**MOVIE-SENTIMENT-ANALYSER**

Contents

[**INTRODUCTION** 2](#_Toc207533636)

[**PROJECT OVERVIEW** 2](#_Toc207533637)

[**KEY FEATURES** 2](#_Toc207533638)

[**ARCHITECTURE AND HOW IT WORKS** 2](#_Toc207533639)

[**INSTALLATION AND SETUP** 2](#_Toc207533640)

[**USAGE GUIDE** 2](#_Toc207533641)

[**CODE EXPLANATION** 2](#_Toc207533642)

[**LIMITATIONS AND FUTURE IMPROVEMENTS** 2](#_Toc207533643)

[**CONCLUSION** 2](#_Toc207533644)

# 

# **INTRODUCTION**

In this project, I developed a web-based application called Movie-Sentiment-Analyser to help users quickly understand the emotional tone of movie reviews. As a movie enthusiast, I often read reviews online and wanted a tool that could automatically classify them as positive, negative, or neutral, along with explanations and confidence scores. This app uses AI to make sentiment analysis easy and insightful, saving time for anyone deciding whether to watch a film.

The project is built using Python, Streamlit for the user interface, and Google's Gemini AI model for the core sentiment analysis. It's designed to be user-friendly, with professional styling and visualizations to make the results engaging. About LLMs (Large Language Models) and web app development, and it evolved into a polished tool that anyone can use.

# **PROJECT OVERVIEW**

The Movie-Sentiment-Analyser is a Streamlit app that takes a movie review as input and outputs a detailed sentiment analysis. It leverages the Gemini-2.0-flash model to process the text, providing not just a label (Positive, Negative, or Neutral) but also a confidence score, a brief explanation, and key evidence phrases from the review that influenced the decision.

The app includes a modern UI with custom CSS for styling, interactive elements like progress bars and charts, and session history to track past analyses. There's also a backend class for handling the AI logic, and some utility functions for batch processing reviews from CSV files. Overall, it's a full-stack mini-project that combines frontend design, AI integration, and data handling.

Technologies used:

