, likes, comments, and publication time, and prediction outputs were observed directly. The testing ensured that the system produced accurate view count predictions and handled incorrect or incomplete inputs gracefully. This approach was important for validating system usability, interface responsiveness, and proper error handling.

7.1.7 Acceptance Testing

Acceptance testing ensured that the system satisfied all specified requirements and user expectations. The system was evaluated based on usability, prediction accuracy, response time, and interface clarity. Stakeholders reviewed the ease of input submission, correctness of predicted outputs, and overall workflow of the application. Feedback confirmed that the system was intuitive, reliable, and capable of providing meaningful predictions for short video popularity.

7.2 Testing Strategies

A structured testing strategy was followed throughout the project lifecycle, ensuring systematic validation from individual module testing to complete system evaluation. Testing was conducted in stages, starting from unit-level verification to full system integration, ensuring that each component functioned correctly before combining with others.

7.2.1 Test Strategy and Approach

Testing was performed both manually and programmatically. Various input combinations were used to validate the performance and consistency of the prediction models. Primary strategic objectives included:

- Verifying correctness of preprocessing steps such as text cleaning, encoding, and scaling
- Ensuring smooth interaction between front-end interface and Flask backend
- Validating prediction accuracy under different input conditions
- Testing system performance under real-time prediction scenarios

7.2.2 Test Objectives

The following objectives guided all testing activities:

- The system must correctly process and validate all user input fields
- The prediction module must generate accurate results with minimal delay
- Invalid inputs must be handled gracefully with proper error messages
- The system must maintain consistency in prediction outputs
- The application should provide smooth and uninterrupted user experience

7.2.3 Features Tested

The major system features examined included:

- Input validation and error handling mechanisms
- Data preprocessing and feature extraction pipeline

- Machine learning model prediction accuracy
- Stacking (ensemble) model integration
- Database storage and retrieval of prediction history
- User authentication and session management

7.2.4 Integration Testing Strategy

Integration testing focused on verifying proper interaction between system components and preventing errors during data flow. Testing confirmed correct:

- Data transfer from front-end to backend server
- Flow of processed data into machine learning models
- Combination of model outputs using stacking technique
- Storage and retrieval of results from SQLite database

7.2.5 Acceptance Criteria

The system was accepted only when it satisfied the following conditions:

- Accurate prediction of video view count based on input data
- Stable and responsive web interface
- Proper handling of invalid and edge-case inputs
- Successful storage and retrieval of prediction history
- Compliance with all functional and non-functional requirements

7.2.6 Overall Test Results

All planned test cases were executed successfully. The short video popularity prediction system demonstrated stable performance, accurate prediction capability, and efficient real-time processing. The hybrid machine learning approach improved prediction reliability and reduced errors compared to single-model approaches.

7.3. Sample Test Cases

S No.	Test Case	Expected Result	Result	Remarks (if any)
01	User Login	User is authenticated and granted access to their dashboard	Pass	Verify incorrect credentials show appropriate error messages
02	User Registration	New user account is created and stored in the database	Pass	Validate empty form handling and database character limits
03	Prediction Input Submission	Valid input data is processed and prediction is generated	Pass	Test invalid inputs and missing fields
04	Prediction History	Stored prediction records are retrieved and displayed	Pass	Verify database retrieval and formatting
05	View Count Prediction Output	Predicted view count is displayed correctly	Pass	Verify transformation and output formatting

Table no. 1 Test Cases

Test Case 1:

