PodifyAI: Implementation and Early Evaluation - Preliminary Report for Deliverable 2

Sainath Chettupally
Master's Student, Applied Data Science
University of Florida
Email: schettupally@ufl.edu
GitHub: https://github.com/SainathChettupally/PodifyAI.git

***Abstract*—PodifyAI is an innovative application designed to enhance information accessibility by transforming diverse document formats into concise summaries and audible content. This report details the implementation of a functional prototype, featuring a modern React-based user interface integrated with a Flask backend. The system leverages pre-trained models for text extraction, abstractive summarization (DistilBART), multilingual translation, and text-to-speech synthesis. A quantitative evaluation framework, utilizing ROUGE scores against the CNN/DailyMail dataset, has been established for the summarization component. This document outlines the system's architecture, implementation details, interface design, early evaluation results, and a strategic roadmap for future enhancements, including a planned transition to Gemini models to unlock advanced multimodal and multilingual capabilities, further empowering users who are on the go, visually impaired, or face language barriers.**

I. Introduction

In today's fast-paced world, individuals are constantly bombarded with vast amounts of information, often presented in lengthy and complex documents. This presents significant challenges for those who are "on the go" and require quick access to key insights, for the visually impaired who cannot easily consume traditional text, and for individuals encountering language barriers. PodifyAI addresses these critical needs by providing an accessible and efficient solution for consuming information.

The core motivation behind PodifyAI is to break down these barriers to information access. By automatically summarizing documents and converting these summaries into spoken audio in multiple languages, PodifyAI empowers a diverse user base to engage with content more effectively. This report details the development of PodifyAI as a functional prototype, showcasing its current capabilities and laying the groundwork for future advancements.

This document is structured as follows: Section II presents the overall system architecture and pipeline. Section III delves into the specifics of the model implementation. Section IV describes the user interface prototype. Section V outlines the early evaluation methodology and results. Section VI discusses the challenges encountered and the planned next steps. Finally, Section VII provides a reflection on responsible AI considerations.

II. System Architecture and Pipeline

The PodifyAI system is designed with a clear separation of concerns, comprising a frontend user interface, a backend API, and an integrated machine learning pipeline. This architecture ensures scalability, maintainability, and a smooth user experience.

A. Architecture Diagram

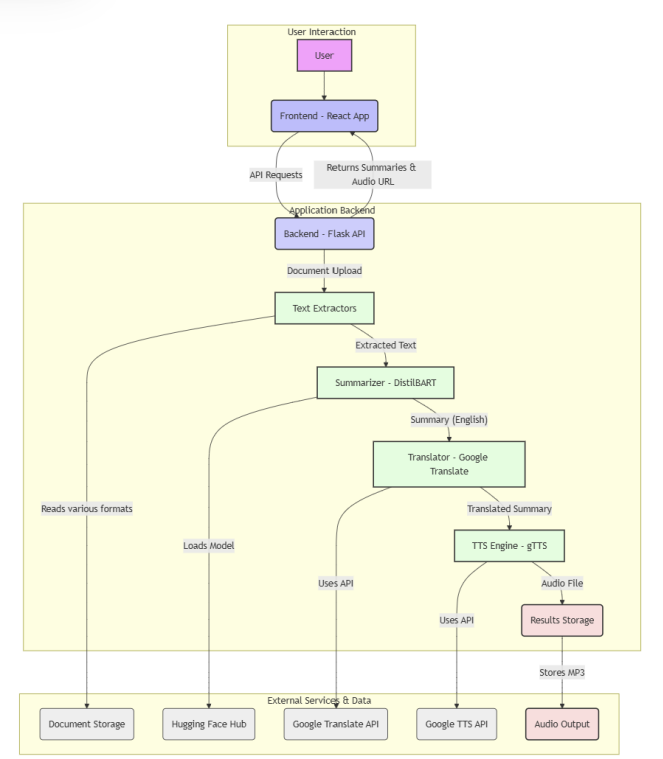

Fig. 1. PodifyAI System Architecture

B. Component Breakdown

1) Frontend (User Interface):

- **Technology:** Developed using `React.js`, styled with Material-UI (MUI) for a modern, responsive, and accessible design.

