

Multilingual Translator & Text-to-Speech Application for English Input

GitHub repo : https://github.com/mcadriaans/genai_translate-and-speak

Application Live Demo: [Google Gemini Translator & Text-to-Speech](#)

Introduction

This documentation presents a comprehensive overview of the project, outlining its primary goals, system architecture, and implementation details. It covers the full development lifecycle: from design and testing to deployment on Streamlit Community Cloud. It also highlights key technical challenges and the innovative solutions that address them.

The aim is to provide a clear understanding of the application's capabilities, its underlying technologies, and the strategic decisions that shaped its successful delivery.

1. Project Overview

This document is a user-friendly web application designed for English to multilingual translation as well as text-to-speech conversion. Built with Streamlit, this application integrates Google's Gemini API for accurate translation and the **gTTS** (Google Text-to-Speech) library to generate downloadable audio files.

Users have the option to either manually input text or upload documents in specific formats (PDF, TXT, CSV, XLSX, PNG, JPG, JPEG) , select a target language, and receive both translated text and spoken output.

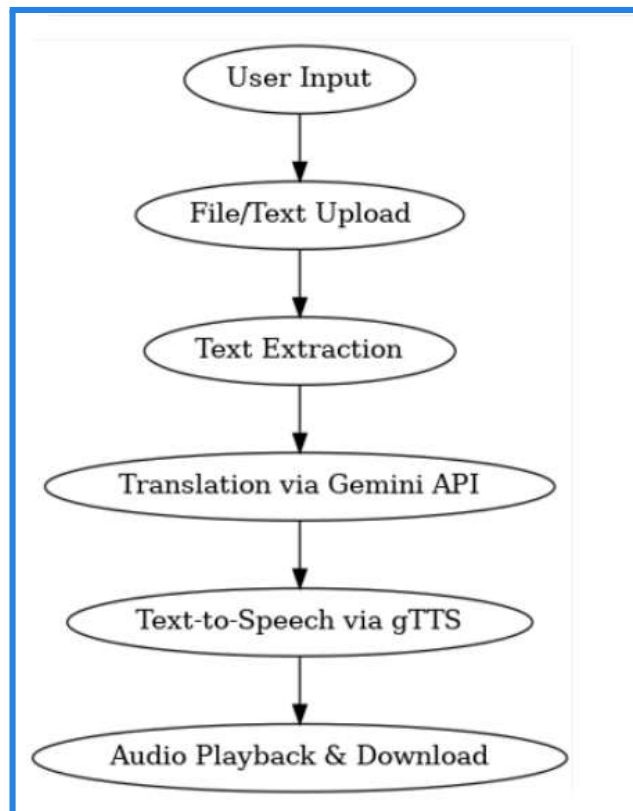


Figure 1: Application Workflow diagram

2. Design Decision : Input Language Restriction

While the application leverages the powerful multilingual capabilities of the Gemini API for target language translation, a deliberate design decision was made to restrict user input to English only. This choice was primarily driven by the following considerations:

- **Managing Complexity:** Accepting input in any language would significantly increase the difficulty of building and maintaining the system. It would require extensive exception handling to deal with unpredictable inputs, making testing and debugging far more complicated. By limiting input to English, the system remains easier to manage and behaves more consistently.
- **Language Detection Challenges:** Although Gemini can usually recognize the language of the input, it doesn't always get it right. For example, if someone writes in a dialect, mixes languages, or uses phrasing that could belong to more than one language, Gemini might misidentify it.

This inconsistency can lead to errors in how the system responds or translates, which is why relying on automatic language detection across all possible inputs can be risky or unreliable.

- **Consistent Source Quality:** Limiting input to English ensures a stable and uniform starting point for translation. This allows the project to focus on evaluating the accuracy and fluency of the Gemini API's output, without being affected by the variability and complexity of handling multiple source languages.

3. Technology Stack

The application leverages a modern Python-based technology stack to deliver its core functionalities:

Category	Tool/Library	Purpose/Description
Web Application Framework	Streamlit	Builds the interactive web interface and handles deployment.
AI/Translation	google.generativeai	Integrates with Google's Gemini 1.5 Flash API for advanced text translation.
Text-to-Speech	gTTS	Converts translated text into high-quality .mp3 audio for playback and download.
File Handling & OCR	PyPDF2, pdf2image, easyocr	Extracts text from diverse document types, including OCR for images.
Data Processing	pandas, openpyxl	Reads and processes structured data from CSV and Excel files.
Environment Management	python-dotenv	Securely manages API keys and environment variables.
Language Detection	langdetect	Detects the input language for validation.
Temporary File Management	tempfile	Handles creation and cleanup of temporary files (e.g., audio output).

