**News Summarization and Text-to-Speech Application Report**

**Project Overview**

The News Summarization and Text-to-Speech Application is a web-based tool designed to fetch news articles about a specified company, summarize their content, analyze sentiment, perform a comparative analysis, and generate Hindi audio summaries. Built with Python, it integrates a Streamlit frontend for user interaction and leverages external APIs and NLP techniques to process news data. This report details the technologies used, their purposes, and the application's functionality as implemented in the provided code.

**Implementation Details**

**Technologies and Tools Used**

**1. News Fetching (api.py)**

* **Library**: requests
  + **Purpose**: Used to make HTTP GET requests to the NewsAPI (https://newsapi.org/v2/everything) to retrieve news articles based on a company name query.
  + **Details**: The function get\_articles\_api() constructs a URL with an API key (4a925976ff6041f8abf459a4241556ae), queries for English articles sorted by publication date, and limits the output to 10 articles. It handles errors (e.g., network issues, invalid responses) and returns a structured JSON-like dictionary with article titles, URLs, and content.
* **External API**: NewsAPI
  + **Purpose**: Provides a reliable source of recent news articles, replacing manual web scraping for efficiency and compliance with terms of service.
  + **Details**: Returns up to 100 articles per request, filtered to 10 in the code, with fields like title, url, and content.

**2. Web Interface (Streamlit App)**

* **Library**: streamlit
  + **Purpose**: Creates an interactive web interface where users input a company name and view results, including article summaries, sentiment analysis, and Hindi audio playback.
  + **Details**: The main() function in the Streamlit script (app.py) defines the UI with text input, buttons, headers, expanders, sliders, and audio players. It integrates with backend functions to display results dynamically.

**3. Sentiment Analysis (model.py)**

* **Library**: nltk with VADER (Valence Aware Dictionary and sEntiment Reasoner)
  + **Purpose**: Analyzes the sentiment of article content, classifying it as Positive, Negative, or Neutral with a compound score.
  + **Details**: The analyze\_sentiment() function uses VADER, a lexicon-based sentiment analyzer, downloaded via nltk.download('vader\_lexicon'). It computes polarity scores and applies thresholds (≥ 0.05 for Positive, ≤ -0.05 for Negative, else Neutral) to categorize sentiment.

**4. Summarization and Text Processing (utils.py)**

* **Library**: nltk with sumy
  + **Purpose**: Summarizes article content to a concise form and a 500-word version for detailed output.
  + **Details**:
    - summarize\_text() uses sumy’s LSA (Latent Semantic Analysis) summarizer to produce a 3-sentence summary.
    - summarize\_text\_500\_words() generates a 500-word summary by extending the LSA output with original text if needed.
    - NLTK’s punkt tokenizer, downloaded via nltk.download('punkt') and nltk.download('punkt\_tab'), supports text parsing.
* **Library**: requests and BeautifulSoup
  + **Purpose**: Alternative web scraping method in scrape\_articles() to fetch articles from search results if NewsAPI is unavailable.
  + **Details**: Scrapes content from Google search results (using googlesearch-python) with a custom query, filtering for relevance (e.g., company mentions).

**5. Text-to-Speech (utils.py)**

* **Library**: gtts (Google Text-to-Speech)
  + **Purpose**: Converts text summaries into Hindi audio files for playback.
  + **Details**:
    - text\_to\_speech\_hindi() generates full audio from translated text.
    - text\_to\_speech\_hindi\_limited() limits input to 700 words per article for efficiency, saving files as summary\_{index}.mp3.
* **Library**: deep\_translator with GoogleTranslator
  + **Purpose**: Translates English summaries to Hindi before TTS conversion.
  + **Details**: Uses Google’s translation service via GoogleTranslator(source='en', target='hi') to ensure Hindi audio is meaningful.

**6. Comparative Analysis (Streamlit App)**

* **Custom Function**: comparative\_analysis()
  + **Purpose**: Aggregates sentiment across articles to provide an overall sentiment score and Hindi summary.
  + **Details**: Calculates sentiment distribution, computes a weighted score (Positive: 100, Neutral: 50, Negative: 0), and determines the dominant sentiment. Outputs a Hindi summary for TTS.

**Functionality and Output**

The application operates as follows:

* **Input**: Users enter a company name (e.g., "Tesla") via the Streamlit UI.
* **Article Fetching**: get\_articles\_api() retrieves 10 articles from NewsAPI, falling back to scrape\_articles() if needed (though the provided code prioritizes NewsAPI).
* **Processing**: Each article’s content is summarized (summarize\_text\_500\_words()), analyzed for sentiment (analyze\_sentiment()), and converted to Hindi audio (text\_to\_speech\_hindi\_limited()).
* **Display**: The UI shows:
  + Article titles, URLs, full content (in expanders), 500-word summaries, sentiment labels with scores (visualized via sliders with emojis), and individual Hindi audio files.
  + A comparative analysis section with overall sentiment, a score slider, and a Hindi audio summary.
* **Error Handling**: Displays errors for invalid inputs, API failures, or processing issues using Streamlit’s st.error() and st.warning().

**Technical Considerations**

**Why These Tools Were Used**

* **NewsAPI**: Ensures legal, reliable article sourcing over manual scraping, with structured data output.
* **Streamlit**: Simplifies UI development with real-time interactivity, ideal for rapid prototyping.
* **VADER**: Chosen for its simplicity and effectiveness on short news texts without requiring model training.
* **Sumy**: Provides lightweight summarization compared to transformer models, balancing performance and resource use.
* **gTTS and deep\_translator**: Enable Hindi audio output with minimal setup, leveraging Google’s robust services.

**Conclusion:**

The application successfully meets its objectives, leveraging NewsAPI for article retrieval, VADER and Sumy for NLP tasks, and gTTS with GoogleTranslator for Hindi audio output. The Streamlit interface provides an engaging user experience, while the code’s modularity supports future enhancements (e.g., caching, advanced models). Limitations in content depth and performance are offset by the tool’s simplicity and functionality, making it a practical solution for news summarization and sentiment analysis.