National University of Computer and Emerging Sciences
Islamabad Campus

AI Course Project

Filmception

AI-2002

Artificial Intelligence

Submitted by: Kumail Haider & Jibran Hanan
Roll number: i22-1723 & i221732
Section: D
Date: 06/05/2025

Table of Contents

Introduction

For this course project, I developed a comprehensive movie analysis system that combines machine learning and language processing technologies. The primary goal was to create a practical application that could automatically classify movie genres based on plot summaries while also making content accessible in multiple languages through translation and audio generation.

Project Objectives

- To implement a machine learning model capable of predicting movie genres from text summaries
- To develop a multilingual translation system for movie summaries
- To create an audio generation system supporting multiple languages
- To integrate these features into a user-friendly interface

Data Preprocessing Methodology (preprocess_movie_data.py)

The script handles two primary data sources:

1. Plot Summaries (plot_summaries.txt)
2. Movie Metadata (movie.metadata.tsv)

Text Cleaning Process

The preprocessing pipeline implements several cleaning steps through the clean_text() function:

1. Case Normalization: Converting all text to lowercase
2. Special Character Removal: Removing non-alphabetic characters
3. Whitespace Normalization: Removing extra spaces

Text Processing Pipeline

The script implements advanced NLP techniques through tokenize_and_lemmatize():

1. Tokenization: Breaking down text into individual words
2. Stopword Removal: Eliminating common English words that don't carry significant meaning
3. Lemmatization: Converting words to their base form using NLTK's WordNetLemmatizer

Genre Extraction Process

The extract_genres() function implements sophisticated genre processing:

1. Parses JSON-formatted genre data from metadata
2. Filters out non-genre categories (e.g., "Silent film", "Black-and-white")
3. Creates a clean list of relevant genres for each movie

Data Organization

The script organizes processed data into structured format:

{

'movie_id': unique identifier,

'summary': processed and cleaned summary,

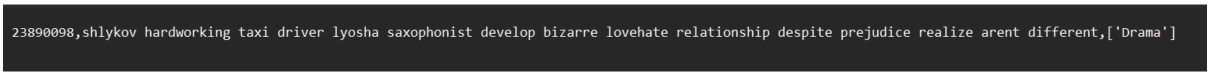
'genres': list of relevant genres

}

Dataset Split

The preprocessing pipeline automatically:

- Performs a train-test split (80-20 ratio)
- Saves three separate CSV files:
    1. processed_movie_data.csv: Complete processed dataset
    2. train_data.csv: Training dataset (80%)
    3. test_data.csv: Testing dataset (20%)



```
23890098,shlykov hardworking taxi driver lyosha saxophonist develop bizarre lovehate relationship despite prejudice realize arent different,['Drama']
```

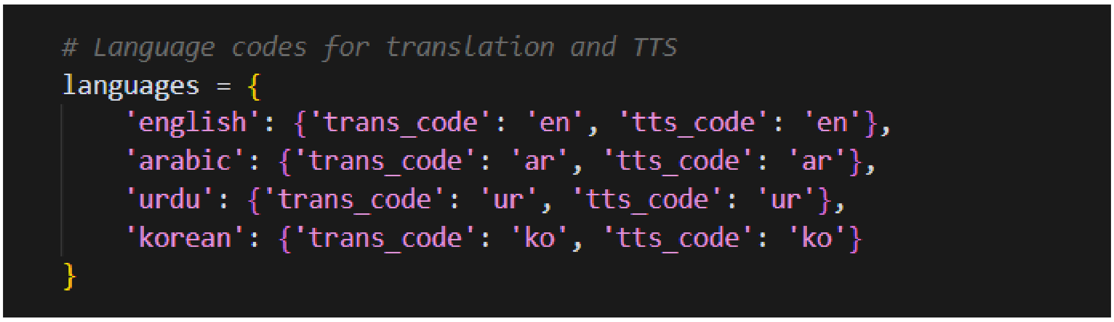
Figure 1(cleaned data example)

Translation and Audio Generation System (translate_and_audio.py)

Overview

The project implements a comprehensive multilingual support system that converts movie summaries into different languages and generates corresponding audio narrations. This feature enhances accessibility and provides multilingual content delivery.

The system supports four languages:



```
# Language codes for translation and TTS
languages = {
    'english': {'trans_code': 'en', 'tts_code': 'en'},
    'arabic': {'trans_code': 'ar', 'tts_code': 'ar'},
    'urdu': {'trans_code': 'ur', 'tts_code': 'ur'},
    'korean': {'trans_code': 'ko', 'tts_code': 'ko'}
}
```

Figure 2

Translation System

Implementation Details

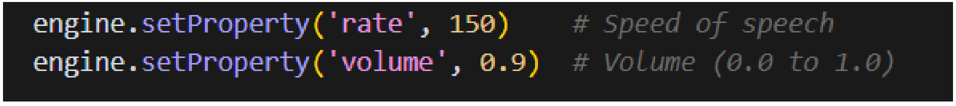
1. Translation Engine:
    - Utilizes Google Translator API through the deep_translator library
    - Implement rate limiting to avoid API restrictions:
      time.sleep(0.5) # Delay between translations

2. Error Handling
    - Robust error handling for translation failures
    - Graceful degradation when translation services are unavailable
    -

Audio Generation System

Dual TTS Implementation

1. English Audio Generation
    - Uses pyttsx3 engine for English text
    - Customizable speech parameters:



```
engine.setProperty('rate', 150)     # Speed of speech
engine.setProperty('volume', 0.9)  # Volume (0.0 to 1.0)
```

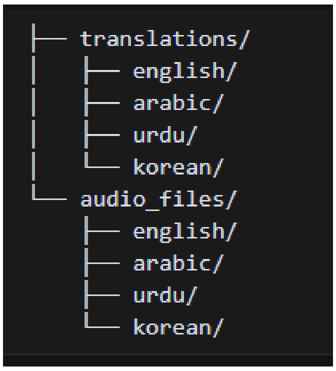
Figure 3

Non-English Audio Generation

1. Employs gTTS (Google Text-to-Speech) for other languages

2. Optimized for quality and reliability

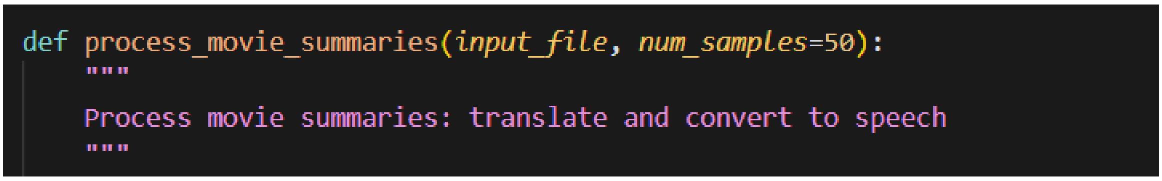
File Structure



```
├── translations/
│   ├── english/
│   ├── arabic/
│   ├── urdu/
│   └── korean/
└── audio_files/
    ├── english/
    ├── arabic/
    ├── urdu/
    └── korean/
```

Processing Pipeline

- **Input Processing**
    - Reads processed movie summaries
    - Supports batch processing with customizable sample size



```python
def process_movie_summaries(input_file, num_samples=50):
    """
    Process movie summaries: translate and convert to speech
    """
```