4. Setup Instructions

Follow these steps to set up and run the application locally.

4.1 Prerequisites

- Python 3.10
- pip (Python package installer)
- **Poppler Utilities:** Essential for PDF-to-image conversion used in OCR.
 - macOS (Homebrew): ``brew install poppler``
 - Linux (apt-get):** ``sudo apt-get install poppler-utils``
 - Windows:** Download a compiled version (e.g., from <https://poppler.freedesktop.org/>). Extract the files and ensure the ``bin`` directory (containing ``pdftocairo.exe``) is added to your system's PATH environment variable.

4.2 Clone the repository

Navigate to the project folder on your local machine:

```
<> CMD , Powershell, bash  
  
git clone https://github.com/mcadriaans/translate-and-speak.git
```

4.3 Create and Activate a Virtual Environment

```
<> PowerShell  
  
python -m venv venv  
.\venv\Scripts\Activate.ps1
```

```
<> PowerShell  
  
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

<> Bash

```
python -m venv venv  
source venv/bin/activate
```

This will activate the virtual environment and we are now ready for development.

4.4 Install Dependencies

<> PowerShell

```
pip install -r requirements.txt
```

<> Bash

```
pip install -r requirements.txt
```

4.5 Configure Your Gemini API Key

1. Generate a key from [Google AI Studio](#).
2. Create a file named `.env` in the root of your project directory (`translate-and-speak/`).
3. Add your API key to this file in the format:

```
GOOGLE_API_KEY = your_api_key_here
```

(Replace `your_api_key_here` with your actual key.)

4. The `python-dotenv` package will securely load this key into the application..

4.6 Launch the application

<> PowerShell