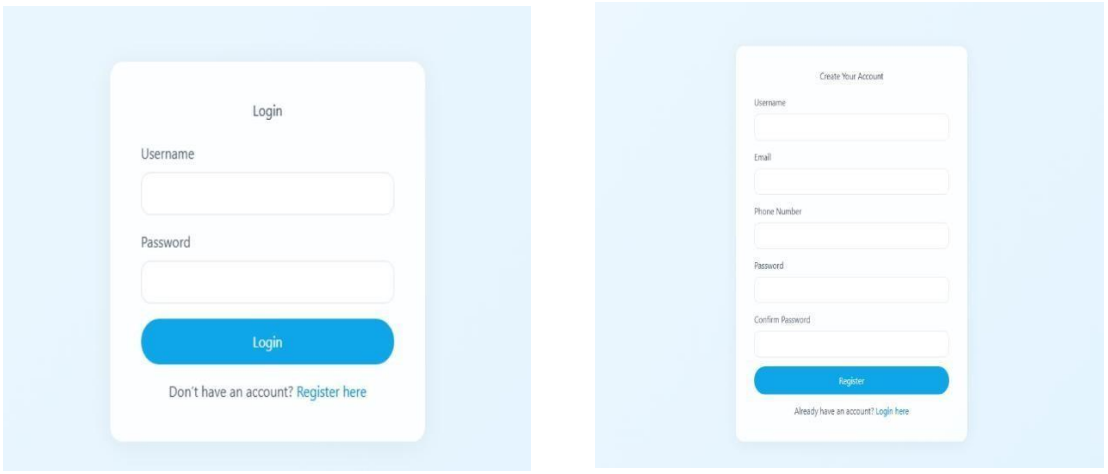

Fig 7.3.1 User Registration and Login Page

Description: The user initiates the login process by entering valid credentials (username and password) into the login form provided on the web application. Upon submission, the system securely validates the entered details by comparing them with the stored records in the database. If the credentials are correct, the system successfully authenticates the user and grants access to the application.

After successful authentication, the user is redirected to the dashboard page of the short video popularity prediction system. The dashboard provides a personalized interface where the user can access prediction features. It displays relevant account information such as username, email ID, and previously generated prediction history. The user can then enter video-related details such as title, duration, likes, comments, and upload time to generate predictions.

Test Case 2:

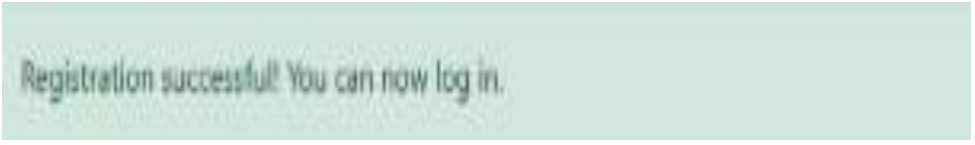

Fig 7.3.2 User Registration

Description: After submitting valid registration details, the system successfully creates a new user account and stores the information securely in the database. A confirmation message is displayed to the user indicating that the registration process has been completed successfully and the account is ready for login. This confirms that the registration module, input validation, and database storage mechanisms are functioning correctly, ensuring that new users can be added to the system in a secure and reliable manner.

Test Case 3:

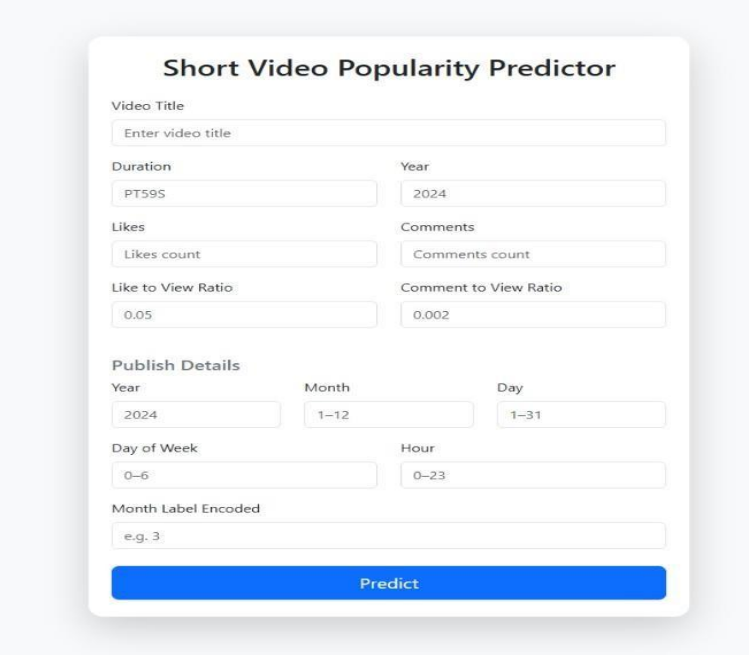

Fig.7.3.3 Short Video Popularity Prediction Interface

Description: The user accesses the short video popularity prediction interface and enters the required input details, including video title, duration, year, number of likes, number of comments, like-to-view ratio, comment-to-view ratio, and publish details such as year, month, day, day of the week, and hour. After filling all the fields with valid data, the user clicks on the “Predict” button to submit the request. The system processes the input by performing validation and preprocessing, then passes the data to the trained machine learning models for prediction. The predicted view count is generated and displayed to the user on the interface. This confirms that the input form handling, data validation, model integration, and output display functionalities are working correctly and efficiently.