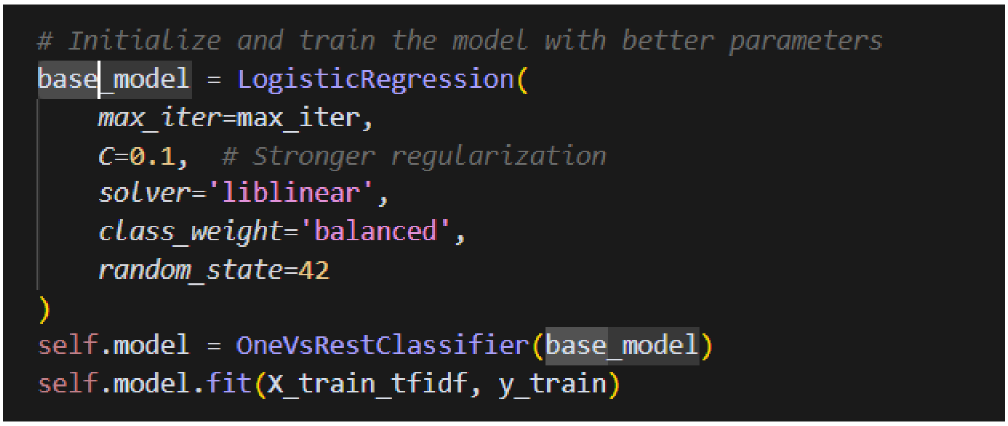
Figure 4

- **Translation Flow**
    - Text translation
    - Translation verification
    - Text file storage
- **Audio Generation Flow**
    - Language-specific TTS selection
    - Audio file generation
    - Quality verification

Genre Prediction Model (train_genre_model.py)

Model Selection

The project uses Logistic Regression with OneVsRestClassifier instead of deep architectures for several justified reasons:



```python
# Initialize and train the model with better parameters
base_model = LogisticRegression(
    max_iter=max_iter,
    C=0.1,  # Stronger regularization
    solver='liblinear',
    class_weight='balanced',
    random_state=42
)
self.model = OneVsRestClassifier(base_model)
self.model.fit(X_train_tfidf, y_train)
```

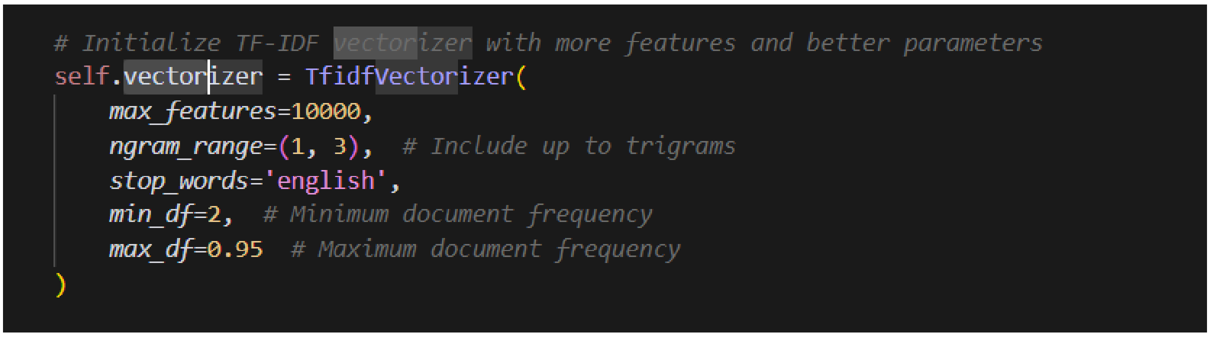
Figure 5

Justification:

- Data Size and Complexity
    - Suitable for medium-sized text datasets
    - Efficient with sparse features (TF-IDF matrices)
    - Less prone to overfitting compared to deep architectures
- Interpretability
    - Provides clear feature importance weights
    - Easier to debug and understand predictions
    - Transparent confidence scores
- Computational Efficiency
    - Faster training compared to deep learning models
    - Lower resource requirements
    - Quicker inference time

Feature Extraction and Input Processing

The system implements sophisticated text processing and feature extraction:



```
# Initialize TF-IDF vectorizer with more features and better parameters
self.vectorizer = TfidfVectorizer(
    max_features=10000,
    ngram_range=(1, 3),  # Include up to trigrams
    stop_words='english',
    min_df=2,  # Minimum document frequency
    max_df=0.95  # Maximum document frequency
)
```

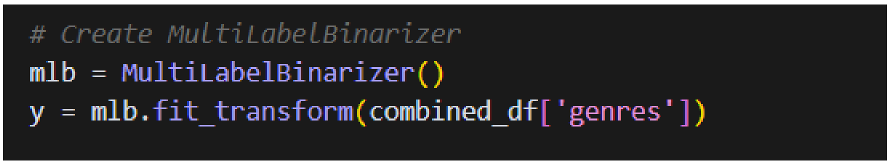
Figure 6

Implementation Details:

- Text Preprocessing
  - Case normalization
  - Special character removal
  - Whitespace normalization
- TF-IDF Features
  - Captures word importance in context
  - Handles rare and common words appropriately
  - N-gram support (up to trigrams)
- Feature Selection
  - Maximum 10,000 features to prevent overfitting
  - Minimum document frequency of 2
  - Maximum document frequency of 95%

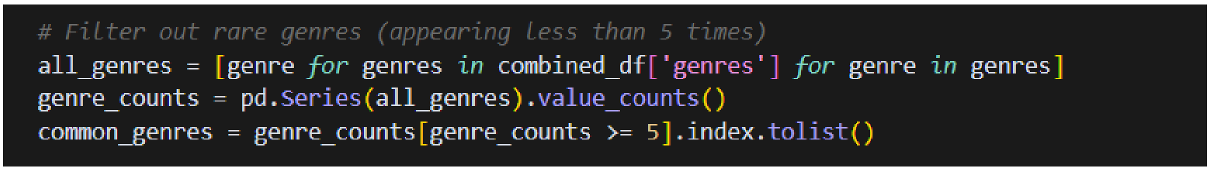
Multi-label Classification Handling

The system effectively handles multiple genres per movie:



```
# Create MultiLabelBinarizer
mlb = MultiLabelBinarizer()
y = mlb.fit_transform(combined_df['genres'])
```

Figure 7



```
# Filter out rare genres (appearing less than 5 times)
all_genres = [genre for genres in combined_df['genres'] for genre in genres]
genre_counts = pd.Series(all_genres).value_counts()
common_genres = genre_counts[genre_counts >= 5].index.tolist()
```

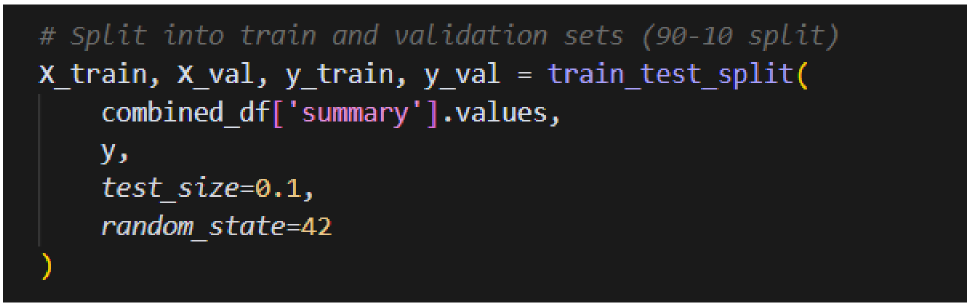
Figure 8

Key Features:

- Label Processing
    - Binary relevance approach
    - Handles overlapping genres
    - Filters rare genres
- Prediction Mechanism
    - Probability threshold of 0.3
    - Returns top 2 most likely genres
    - Includes confidence scores

Training and Test Set Preparation

The dataset is split with careful consideration:



```
# Split into train and validation sets (90-10 split)
X_train, X_val, y_train, y_val = train_test_split(
    combined_df['summary'].values,
    y,
    test_size=0.1,
    random_state=42
)
```

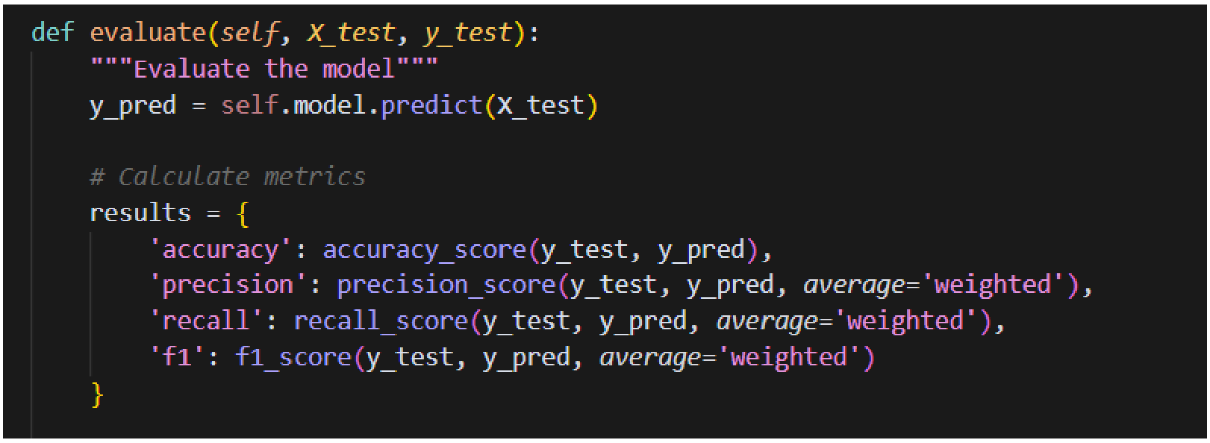
Figure 9

Split Rationale:

- 90% training, 10% validation split
- Random state fixed for reproducibility
- Stratified split maintaining genre distribution

Evaluation Metrics

Comprehensive evaluation metrics are implemented:



```
def evaluate(self, X_test, y_test):
    """Evaluate the model"""
    y_pred = self.model.predict(X_test)

    # Calculate metrics
    results = {
        'accuracy': accuracy_score(y_test, y_pred),
        'precision': precision_score(y_test, y_pred, average='weighted'),
        'recall': recall_score(y_test, y_pred, average='weighted'),
        'f1': f1_score(y_test, y_pred, average='weighted')
    }
```

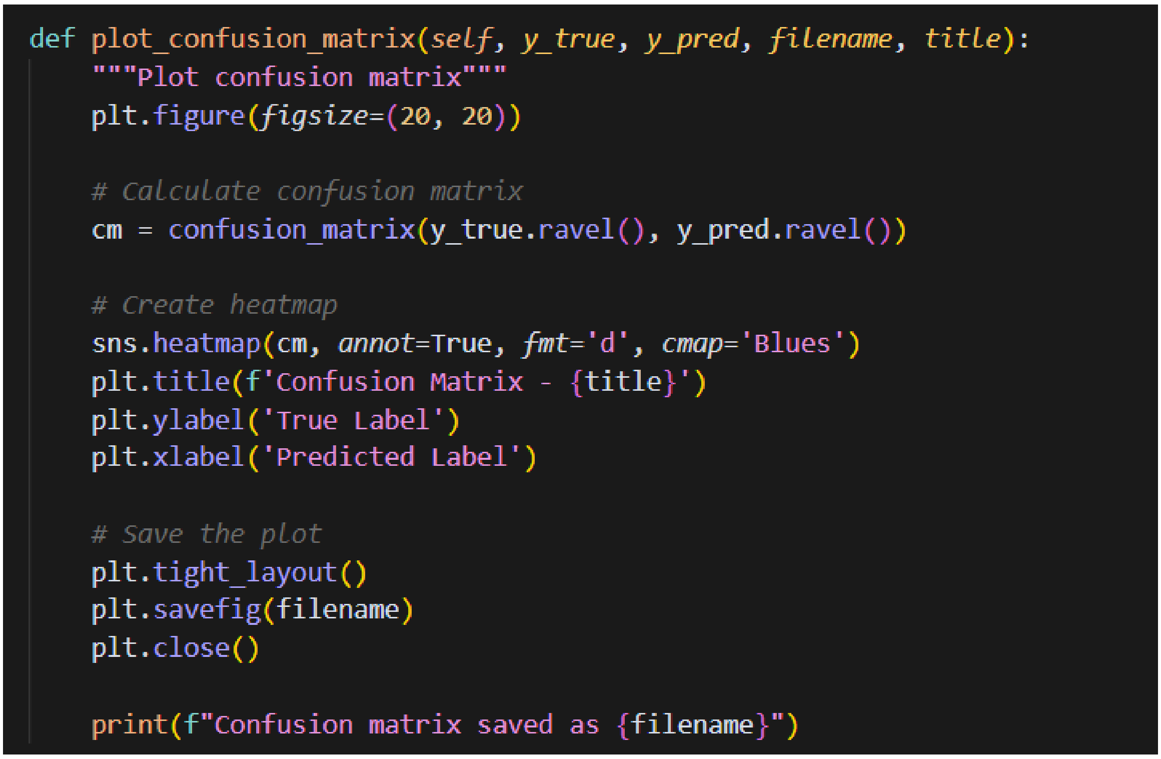
Figure 10

Metrics Implementation:

- Overall Metrics
  - Accuracy for general performance
  - Weighted precision for prediction quality
  - Weighted recall for coverage
  - Weighted F1-score for balanced measure
- Per-Genre Metrics
  - Individual genre performance
  - Class-wise precision and recall
  - Detailed classification report

Confusion Matrix

The system generates detailed confusion matrices:



```
def plot_confusion_matrix(self, y_true, y_pred, filename, title):
    """Plot confusion matrix"""
    plt.figure(figsize=(20, 20))

    # Calculate confusion matrix
    cm = confusion_matrix(y_true.ravel(), y_pred.ravel())

    # Create heatmap
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix - {title}')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')

    # Save the plot
    plt.tight_layout()
    plt.savefig(filename)
    plt.close()

    print(f"Confusion matrix saved as {filename}")
```