```
Streamlit run translator_speak_app.py
```

<> Bash

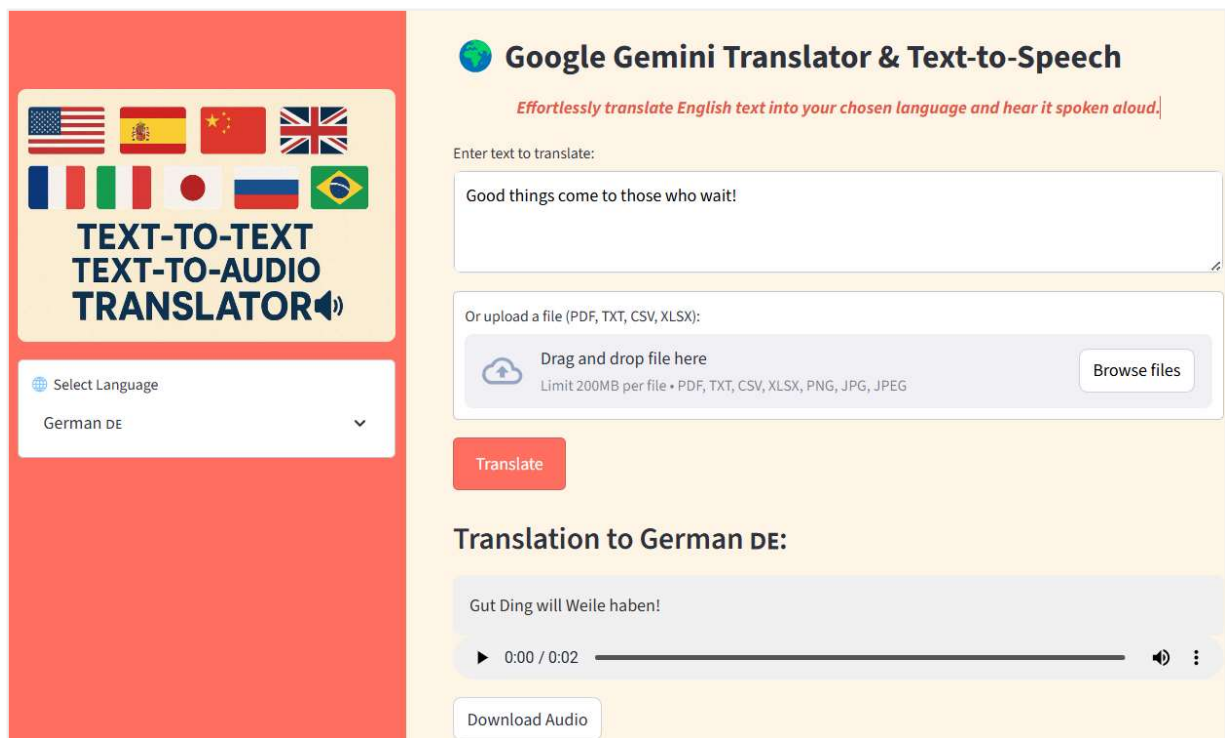
```
streamlit run translator_speak_app.py
```

Powershell will launch a local server and the default browser will open the application:

You can now view your Streamlit app in your browser.

Local URL: <http://localhost:8501>

Network URL: <http://192.168.0.231:8501>



5. Key Features

5.1 Elegant and Responsive Interface

- Custom styling with warm tones and an intuitive layout.

- Sidebar featuring a **Language Selection drop down menu** for choosing the target translation language.

5.2 Dual Input Options

- **Text Box:** Users can directly type or paste English text for translation.
- **File Upload :** Supports dragging and dropping or browsing files in formats like PDF, TXT, CSV, XLSX, PNG, JPG, and JPEG for automatic text extraction and translation.

5.3 Smart File Parsing

The application employs a robust text extraction pipeline to handle diverse input formats, intelligently determining the best method to retrieve content for translation. This includes direct text extraction from readable documents and an OCR fallback for image-based content.

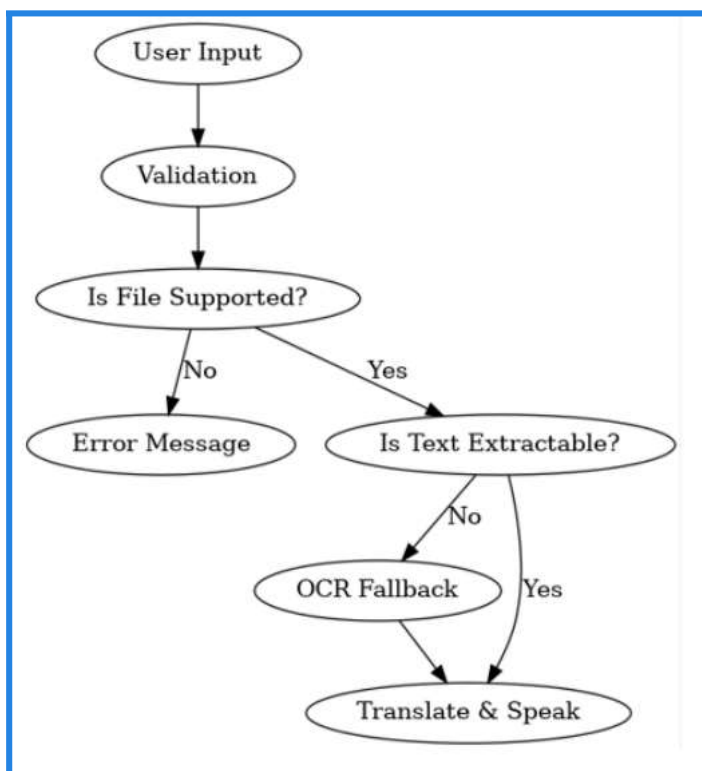


Figure 2: Smart Flow Parsing Workflow

This workflow illustrates the sequence:

1. **User Input & Validation:** The system first processes any direct text or uploaded file. It performs basic validation checks for empty input.

2. It then determines if the uploaded file's format (PDF, TXT, CSV, XLSX, PNG, JPG, JPEG) is supported. Unsupported files lead to an error.
3. For supported files, the system attempts to directly extract text (e.g., from text-based PDFs, TXT, CSV, XLSX).
4. If direct text extraction yields no content (e.g., image-only PDFs, pure image files), an OCR process is initiated using **EasyOCR** to convert visual text into readable data.
5. Once text is successfully extracted (either directly or via OCR), it proceeds to the translation and text-to-speech modules.

5.4 Multilingual Translation Powered by Gemini

- Users can choose from over 20+ languages via a flag-enhanced dropdown menu.
- Translates English text into selected language with high accuracy and natural phrasing leveraging Google's Gemini 1.5 Flash API.


5.5 Instant Text-to-Speech

- Translated text can be heard spoken aloud using realistic voice synthesis powered by **gTTS**.
- Includes a built-in audio playback bar and a **Download Audio button** for saving the **.mp3** file.





5.6 Robust Error Handling and User Feedback

In this user-facing application which involves file uploads, OCR, translation, and audio generation, clear communication is key. Streamlit's built-in messaging functions therefore play a vital role in guiding users, managing expectations, and gracefully handling errors.

Here's an overview of how the application leverages these messages to foster a seamless and reliable user experience while encouraging confidence and ease of interaction:

Message	Type of Message	Purpose	Usefulness