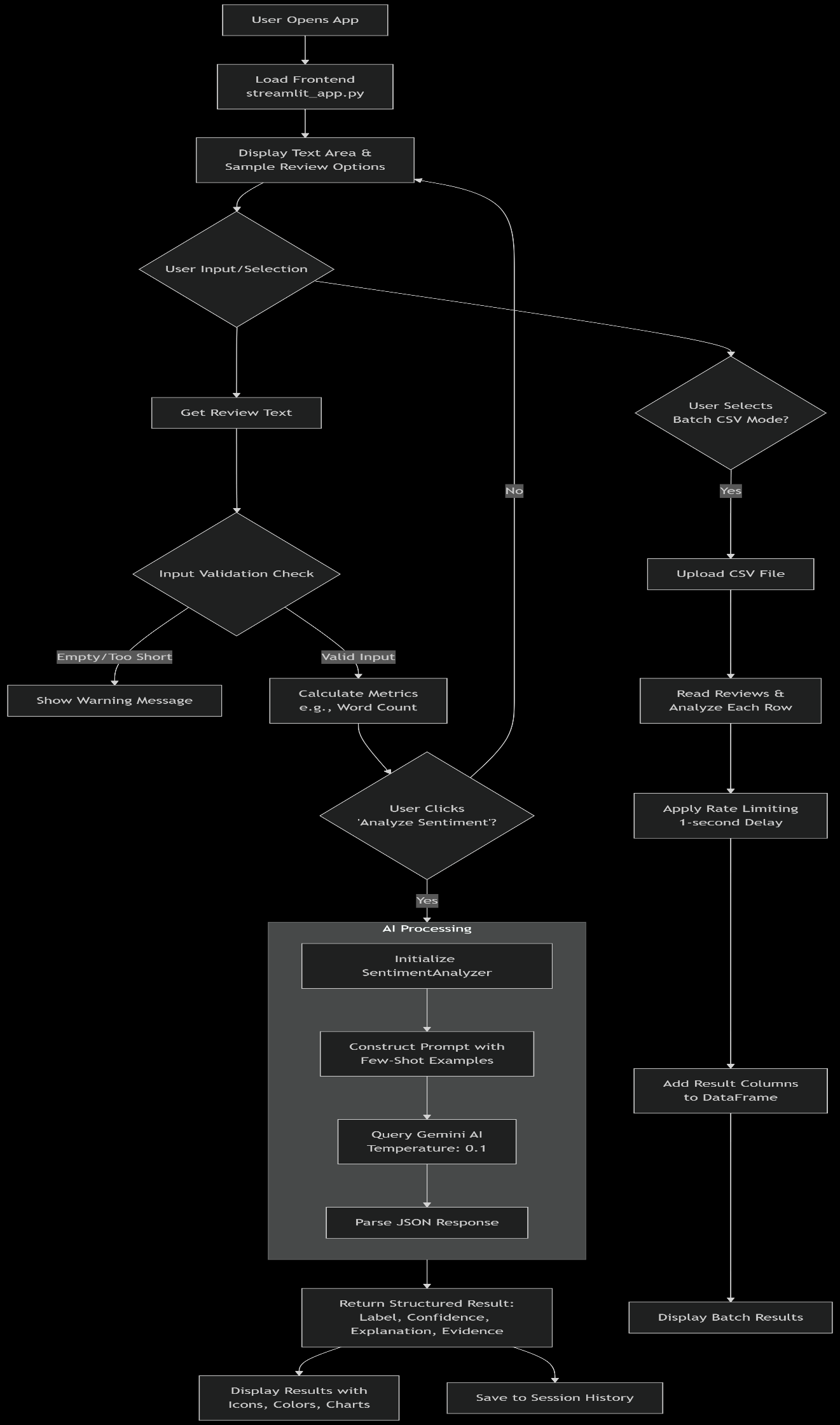
* **Python 3.x**: Core language.
* **Streamlit**: For building the interactive web app.
* **Google Generative AI (Gemini API)**: Gemini-2.0-flash model For sentiment analysis.
* **Pandas**: For batch processing CSV files.
* **Other libraries**: JSON for data handling, dotenv for API keys, and typing for type hints.

# **KEY FEATURES**

Here are the main features I implemented, based on what users might need:

* **Sentiment Classification**: Analyzes reviews and labels them as Positive, Negative, or Neutral.
* **Confidence Scoring**: Provides a score from 0 to 1 indicating how sure the AI is about the label.
* **Detailed Explanations**: Gives 1-2 sentences explaining the reasoning, plus 2-3 key phrases as evidence.
* **Interactive UI**: Custom-styled input area, sample reviews for quick testing, character/word counters, and animated result cards.
* **History Tracking**: Stores recent analyses in the sidebar for easy reference, with a clear history button.
* **Batch Processing**: A separate function to analyze multiple reviews from a CSV file and save results to a new CSV.
* **Error Handling**: Validates input (e.g., warns for short reviews) and gracefully handles empty or invalid inputs.
* **Sidebar Guidance**: Includes "How It Works" and "Pro Tips" sections to help users get the best results.

# **ARCHITECTURE AND HOW IT WORKS**



# **INSTALLATION AND SETUP**

To run this project on your machine, follow these steps:

1. **Clone the Repository**: If you have it on GitHub, use git clone <repo-url>. Otherwise, create a folder and add the code files.
2. **Install Dependencies**: Run pip install streamlit google-generativeai plotly pandas python-dotenv.
3. **Set Up API Key**: Create a .env file in the project root with GEMINI\_API\_KEY=your\_key\_here. Get a free key from Google's AI Studio.
4. **Run the App**: In your terminal, navigate to the project folder and run streamlit run streamlit\_app.py.
5. **For Batch Processing**: Use the functions in sentiment\_llm.py like process\_csv\_file('input.csv', 'output.csv').