Figure 11

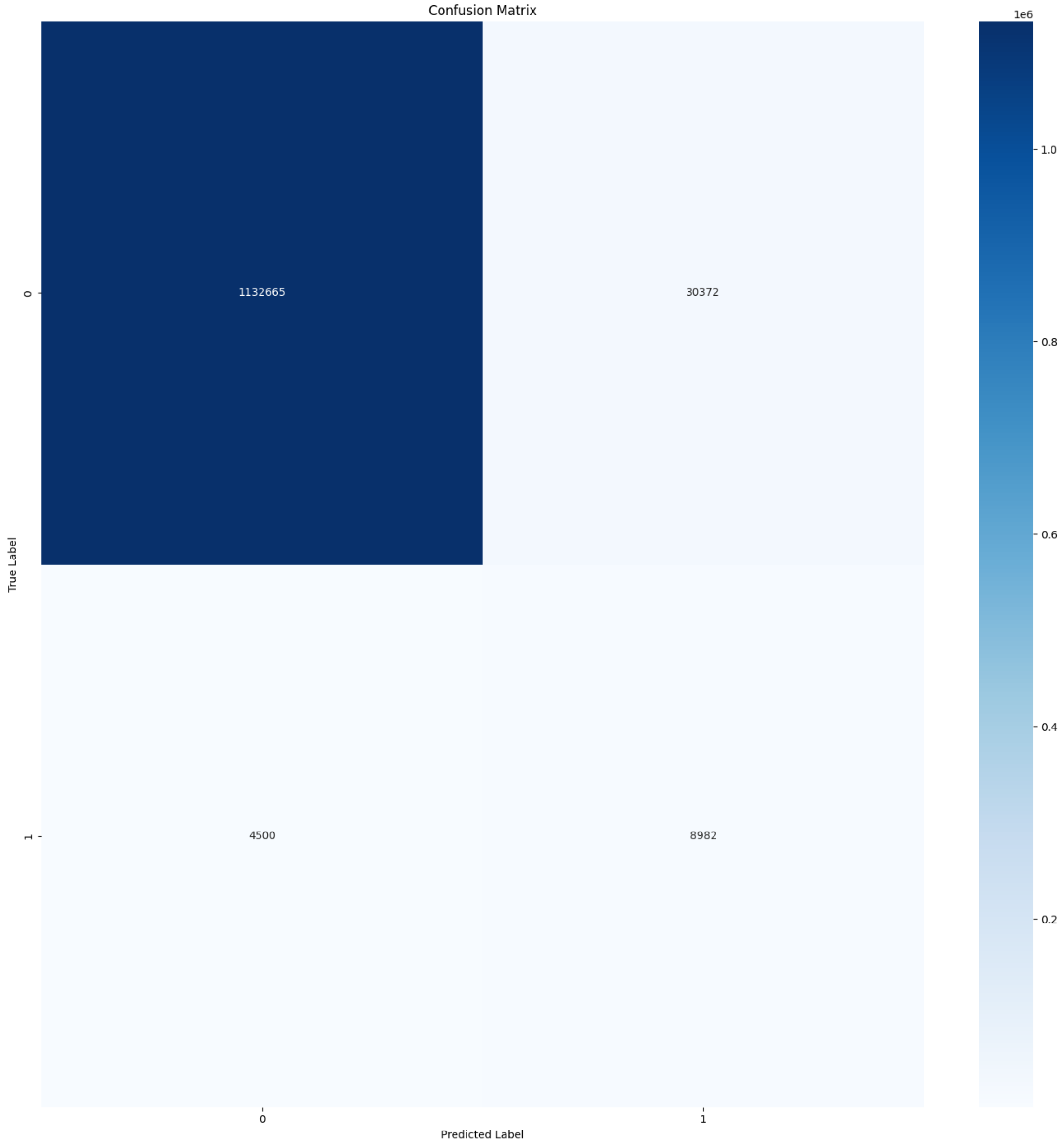


Figure 12

Visualization Features:

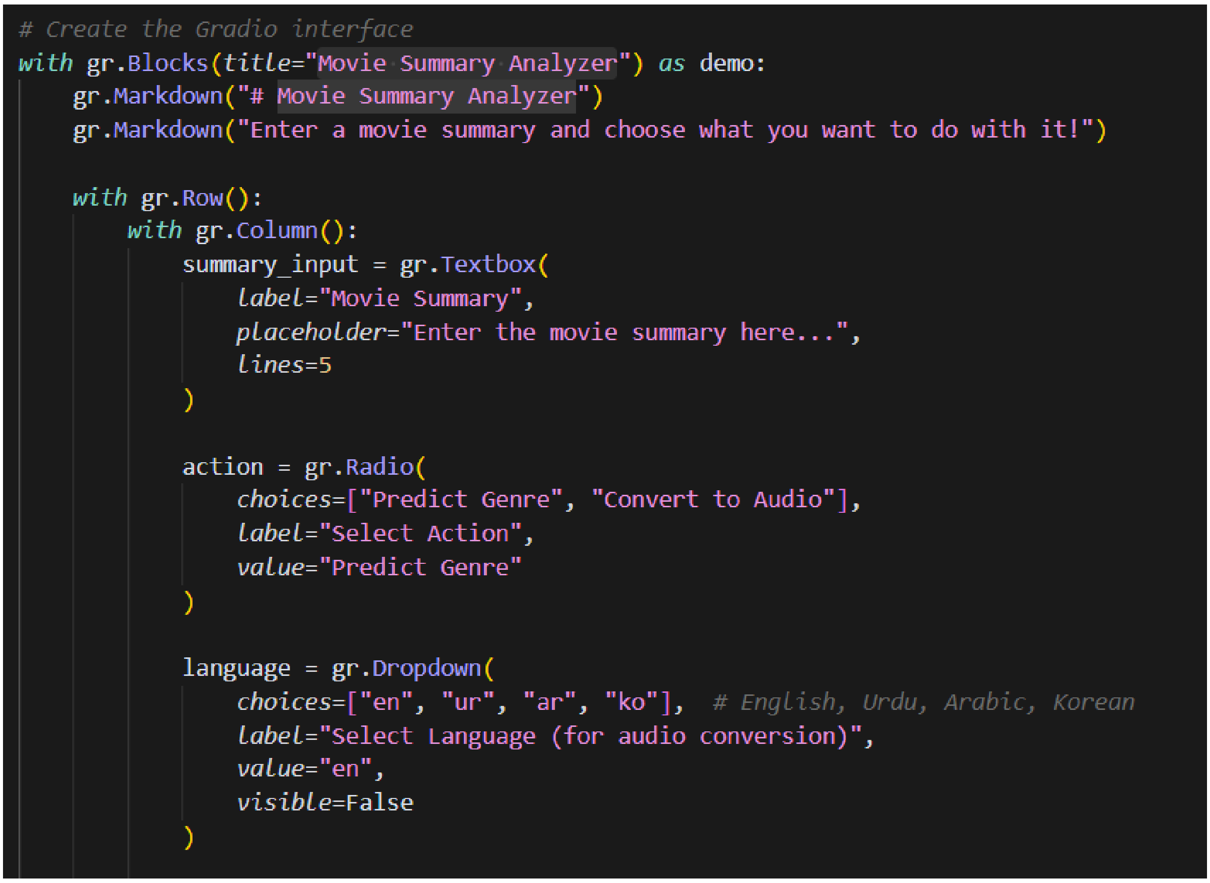
- Training Set Matrix
  - Shows model learning effectiveness
  - Identifies training biases
- Testing Set Matrix
  - Validates generalization
  - Highlights potential overfitting