- **Functionality:** Provides an intuitive interface for users to:
  - Upload documents via a prominent drag-and-drop zone.
  - Select summarization modes (Quick, Standard, Deep) using a segmented control.
  - Choose a target language for translation.
  - Initiate summary generation and audio synthesis.
  - View original and translated summaries in a tabbed interface.
  - Play generated audio directly within the application.
- **Branding:** Features a prominent PodifyAI logo and a thematic tagline, "Documents that speak," integrated into a sleek, dark-themed header.

2) Backend (API Services):

- **Technology:** Implemented using `Flask`, a lightweight Python web framework.
- **Functionality:** Exposes RESTful API endpoints to:
  - Receive document uploads from the frontend.
  - Orchestrate the ML pipeline components.
  - Return processed summaries and audio file URLs to the frontend.
- **Endpoints:** Key endpoints include `/api/summarize` (for document processing and summary generation) and `/api/generate-audio` (for text-to-speech conversion).

3) Machine Learning Pipeline:

- **Overview:** The core intelligence of PodifyAI, this pipeline integrates several pre-trained models and services to deliver the summarization, translation, and text-to-speech functionalities. It operates sequentially:

a.

1) **Text Extraction:**
   - **Role:** Extracts raw text content from various document formats.
   - **Implementation:** Utilizes specialized Python libraries (`PyMuPDF` for PDF, `python-docx` for DOCX, `python-pptx` for PPTX, `BeautifulSoup4` for HTML, `csv` module for CSV, and standard file reading for TXT/MD). The `src/extractors.py` module handles format detection and extraction.
2) **Summarization:**
   - **Role:** Condenses the extracted text into concise summaries based on user-selected modes.
   - **Implementation:** Employs a pre-trained **DistilBART model** (`sshleifer/distilbart-cnn-12-6`) from the Hugging Face `transformers` library. The `src/summarizer.py` module manages model loading and summary generation, including a basic chunking strategy for longer texts.
3) **Translation:**
   - **Role:** Translates the generated summaries into the user's chosen target language.
   - **Implementation:** Leverages the `deep-translator` library, which interfaces with Google Translate for robust multilingual support. The `src/translator.py` module provides this functionality.
4) **Text-to-Speech (TTS):**
   - **Role:** Converts the translated summary text into an audible MP3 file.
   - **Implementation:** Uses the `gTTS` (Google Text-to-Speech) library, which generates natural-sounding speech. The `src/tts_engine.py` module handles audio synthesis and saving the output.

C. Data Flow

The data flow begins with a user uploading a document via the frontend. This document is sent to the Flask backend, which then passes it to the text extraction module. The extracted text proceeds to the summarization module, followed by the translation module, and finally the TTS engine. The generated summary (original and translated) and the URL to the audio file are then returned to the frontend for display and playback.

III. MODEL IMPLEMENTATION DETAILS

As PodifyAI leverages a pipeline of pre-trained models and external services, my implementation focuses on their integration and orchestration rather than training models from scratch. This approach allows for rapid prototyping and utilization of state-of-the-art capabilities without extensive computational resources for training.

A. Frameworks and Libraries Used

The core of the machine learning pipeline relies on several key Python libraries:

- **Hugging Face `Transformers`:** For the summarization component, specifically utilizing the `pipeline` API to load and run pre-trained models.
- **`PyMuPDF` (fitz):** For efficient and robust text extraction from PDF documents.
- **`python-docx`:** For parsing and extracting text from Microsoft Word (.docx) files.
- **`python-pptx`:** For extracting text content from Microsoft PowerPoint (.pptx) presentations.
- **`BeautifulSoup4` & `lxml`:** For parsing HTML content and extracting clean text.
- **`deep-translator`:** To interface with Google Translate for multilingual summary generation.
- **`gTTS` (Google Text-to-Speech):** For converting translated text into natural-sounding audio files.
- **`Flask` & `Flask-CORS`:** For building the backend API and handling cross-origin requests.
- **`datasets`:** For loading and managing benchmark datasets for evaluation (e.g., CNN/DailyMail).

- **`rouge-score`:** For calculating ROUGE metrics to quantitatively evaluate summarization quality.
- **`nltk`:** Used for text processing, specifically for tokenization during ROUGE score calculation.