Test Case 4:

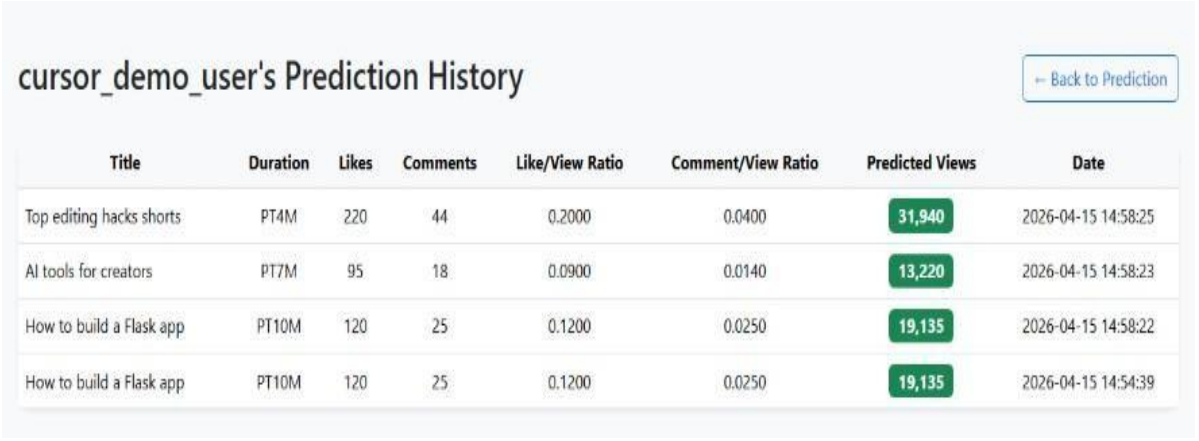

Fig 7.3.4. Prediction History

Description: The user navigates to the prediction history section after performing one or more predictions. The system retrieves stored records from the SQLite database and displays them in a tabular format. The table includes details such as video title, duration, number of likes, number of comments, like-to-view ratio, comment-to-view ratio, predicted view count, and the date and time of prediction. Each entry is displayed clearly, allowing the user to review previous predictions and analyze patterns over time. The “Back to Prediction” option enables the user to return to the main prediction interface seamlessly.

Test Case 5:

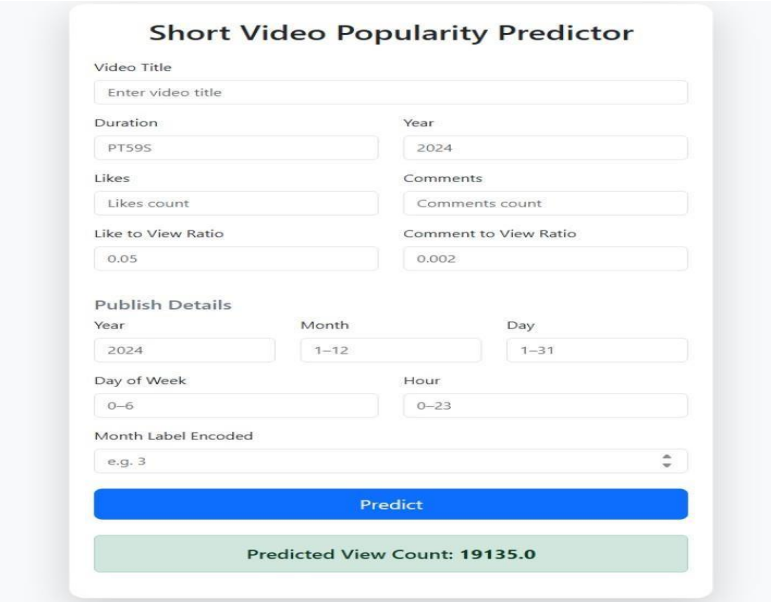

Fig 7.3.5 Prediction View Count

Description:The user enters valid video-related input details into the prediction form, including video title, duration, likes, comments, and publish information, and then clicks the "Predict" button. Upon submission, the system processes the input data through preprocessing and passes it to the trained hybrid machine learning models. The system successfully generates the prediction and displays the result on the interface as "Predicted View Count: 19135.0." The output is clearly shown below the prediction button in a highlighted format for easy visibility. This confirms that the end-to-end workflow—including input handling, data preprocessing, model prediction, inverse transformation, and output display—is functioning correctly and providing accurate results in real time.