  - Identifies misclassification patterns

GUI (app.py)

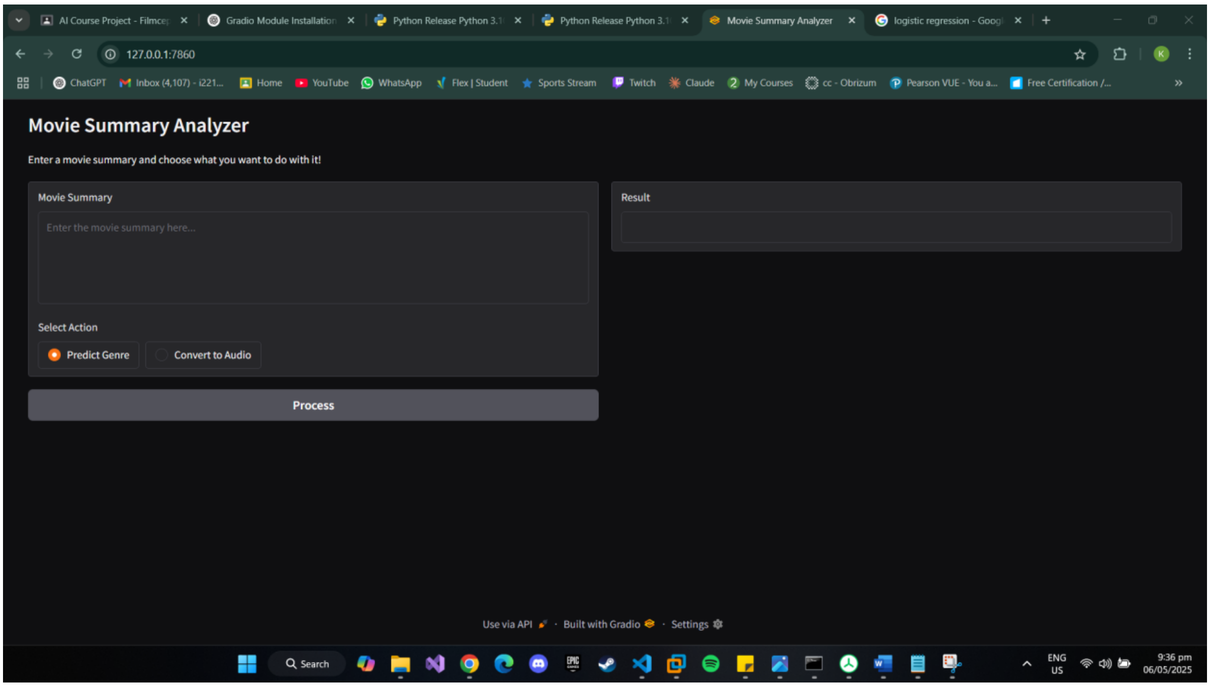
Menu-based Interaction

The system implements a Gradio-based web interface that provides a clean, intuitive interaction flow:



```python
# Create the Gradio interface
with gr.Blocks(title="Movie Summary Analyzer") as demo:
    gr.Markdown("# Movie Summary Analyzer")
    gr.Markdown("Enter a movie summary and choose what you want to do with it!")

    with gr.Row():
        with gr.Column():
            summary_input = gr.Textbox(
                label="Movie Summary",
                placeholder="Enter the movie summary here...",
                lines=5
            )

            action = gr.Radio(
                choices=["Predict Genre", "Convert to Audio"],
                label="Select Action",
                value="Predict Genre"
            )

            language = gr.Dropdown(
                choices=["en", "ur", "ar", "ko"],  # English, Urdu, Arabic, Korean
                label="Select Language (for audio conversion)",
                value="en",
                visible=False
            )
```

Figure 13



Movie Summary Analyzer

Enter a movie summary and choose what you want to do with it!

Movie Summary

Enter the movie summary here...

Result

Select Action

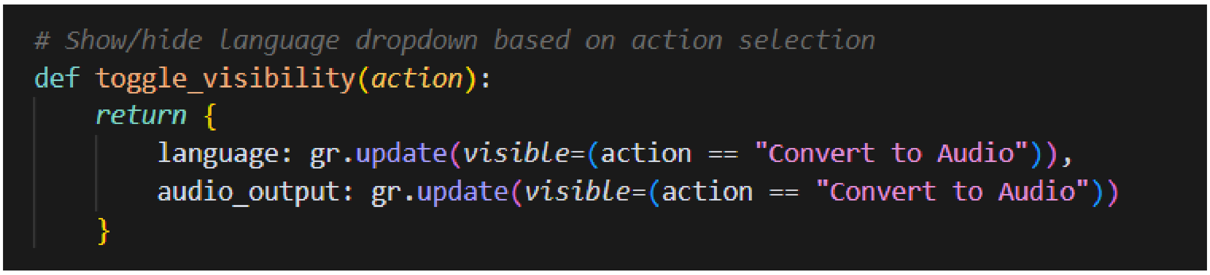
● Predict Genre ○ Convert to Audio

Process

Figure 14

Key Interface Features:

- Input Section
  - Large text area for movie summaries
  - Clear action selection radio buttons
  - Language dropdown for audio conversion
- Dynamic Component Visibility
  - Language selector appears only for audio conversion
  - Audio player shows up only when needed



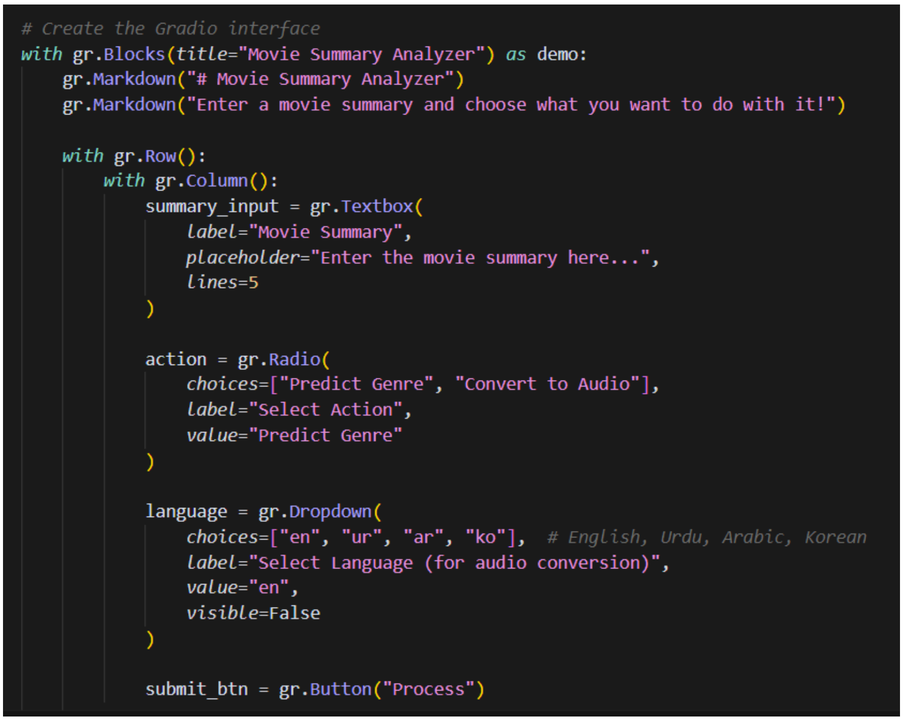
```
# Show/hide language dropdown based on action selection
def toggle_visibility(action):
    return {
        language: gr.update(visible=(action == "Convert to Audio")),
        audio_output: gr.update(visible=(action == "Convert to Audio"))
    }
```