Initializing EasyOCR reader. Loading models from local storage (this happens only once per deployment).	 Information	Informs user that OCR setup is in progress	Sets expectations for delay and reassures it's a one-time setup.
Message	Type of Message	Purpose	Usefulness

EasyOCR reader initialized!	✓ Success	Confirms OCR engine is ready.	Signals readiness for file processing.
Failed to initialize EasyOCR reader from local storage: {e}. Ensure models are in {EASYOCR_MODEL_DIR}	✗ Error	Alerts user to OCR setup failure.	Helps diagnose issues and prevents further errors.
OCR functionality is not available due to initialization failure.	✗ Error	Warns that OCR cannot be used.	Reinforces earlier errors and blocks further actions.
Performing OCR on image-based PDF pages (this can be memory intensive for large files).	i Information	Notifies users that OCR fallback is in progress.	Explains delay and adds transparency to application logic.
Skipping empty image for PDF page X.	⚠ Warning	Alerts users that a page couldn't be processed.	Helps users understand partial results or issues.
Error during PDF to image conversion or OCR fallback: [error]. This could be due to the size or complexity of the PDFs which exceeds memory limits. Try a smaller PDF.	✗ Error	Communicates failure during fallback OCR.	Offers context and suggests trying smaller files.
The model {model_name} could not be found. It may have been deprecated or is unsupported in the current API version. Please select a newer model.	✗ Error	The model that is used has been deprecated or is unsupported in the current API version.	Provides clear, actionable insight into why your API call failed.
Translation failed due to API quota exhaustion. Please wait or upgrade your plan to continue.	✗ Error	Informs users of quota-related failure.	Helps users understand API limits and next steps.
Translation failed. Please try again or check the input.	✗ Error	Generic translation error message.	Encourages retry and input validation.
No input detected.	✗ Error	Warns users that no	Prevents empty

Please type something or upload a file.		text was provided.	translation attempts.
Translation only works with English input. Please switch your text to English and try again.	✖ Error	Alerts user to unsupported input language.	Ensures translation logic works as intended.
Message	Type of Message	Purpose	Usefulness
Translating... 	 Spinner	Indicates translation is in progress.	Keeps users informed during processing delay.
Generating audio... 	 Spinner	Indicates TTS conversion is in progress.	Improves UX by showing active processing.
Text-to-speech is currently unavailable for [language]. Please select a different language to hear the spoken translation.	✖ Error	Alerts users to unsupported TTS language.	Guides users to choose a compatible option.
An error occurred: [error message]	✖ Error	Displays unexpected runtime errors.	Provides transparency and debugging info.

6. Project Structure

The project appears as follows in the dedicated github remote repository:

```

translate-and-speak/      # Root directory of the Translate & Speak application
├── assets/                # Static Resources
│   └── images
│       └── sample_files/  # Sample input files
├── ml_models/             # ML models
│   └── easyocr/           # EasyOCR pre-trained model files
│       ├── craft_mlt_25k.pth # Text region detector (CRAFT)
│       └── english_g2.pth   # English text recogniser (G2)
├── utils/                 # Helper scripts
│   └── file_parser.py      # Parses and extracts text from files
├── .gitignore             # Tells Git which files/folders to ignore (e.g., .env, venv)
├── README.md              # App overview and usage
├── packages.txt            # Python dependencies
├── runtime.txt             # Python version spec
├── translator_speak_app.py # The main Streamlit app logic
└── documentation.pdf       #Project documentation

```

At its core is a streamlit-based application that manages user interaction, file uploads, text extraction, translation and speech synthesis. There are sample test documents in various formats which can be used to test the application's OCR and translation capabilities.

A key component of the system is the pre-trained machine learning models housed in the `ml_models/easyocr` directory. There are two models:

- `craft_mlt_25k.pth` : responsible for text detection, identifying regions within images that contain text using the CRAFT (Character Region Awareness for Text detection) algorithm.
- `english_g2.pth` : once text regions have been detected, this model performs text recognition. It interprets the characters within the text regions and is specifically optimized for English language input.

The supporting script `file_parser.py` is stored in the `utils` folder and will ensure that the input text strings/documents are properly processed before translation.

Configuration files like `requirements.txt` and `runtime.txt` define the Python environment and dependencies, `packages.txt` lists system-level dependencies that need to be installed on the deployment platform and documentation files provide guidance for setup and usage.

7. Deployment to Streamlit Community Cloud

This lightweight application was built with Streamlit and the repository is public, allowing for easy deployment on Streamlit Community Cloud.

7.1 Key Benefits of Streamlit Community Cloud

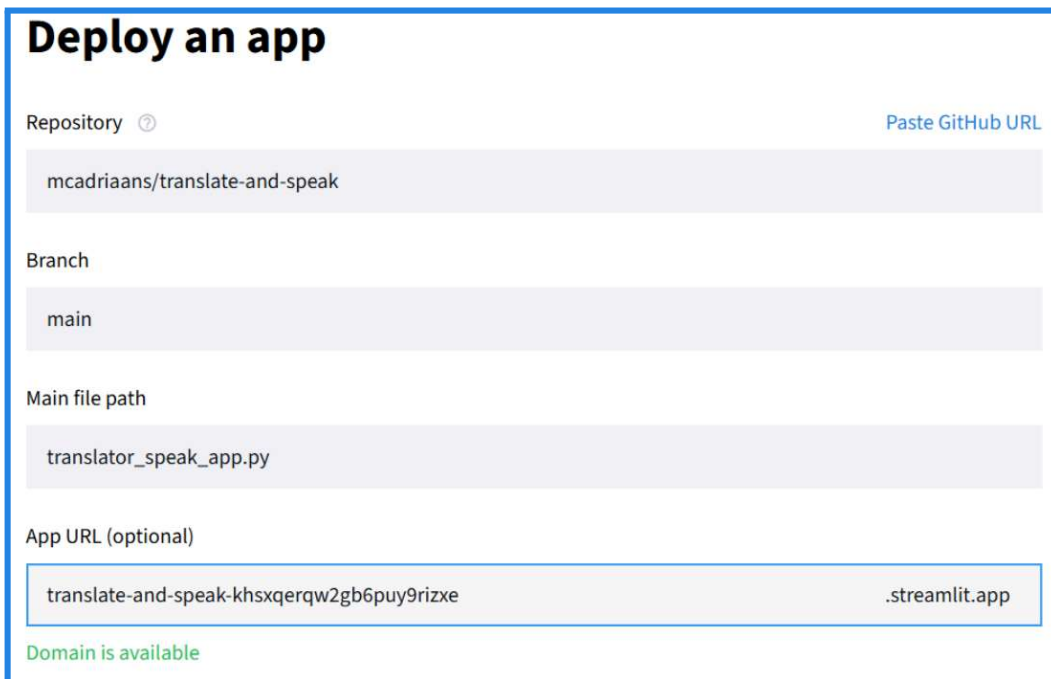
- The application can be deployed at no cost.
- The deployment is very fast. We simply connect the GitHub repo and click Deploy.
- Since the application is tied to GitHub it makes for easier collaboration with others.
- Real-time updates as updates made to the code in GitHub are instantly reflected in the cloud.