8. OUTPUT SCREENS

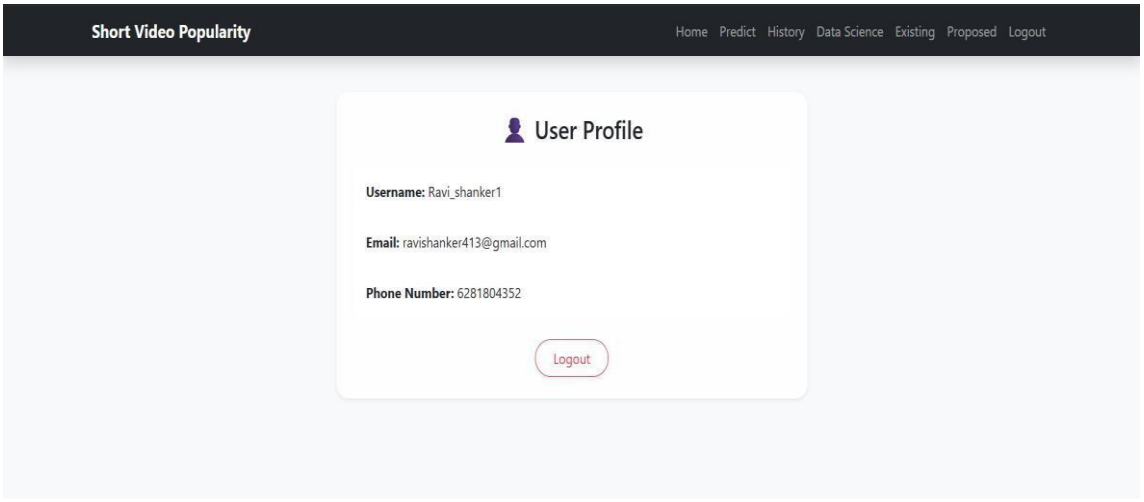

Fig 8.1. Home page

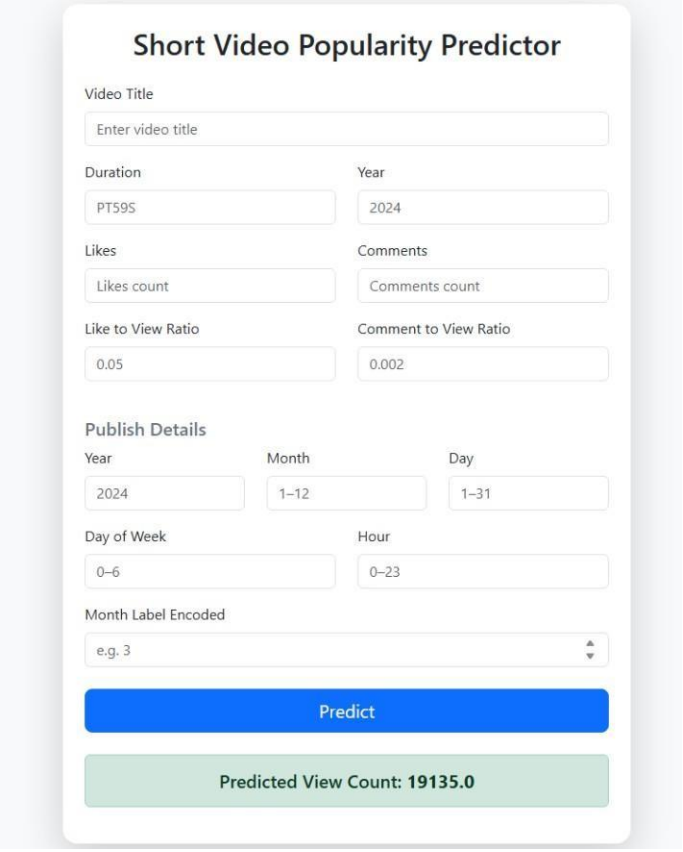

Fig 8.2. Short Video Popularity Predictor

Description: The system successfully generates the prediction and displays the result on the interface as "Predicted View Count: 19135.0." The output is clearly shown below the prediction button in a highlighted format for easy visibility. This confirms that the end-to-end workflow—including input handling, data preprocessing, model prediction, inverse transformation, and output display—is functioning correctly and providing accurate results in real time.