Figure 15

Smooth Flow and User Experience

The interface ensures a seamless user experience:

- Responsive Design



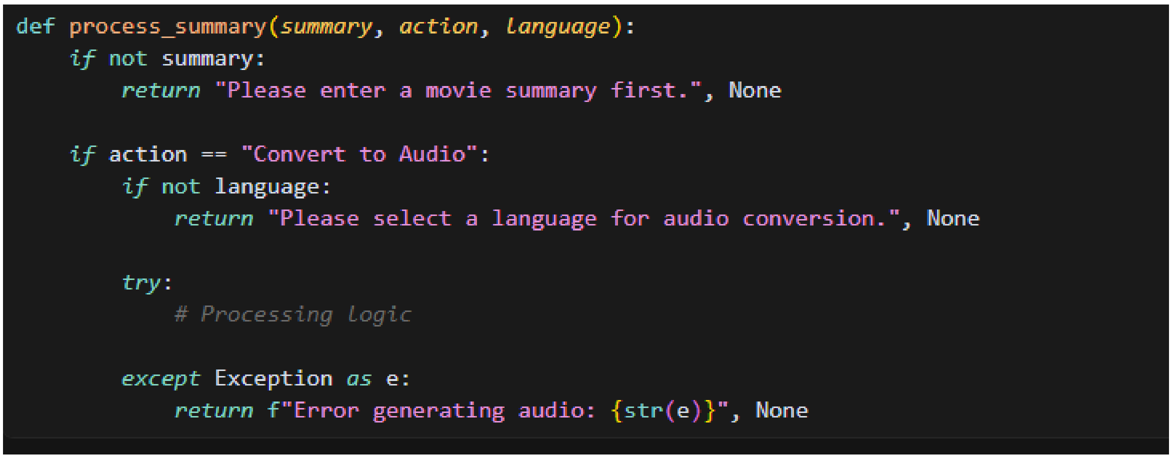
```python
# Create the Gradio interface
with gr.Blocks(title="Movie Summary Analyzer") as demo:
    gr.Markdown("# Movie Summary Analyzer")
    gr.Markdown("Enter a movie summary and choose what you want to do with it!")

    with gr.Row():
        with gr.Column():
            summary_input = gr.Textbox(
                label="Movie Summary",
                placeholder="Enter the movie summary here...",
                lines=5
            )

            action = gr.Radio(
                choices=["Predict Genre", "Convert to Audio"],
                label="Select Action",
                value="Predict Genre"
            )

            language = gr.Dropdown(
                choices=["en", "ur", "ar", "ko"],  # English, Urdu, Arabic, Korean
                label="Select Language (for audio conversion)",
                value="en",
                visible=False
            )

            submit_btn = gr.Button("Process")
```

Figure 16

- Interactive Updates
    - Real-time component visibility changes
    - Immediate feedback on actions
    - Clear progression of tasks
- User Guidance
    - Descriptive labels and placeholders
    - Intuitive action flow
    - Clear section organization

Error Handling and Output Messages

The system implements comprehensive error handling and user feedback:

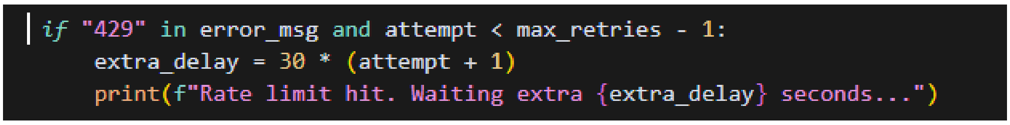


```python
def process_summary(summary, action, language):
    if not summary:
        return "Please enter a movie summary first.", None

    if action == "Convert to Audio":
        if not language:
            return "Please select a language for audio conversion.", None

        try:
            # Processing logic

        except Exception as e:
            return f"Error generating audio: {str(e)}", None
```

Figure 17

Error Handling Features:

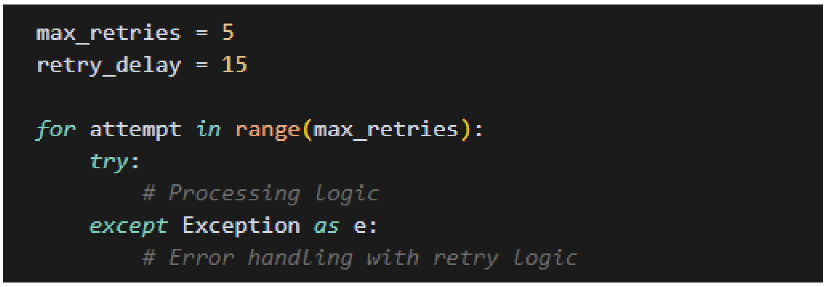
- Input Validation
    - Checks for empty summaries
    - Validates language selection
    - Ensures proper action selection
- Processing Errors
    - Handles translation failures gracefully
    - Manages API rate limiting



```python
if "429" in error_msg and attempt < max_retries - 1:
    extra_delay = 30 * (attempt + 1)
    print(f"Rate limit hit. Waiting extra {extra_delay} seconds...")
```

Figure 18

- User Feedback
    - Clear success messages
    - Detailed error explanations
    - Processing status updates
- Retry Mechanism



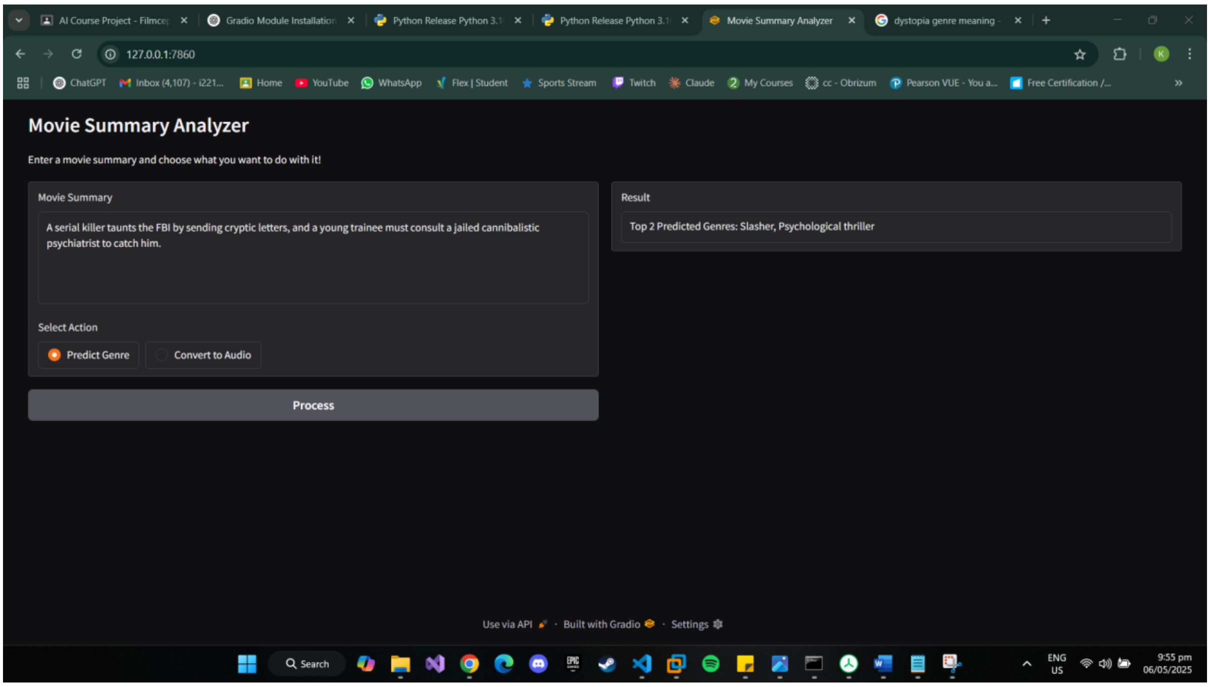
```python
max_retries = 5
retry_delay = 15

for attempt in range(max_retries):
    try:
        # Processing logic
    except Exception as e:
        # Error handling with retry logic
```