Make sure you have Python 3.8+ installed. No other setup is needed—the app runs locally in your browser.

# **USAGE GUIDE**

Using the app is straightforward:

1. Open the app in your browser (it auto-launches after running the command).
2. Enter or select a movie review in the text area.
3. Click "Analyze Sentiment".
4. View the results: See the label, confidence gauge, explanation, and evidence chips.
5. Check the sidebar for history or tips.
6. For batch analysis: Prepare a CSV with a "review\_text" column, then call the processing function in a Python script.

Example Input: "This movie was absolutely amazing! Great acting and fantastic storyline." Expected Output: Positive label, high confidence (e.g., 0.95), explanation like "The review uses enthusiastic words praising the movie," and evidence like ["amazing", "great acting", "fantastic storyline"].

If something goes wrong (e.g., API error), the app shows a friendly error message.

# **CODE EXPLANATION**

The code is divided into logical parts. Here's a breakdown with snippets:

* **Streamlit App (streamlit\_app.py)**: Sets up the page config, CSS styles, and UI components. The main() function handles input, button clicks, and result display. I used Plotly for charts and custom HTML for styled elements like evidence chips.

Example Snippet:

def display\_sentiment\_result(result):

    """Display sentiment analysis results with enhanced visualization"""

    label = result["label"]

    confidence = result["confidence"]

    explanation = result["explanation"]

    evidence\_phrases = result.get("evidence\_phrases", [])

    # Define sentiment configurations

    sentiment\_config = {

        "Positive": {"color": "#10b981", "icon": "✅"},

        "Negative": {"color": "#ef4444", "icon": "❌"},

        "Neutral": {"color": "#f59e0b", "icon": "⚪"}

    }

    config = sentiment\_config.get(label, sentiment\_config["Neutral"])

    # Main results container

    with st.container():

        st.markdown('<div class="fade-in">', unsafe\_allow\_html=True)

        # Results header with two columns

        col1, col2 = st.columns([2, 1])

        with col1:

            # Use Streamlit native components for better reliability

            st.markdown(f"## {config['icon']} {label} Sentiment")

            # Confidence display using Streamlit's metric

            st.metric(

                label="Confidence Score",

                value=f"{confidence:.1%}",

                delta=None

            )

            # Visual progress bar

            st.progress(confidence)

            # Color-coded message based on sentiment

            if label == "Positive":

                st.success(f" Strong positive sentiment detected with {confidence:.1%} confidence!")

            elif label == "Negative":

                st.error(f" Strong negative sentiment detected with {confidence:.1%} confidence!")

            else:

                st.warning(f" Neutral sentiment detected with {confidence:.1%} confidence!")

        with col2:

            # Confidence gauge chart

            if confidence > 0:

                fig = create\_confidence\_chart(confidence, label)

                st.plotly\_chart(fig, use\_container\_width=True)

        st.markdown('</div>', unsafe\_allow\_html=True)

    # Explanation section

    st.markdown("### Analysis Explanation")

    st.info(f" {explanation}")

    # Evidence phrases

    if evidence\_phrases:

        st.markdown("### Key Evidence Phrases")

        st.markdown("\*The following phrases were identified as strong sentiment indicators:\*")

        # Display evidence chips

        evidence\_html = "".join([

            f'<span class="evidence-chip">{phrase}</span>'

            for phrase in evidence\_phrases

        ])

        st.markdown(f"""

        <div style="margin: 20px 0;">

            {evidence\_html}

        </div>

        """, unsafe\_allow\_html=True)

    # Metrics row

    col1, col2, col3 = st.columns(3)

    with col1:

        st.markdown(f"""

        <div class="metric-card">

            <div class="metric-value">{confidence:.1%}</div>

            <div class="metric-label">Confidence</div>

        </div>

        """, unsafe\_allow\_html=True)

    with col2:

        st.markdown(f"""