Fig 8.3. Prediction history

Description: The table includes details such as video title, duration, number of likes, number of comments, like-to-view ratio, comment-to-view ratio, predicted view count, and the date and time of prediction. Each entry is displayed clearly, allowing the user to review previous predictions and analyze patterns over time. The "Back to Prediction" option enables the user to return to the main prediction interface seamlessly. This confirms that the database storage, retrieval mechanism, and history display functionalities are working correctly and efficiently.

9. CONCLUSION

The short video popularity prediction system developed in this project demonstrates an effective and practical approach to estimating video view count at an early stage using hybrid machine learning techniques. The system successfully integrates data preprocessing, feature engineering, multi-model learning, and web-based deployment into a unified and scalable framework. By utilizing input features such as video title, duration, likes, comments, and publication time, the system eliminates dependency on post-publication metrics and enables early prediction of video performance.

The implementation of multiple machine learning models, including Gradient Boosting Regressor, Multi-Layer Perceptron Regressor, and Convolutional Neural Network, allows the system to capture both linear and complex non-linear relationships within the data. The stacking-based ensemble approach further enhances prediction accuracy by combining the strengths of individual models and reducing their limitations. This hybrid strategy ensures improved generalization, robustness, and consistency in prediction outcomes across different types of input data.

The system also effectively handles multi-modal data by combining textual, numerical, and temporal features into a unified representation. Advanced preprocessing techniques such as text cleaning, tokenization, encoding, and feature scaling contribute to improved model performance. The use of log transformation (log1p) and inverse transformation (expm1) ensures better handling of skewed data distributions, resulting in more stable and realistic predictions.

From an implementation perspective, the integration of machine learning models with a Flask- based web application provides a user-friendly and interactive platform. Users can easily input video details and obtain real-time predictions, while additional functionalities such as user authentication and prediction history storage enhance system usability. The modular design of the system allows for easy maintenance, scalability, and future enhancements without affecting existing components.

Extensive system testing confirmed that the application performs reliably under various conditions, including valid, invalid, and edge-case inputs. The system demonstrates stable performance, efficient response time, and accurate prediction results, making it suitable for real-world deployment. The testing process also ensured proper interaction between all modules, including front-end, backend, machine learning models, and database.

Overall, this project successfully bridges the gap between theoretical machine learning concepts and practical application by delivering a complete, efficient, and deployable solution for short video popularity prediction. The system can assist content creators, marketers, and platform providers in making informed decisions, optimizing content strategies, and improving audience engagement. With its scalable architecture and hybrid learning approach, the system lays a strong foundation for further research and development in predictive analytics for digital media platforms.

10. FUTURE ENHANCEMENTS

The current system establishes a strong foundation for predicting short video popularity using hybrid machine learning techniques; however, several enhancements can be introduced to further improve performance, scalability, and real-world applicability. Future developments may involve integrating advanced deep learning models such as BERT, RoBERTa, or transformer-based architectures to better capture semantic meaning from video titles and descriptions. These models can significantly enhance prediction accuracy by understanding contextual relationships and trends in textual data more effectively than traditional approaches.

The system can also be expanded by incorporating larger and more diverse datasets collected from multiple short video platforms such as YouTube Shorts, Instagram Reels, and TikTok. This would improve model generalization and allow the system to adapt to different content trends, user behaviors, and regional preferences. Additionally, multimodal analysis can be enhanced by including visual features from video thumbnails and frames, enabling the system to better understand content quality and visual appeal.

Real-time analytics capabilities can be integrated to continuously monitor trending patterns and update predictions dynamically. Implementing feedback mechanisms where users can validate or adjust predictions would enable incremental learning and improve long-term system accuracy. Furthermore, the addition of interactive dashboards and visualization tools would help users analyze performance trends, engagement patterns, and prediction insights more effectively.

While the current system is stable and functionally effective, several advanced enhancements can significantly increase its practical utility and scalability:

1. **Advanced Deep Learning Models:**
   Integration of transformer-based models such as BERT or GPT-based embeddings to improve textual feature extraction and semantic understanding.
2. **Multimodal Data Integration:**
   Incorporating video thumbnails, frames, and audio features to enhance prediction accuracy by analyzing visual and auditory cues.