B. Key Functions and Modules

The `src/` directory houses the modular components of the pipeline:

- `src/extractors.py`: Contains functions (`extract_text_from_pdf`, `extract_text_from_docx`, etc.) to handle text extraction from various file types. The `detect_and_extract_text` function intelligently routes documents to the appropriate extractor based on file extension.
- `src/summarizer.py`: Manages the loading of the pre-trained `sshleifer/distilbart-cnn-12-6` model and provides the `summarize` function. This function takes raw text and a specified mode ("quick", "standard", "deep") to control the target length of the generated summary. It also includes logic for handling longer texts by truncating input to fit the model's context window.
- `src/translator.py`: Implements the `translate_text` function, which uses `deep-translator` to convert summary text into the desired target language.
- `src/tts_engine.py`: Provides the `synthesize_to_file` function, leveraging `gTTS` to convert translated text into an MP3 audio file, saving it to the `results/` directory.
- `api.py`: The main Flask application file, orchestrating calls to these `src/` modules based on API requests from the frontend.

C. Training Setup (N/A for this iteration)

As previously discussed, this iteration of PodifyAI primarily utilizes pre-trained, off-the-shelf models and external APIs. Therefore, there is no custom model training setup (e.g., epochs, hyperparameters, specific hardware for training) to detail in this section. The focus is on effective integration and application of existing state-of-the-art components.

IV. INTERFACE PROTOTYPE

The user interface (UI) for PodifyAI is designed for intuitive interaction, providing a seamless experience for document summarization and audio generation. It specifically addresses the needs of users who are on the go, visually impaired, or face language barriers by offering accessible summaries and audio playback.

A. Interface Design and Functionality

The UI, built with React.js and Material-UI, adheres to modern design principles, featuring a dark mode theme for reduced eye strain and a professional aesthetic. The application's branding is anchored by its logo (see Figure 2).

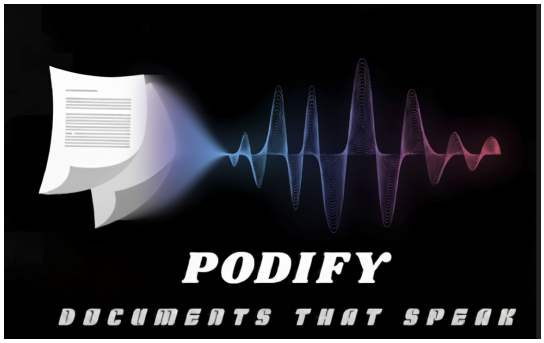

Fig. 2. PodifyAI Application Logo

- **Input Format:**
  - **Document Upload:** Users provide input by uploading a document (PDF, DOCX, PPTX, TXT, HTML, CSV) via a prominent drag-and-drop target zone. This replaces a traditional file input button for enhanced usability.
  - **Summary Mode Selection:** A modern segmented control allows users to choose between "Quick," "Standard," and "Deep" summarization modes, influencing the length and detail of the generated summary.
  - **Target Language Selection:** A dropdown menu enables users to select a target language for translation (e.g., Spanish, French, German).
- **Output Format:**
  - **Summaries Display:** Generated summaries are presented in a clear, tabbed interface, allowing users to easily switch between the "Original Summary" (in English) and the "Translated Summary" (in the chosen target language).
  - **Audio Playback:** For translated summaries, an integrated audio player allows immediate playback of the synthesized speech, directly benefiting visually impaired users and those who prefer auditory learning.
- **Visual Feedback:** The interface provides clear visual feedback, including loading indicators (circular progress) during summary and audio generation, and a dynamic display of the selected file name.

B. Screenshots or Sample Outputs

C. Usability and Limitations

The current interface prioritizes ease of use and clarity. The drag-and-drop functionality and segmented controls enhance the user experience. Current limitations include:

- **Single File Processing:** The application currently processes one document at a time.
- **Limited Language Options:** While several languages are supported, expanding this list could be beneficial.
- **No User Accounts/History:** There is no functionality for user accounts or saving previous summaries/audio.