        <div class="metric-card">

            <div class="metric-value">{len(evidence\_phrases)}</div>

            <div class="metric-label">Key Phrases</div>

        </div>

        """, unsafe\_allow\_html=True)

    with col3:

        word\_count = len(result.get("original\_text", "").split()) if "original\_text" in result else 0

        st.markdown(f"""

        <div class="metric-card">

            <div class="metric-value">{word\_count}</div>

            <div class="metric-label">Words Analyzed</div>

        </div>

        """, unsafe\_allow\_html=True)

    # Raw JSON (collapsible)

    with st.expander("Raw JSON Output", expanded=False):

        st.json(result)

**SentimentAnalyzer Class (sentiment\_llm.py)**: Initializes Gemini, analyzes single reviews with prompts, and handles batches. It includes few-shot examples for better AI performance and error handling for invalid inputs.

Example Snippet:

def analyze\_sentiment(self, review\_text: str) -> Dict:

        """

        Analyze sentiment of a movie review using Gemini LLM

        """

        # Input validation

        if not review\_text or not isinstance(review\_text, str) or review\_text.strip() == "":

            return {

                "label": "Neutral",

                "confidence": 0.0,

                "explanation": "Invalid input: empty or non-string review",

                "evidence\_phrases": []

            }

        prompt = f"""

        Analyze the sentiment of this movie review and provide:

        1. Sentiment label: Positive, Negative, or Neutral

        2. Confidence score between 0-1

        3. Brief explanation (1-2 sentences)

        4. 2-3 key evidence phrases from the text that support your analysis

        {self.few\_shot\_examples}

        Review: "{review\_text}"

        Return ONLY a valid JSON object with this exact structure:

        {{

            "label": "Positive|Negative|Neutral",

            "confidence": 0.00,

            "explanation": "short reason",

            "evidence\_phrases": ["phrase1", "phrase2"]

        }}

        """

        try:

            response = self.model.generate\_content(

                prompt,

                generation\_config=genai.types.GenerationConfig(

                    temperature=self.temperature,

                    max\_output\_tokens=200

                )

            )

            # Extract JSON from response

            result\_text = response.text.strip()

            # Clean the response

            if result\_text.startswith('```json'):

                result\_text = result\_text[7:-3]  # Remove ```json and ```

            elif result\_text.startswith('```'):

                result\_text = result\_text[3:-3]  # Remove ``` and ```

            result = json.loads(result\_text)

            # Validate the response structure

            required\_keys = ["label", "confidence", "explanation", "evidence\_phrases"]

            if not all(key in result for key in required\_keys):

                raise ValueError("Invalid response format from LLM")

            return result

        except json.JSONDecodeError as e:

            print(f"JSON parsing error: {e}")

            print(f"Raw response: {response.text if 'response' in locals() else 'No response'}")

            return {

                "label": "Neutral",

                "confidence": 0.0,

                "explanation": "JSON parsing error in analysis",

                "evidence\_phrases": []

            }

        except Exception as e:

            print(f"Error analyzing sentiment: {e}")

            return {

                "label": "Neutral",

                "confidence": 0.0,

                "explanation": "Error in analysis",

                "evidence\_phrases": []

            }

# **LIMITATIONS AND FUTURE IMPROVEMENTS**

While the app works well, it has some limitations: It relies on the Gemini API, so internet and API quotas are needed. Short reviews might give lower confidence, and sarcastic text could confuse the AI. Batch processing is sequential and slow for large files.

Future ideas: Integrate more AI models (e.g., Hugging Face), add real-time feedback, support uploading CSV directly in the app, or deploy to the cloud (e.g., Streamlit Sharing).

# **CONCLUSION**

Building the Movie-Sentiment-Analyser was a rewarding experience that taught me about AI integration, UI design, and handling real-world data. It's a practical tool for movie lovers and demonstrates how LLMs can make everyday tasks smarter. I hope this documentation helps others understand and use it—feel free to fork or contribute! If you have feedback, reach out.

**Project created by V Bhuvaneswari, August 2025.**