3. **Real-Time Trend Analysis:**
   Implementing live data streaming and trend monitoring to provide dynamic and up-to-date prediction results.
4. **Cloud-Based Deployment:**
   Migrating the system to cloud platforms such as AWS or Google Cloud to support large-scale usage and real-time access.
5. **Mobile Application Development:**
   Developing Android and iOS applications to allow users to access prediction features on mobile devices.
6. **Recommendation System Integration:**
   Extending the system to suggest content optimization strategies, such as best upload time, title improvements, and engagement techniques.
7. **User Feedback and Adaptive Learning:**
   Incorporating feedback loops to continuously improve model performance through retraining based on user inputs.
8. **Scalability and Big Data Handling:**
   Enhancing the system to handle large datasets using distributed computing frameworks like Hadoop or Spark.
9. **Improved Model Optimization:**
   Using lightweight models and optimization techniques to reduce prediction latency and improve real-time performance.
10. **Data Privacy and Security:**
    Implementing secure data handling practices such as encryption and user authentication to protect sensitive information.
11. **Cross-Platform Web Integration:**
    Making the application fully responsive and compatible across desktops, tablets, and mobile browsers.
12. **Explainable AI (XAI):**
    Adding interpretability features to explain how predictions are generated, increasing transparency and user trust.

REFERENCES

[1] X. Ma et al., "Deep Attention Networks for Short Video Popularity Prediction," *IEEE Transactions on Multimedia*, 2026.

[2] Z. He and D. Li, "Short Video Popularity Prediction Using IoT and Deep Learning Regression Models," *IEEE Access*, 2026.

[3] Y. Liu et al., "Transformer-Based Multimodal Learning for Short Video Popularity Forecasting," *Neurocomputing*, 2026.

[4] S. Patel and R. Mehta, "Deep Learning Approaches for Social Media Video Popularity Prediction," *IEEE Access*, 2025.

[5] J. Kim and H. Lee, "Multimodal Deep Neural Networks for Predicting Online Video Popularity," *Information Sciences*, 2025.

[6] S. Patel and R. Mehta, "Deep Learning Approaches for Social Media Video Popularity Prediction," *IEEE Access*, 2025.

[7] J. Kim and H. Lee, "Multimodal Deep Neural Networks for Predicting Online Video Popularity," *Information Sciences*, 2025.

[8] R. Singh and A. Verma, "LSTM-Based Temporal Modeling for Video Popularity Prediction," *Pattern Recognition Letters*, 2024.

[9] T. Chen and X. Zhou, "CNN-Based Feature Extraction for Online Video Popularity Prediction," *Multimedia Tools and Applications*, 2023.

[10] L. Zhao and Y. Wu, "Predicting Online Video Popularity Using Machine Learning Techniques," *Future Generation Computer Systems*, 2022.

[11] Y. Zhang and Q. Liu, "Predicting Video Popularity with Metadata and Content Features," *IEEE Access*, 2022.

[12] H. Li and X. Wu, "Deep Neural Networks for Online Video Popularity Prediction," *ACM Multimedia*, 2022.

[13] P. Kumar and S. Reddy, "Machine Learning Techniques for Social Media Content Popularity Prediction," *Procedia Computer Science*, 2022.

[14] X. Zhou and T. Chen, "Understanding Video Popularity Dynamics on Social Media Platforms," *Multimedia Tools and Applications*, 2021.

[15] L. Wang and H. Chen, "Deep Learning-Based Analysis of Online Video Popularity," *Neurocomputing*, 2021.

[16] S. Gupta and R. Kumar, "Hybrid Machine Learning Models for Predicting Social Media Engagement," *Expert Systems with Applications*, 2021.

[17] M. Tan and J. Lee, “Multimodal Feature Learning for Video Popularity Prediction,” *Pattern Recognition Letters*, 2020.

[18] D. Singh and P. Sharma, “Temporal Modeling of Video Popularity Using Recurrent Neural Networks,” *Knowledge-Based Systems*, 2020.

[19] A. Roy and S. Das, “Predicting Viral Content Using Deep Neural Networks,” *IEEE Access*, 2020.

[20] K. Patel and N. Shah, “Machine Learning Approaches for YouTube Video Popularity Prediction,” *Future Generation Computer Systems*, 2019.