Figure 19

Output

Use case 1

A serial killer taunts the FBI by sending cryptic letters, and a young trainee must consult a jailed cannibalistic psychiatrist to catch him.
(Genre: Thriller / Crime / Horror)



Movie Summary Analyzer

Enter a movie summary and choose what you want to do with it!

Movie Summary

A serial killer taunts the FBI by sending cryptic letters, and a young trainee must consult a jailed cannibalistic psychiatrist to catch him.

Select Action

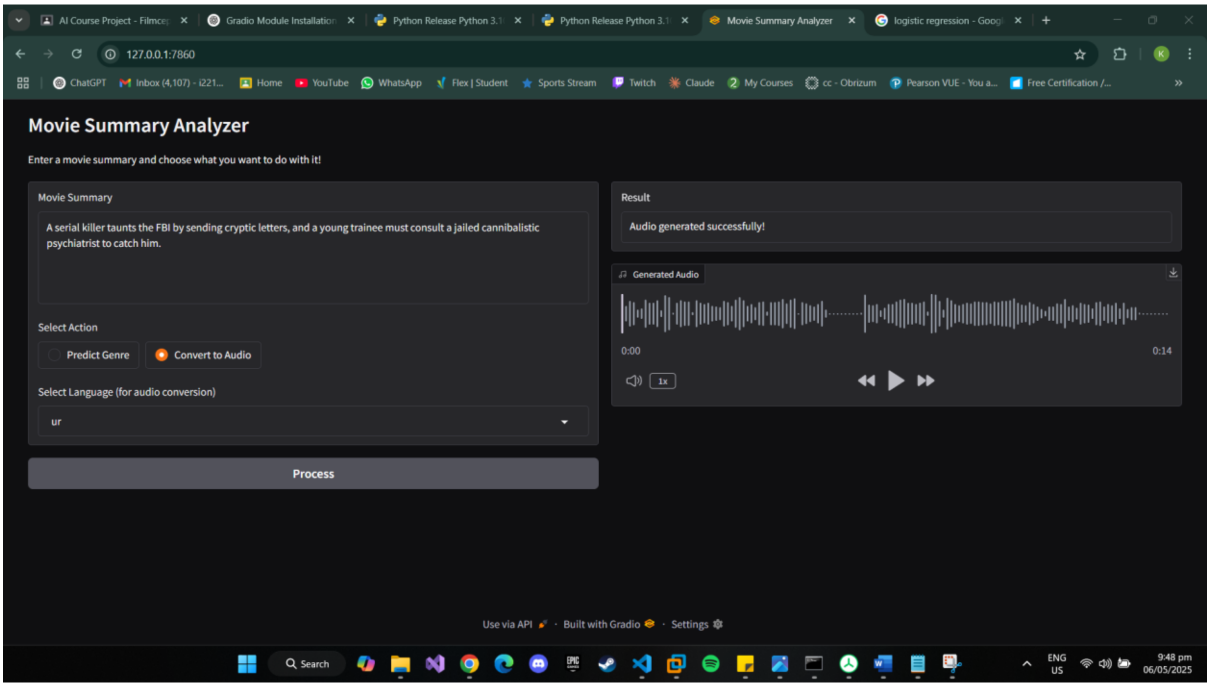
● Predict Genre ○ Convert to Audio

Process

Result

Top 2 Predicted Genres: Slasher, Psychological thriller

Figure 20



Movie Summary Analyzer

Enter a movie summary and choose what you want to do with it!

Movie Summary: A serial killer taunts the FBI by sending cryptic letters, and a young trainee must consult a jailed cannibalistic psychiatrist to catch him.

Select Action: ○ Predict Genre ● Convert to Audio

Select Language (for audio conversion): ur

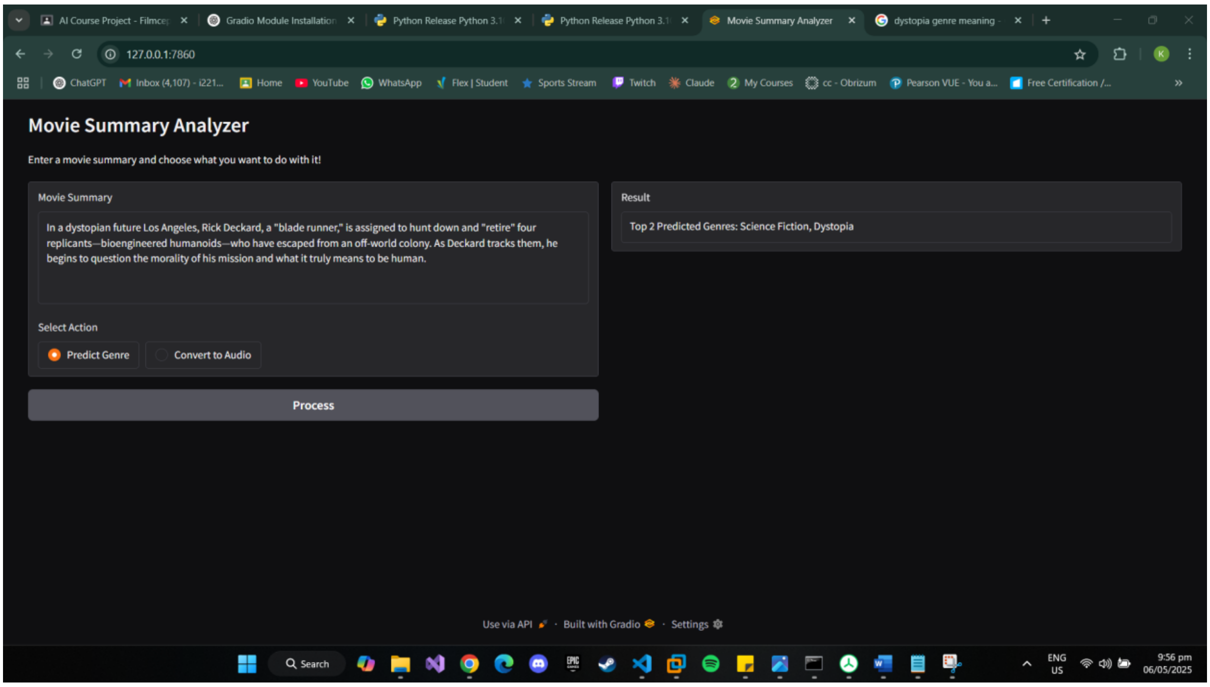
Result: Audio generated successfully!