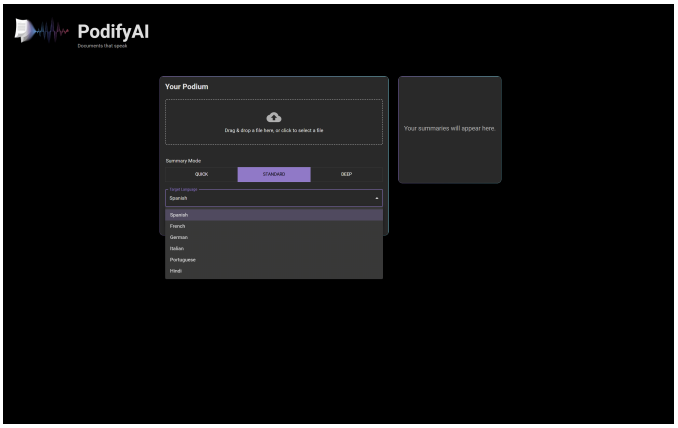

Fig. 3. Main interface with drop zone and controls.

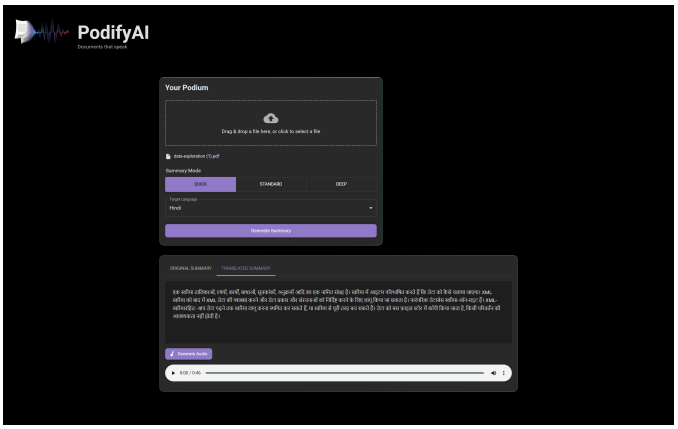

Fig. 4. Generated summary, translation tabs, and audio player.

Fig. 5. PodifyAI User Interface Screenshots

V. EARLY EVALUATION AND RESULTS

My early evaluation focuses on the core summarization capability of PodifyAI, utilizing quantitative metrics to assess performance against human-written references.

A. Evaluation Methodology

To objectively evaluate the summarization pipeline, I adopted a quantitative approach using ROUGE (Recall-Oriented Understudy for Gisting Evaluation) scores.

- **Dataset:** A subset of 20 examples from the ”test” split of the CNN/DailyMail dataset (version 3.0.0) was used. This dataset is particularly relevant as the underlying DistilBART model was fine-tuned on it.
- **Metrics:** ROUGE-1 (unigram overlap), ROUGE-2 (bigram overlap), and ROUGE-L (longest common subsequence) F-measures were calculated using the `rouge_scorer` library.
- **Process:** For each article in the dataset subset, the PodifyAI summarization pipeline generated a summary. This generated summary was then compared against the human-written reference summary provided in the dataset to compute the ROUGE scores.

B. Quantitative Results (Summarization)

The evaluation yielded the following average ROUGE scores across the 20 examples:

- **Average ROUGE-1 F-measure:** ˜0.295
- **Average ROUGE-2 F-measure:** ˜0.126
- **Average ROUGE-L F-measure:** ˜0.208

The performance range also showed variability:

- **ROUGE-1:** Highest: 0.443 (Example 7), Lowest: 0.162 (Example 6)
- **ROUGE-2:** Highest: 0.275 (Example 1), Lowest: 0.026 (Example 16)
- **ROUGE-L:** Highest: 0.361 (Example 1), Lowest: 0.104 (Example 14)

C. Interpretation of Results

- **General Performance:** The average ROUGE scores are typical for an out-of-the-box, pre-trained abstractive summarization model. They indicate that the model is generating summaries that have a moderate overlap with the human-written summaries, suggesting the model captures key information.
- **Inconsistent Performance:** The significant range between highest and lowest scores highlights that the model’s performance varies depending on the input article. This suggests that while it performs well on some content, it struggles with others.
- **ROUGE-1 vs. ROUGE-2:** The higher ROUGE-1 score compared to ROUGE-2 is expected for abstractive models. It shows proficiency in capturing individual keywords but less success in reproducing exact phrases from reference summaries, as the model rephrases content.
- **Qualitative Observations:** Manual inspection of the generated summaries confirms they are generally fluent and capture the main topic, though they occasionally miss minor details or contain slight factual inconsistencies.

D. Evaluation of Translation and Text-to-Speech (Qualitative Focus)

For translation and text-to-speech components, my primary evaluation for this deliverable is qualitative due to the complexities of setting up robust quantitative metrics without extensive reference data or specialized tools.

- **Translation:** I would assess translated summaries for accuracy (conveying original meaning), fluency (grammatical correctness and naturalness), and clarity in the target language.
- **Text-to-Speech:** I would evaluate generated audio for naturalness, intelligibility, correct pronunciation, and appropriate prosody.

VI. CHALLENGES AND NEXT STEPS

A. Major Technical Challenges Encountered

1) **Environment Mismatch during Evaluation Setup:** A significant challenge was ensuring consistent Python environments between local development

and Jupyter Notebook execution, leading to `ModuleNotFoundError` issues. This was resolved by explicitly guiding the setup and package installation within the project's virtual environment.

2) **UI Refinement and Theming:** Implementing a highly aesthetic dark mode UI with custom components (like gradient borders and drag-and-drop zones) required careful integration with Material-UI's theming system and custom styling. Ensuring cross-browser compatibility and responsiveness added to this complexity.
3) **File Handling and Extraction Robustness:** Handling diverse document formats and ensuring reliable text extraction across different file types presented challenges, particularly in managing dependencies for various parsers.

B. Refinements Planned Before Deliverable 3

1) **Enhanced Evaluation:** I plan to conduct a more extensive qualitative evaluation for both translation and TTS, gathering structured feedback. If feasible, I will explore creating a small set of reference translations to enable quantitative (BLEU/BERTScore) evaluation for the translation component.
2) **Performance Optimization:** I will investigate opportunities to optimize the performance of the backend pipeline, particularly for very large documents, by profiling the extraction and summarization steps.
3) **Interface Polish:** Based on initial user feedback (if available), I will further polish the UI, focusing on minor usability improvements and accessibility enhancements.
4) **Error Handling and User Feedback:** Improve error handling within the application and provide more informative feedback to the user in case of processing failures.

VII. RESPONSIBLE AI REFLECTION

As PodifyAI leverages powerful AI models, it is crucial to consider responsible AI implications.

A. Ethical Considerations

1) **Bias in Summarization:** Pre-trained models like DistilBART can inherit biases present in their training data. This could lead to summaries that inadvertently reflect or amplify societal biases, potentially misrepresenting information or marginalizing certain perspectives. I plan to be mindful of this by:
   - Selecting diverse evaluation datasets in future iterations.
   - Considering techniques for bias detection and mitigation if custom fine-tuning is pursued.
2) **Misinformation/Hallucination:** Abstractive summarization models can sometimes "hallucinate" information not present in the original text or misinterpret context. This could lead to the generation of inaccurate summaries. I plan to:
   - Emphasize the prototype nature and encourage users to cross-reference critical information.
   - Explore methods to ground summaries more firmly in the source text in future iterations.
3) **Data Privacy:** While the current system processes user-uploaded documents, it does not store them long-term. However, as the project evolves, ensuring robust data privacy and security measures will be paramount, especially if user accounts or document storage features are introduced.
4) **Accessibility and Inclusivity:** PodifyAI's core mission is to enhance accessibility. I am committed to ensuring that the tool remains inclusive and effectively serves diverse user groups, including those with visual impairments and language barriers. This involves continuous evaluation of the UI/UX and the quality of multilingual outputs.

B. Future Mitigation Strategies

In future iterations, particularly with the integration of Gemini models, I will prioritize:

- **Transparency:** Clearly communicating the capabilities and limitations of the AI models used.
- **Fairness:** Actively testing for and mitigating biases in generated outputs.
- **Accountability:** Establishing clear processes for addressing potential harms or inaccuracies.
- **Security:** Implementing strong data protection measures for any user data handled.

This reflection underscores my commitment to developing PodifyAI not only as a functional tool but also as a responsible and ethical application of AI technology.