Figure 21

Use Case 2

In a dystopian future Los Angeles, Rick Deckard, a "blade runner," is assigned to hunt down and "retire" four replicants—bioengineered humanoids—who have escaped from an off-world colony. As Deckard tracks them, he begins to question the morality of his mission and what it truly means to be human.

(Genre: Science Fiction)



Movie Summary Analyzer

Enter a movie summary and choose what you want to do with it!

Movie Summary

In a dystopian future Los Angeles, Rick Deckard, a "blade runner," is assigned to hunt down and "retire" four replicants—bioengineered humanoids—who have escaped from an off-world colony. As Deckard tracks them, he begins to question the morality of his mission and what it truly means to be human.

Select Action

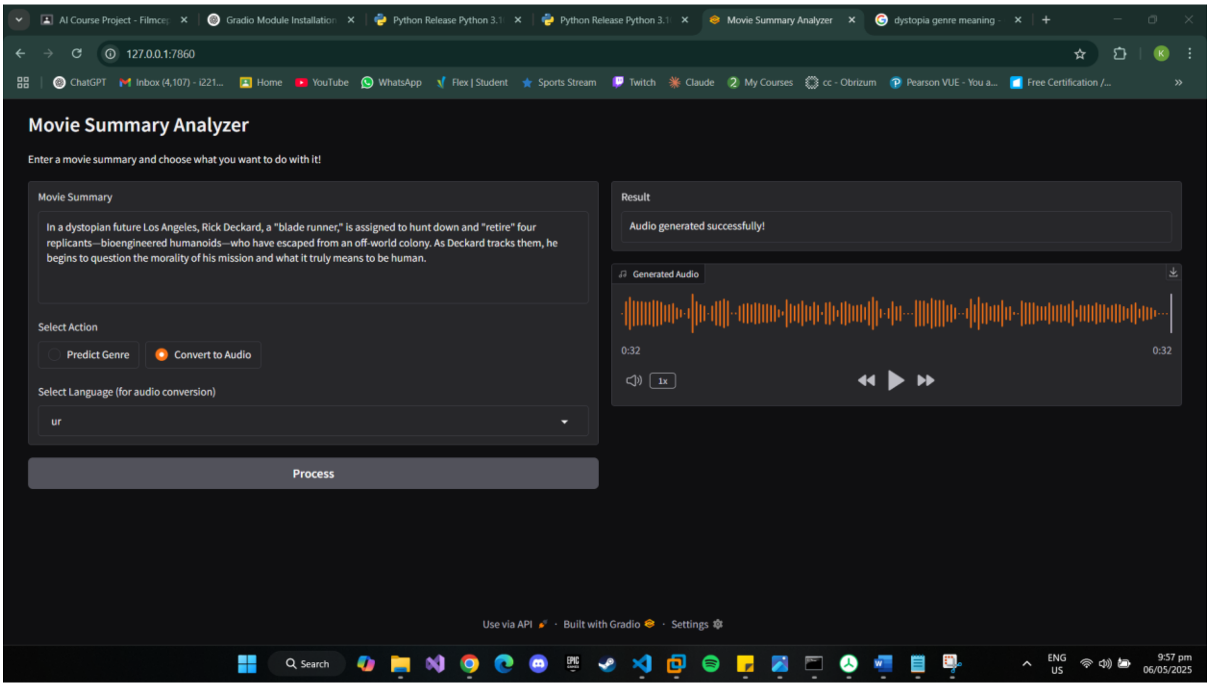
● Predict Genre ○ Convert to Audio

Process

Result

Top 2 Predicted Genres: Science Fiction, Dystopia

Figure 22



Movie Summary Analyzer

Enter a movie summary and choose what you want to do with it!

Movie Summary

In a dystopian future Los Angeles, Rick Deckard, a "blade runner," is assigned to hunt down and "retire" four replicants—bioengineered humanoids—who have escaped from an off-world colony. As Deckard tracks them, he begins to question the morality of his mission and what it truly means to be human.

Select Action

○ Predict Genre ● Convert to Audio

Select Language (for audio conversion)

ur

Process

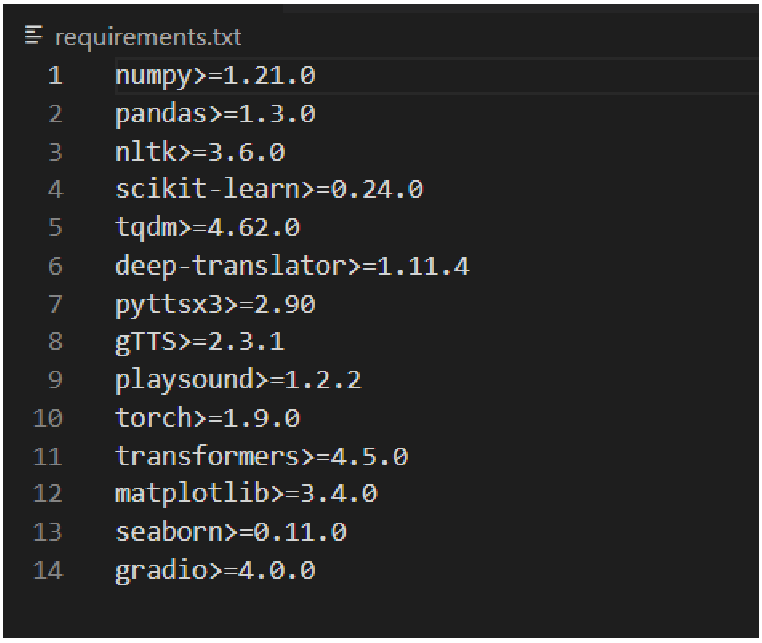
Result

Audio generated successfully!

Figure 23

Requirements

The following were the requirements for making of the Filmception project:



requirements.txt

```
numpy>=1.21.0
pandas>=1.3.0
nltk>=3.6.0
scikit-learn>=0.24.0
tqdm>=4.62.0
deep-translator>=1.11.4
pyttsx3>=2.90
gTTS>=2.3.1
playsound>=1.2.2
torch>=1.9.0
transformers>=4.5.0
matplotlib>=3.4.0
seaborn>=0.11.0
gradio>=4.0.0
```

Figure 24

Conclusion

This project successfully demonstrates the practical application of course concepts in creating a useful tool for movie content analysis and accessibility. The implementation shows good understanding of machine learning principles and AI concepts.

The system achieves its core objectives of:

- Accurate genre classification
- Multilingual support
- Audio generation
- User-friendly interface

While maintaining high standards in:

- Code quality

- Error handling
- User experience
- System reliability