- Streamlit Community Cloud allows users to safely store API Keys.

7.2 Configuration and Launch

First we navigate to share.streamlit.io and log in with a GitHub account.

7.2.1 Configure Deployment Settings

Click the “New application” button to access the “Deploy an application” dialog box. Select the GitHub repository that contains the codebase, branch for deployment(typically main) and then specify the path to the main script.



Deploy an app

Repository ⓘ Paste GitHub URL

mcdriaans/translate-and-speak

Branch

main

Main file path

translator_speak_app.py

App URL (optional)

translate-and-speak-khsxqerqw2gb6puy9rizxe .streamlit.app

Domain is available

Figure 3: Streamlit Cloud Deployment Configuration

Streamlit Community Cloud does not directly support .env files for security reasons. Instead, environment variables (such as API Keys) can be securely stored using **Streamlit Secrets**.

From the deployed application navigate to “Manage application” (gear icon bottom right) then select “Secrets”. Now we can add the Google API key in TOML format.

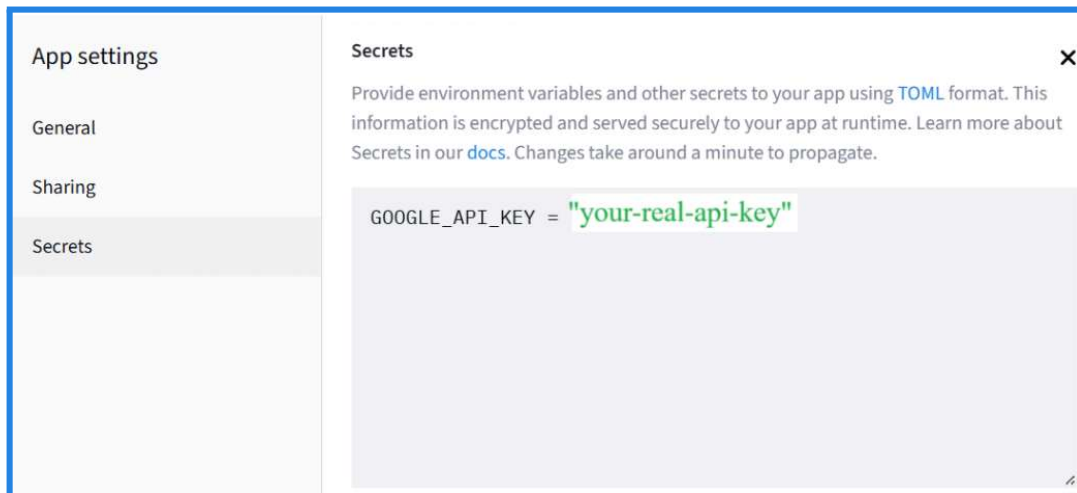


Figure 4: Streamlit Secrets Configuration

It is crucial to select the Python version that was specified in the runtime.txt file to ensure compatibility and avoid errors during the build process. For this project we select version 3.10.

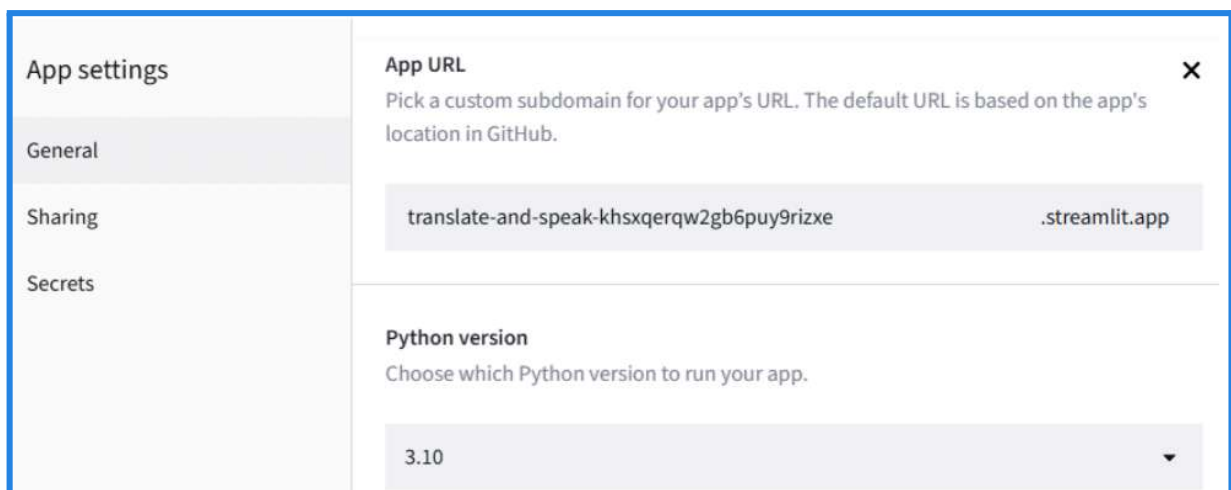


Figure 5: application URL and Python Version Selection

8. Testing & Debugging

8.1 Local Deployment Testing

Executed a series of test cases using the various allowable file formats : PDF, TXT, CSV, XLSX, PNG, JPG and JPEG, to verify the accuracy of the parsing logic and robustness of the application. Key findings and resolutions from these tests include:

OCR Library Deployment Challenges : Transition from [pytesseract](#) to [easyocr](#)

Initially, pytesseract (Python wrapper for the Tesseract OCR engine) was used for OCR local deployment. However, when attempting to deploy to the Streamlit Community Cloud , the application consistently failed to extract and subsequently translate text from documents that require OCR processing.

- **Root Cause:** The [pytesseract](#) library depends on the external Tesseract OCR engine, which poses compatibility challenges within Streamlit Community Cloud's containerized environment—an environment that restricts the installation of native binaries. As a result, deployments frequently encountered issues where the Tesseract executable was either absent or improperly configured, leading to OCR failures during runtime.
- **Resolution:** The OCR system was refactored to use [easyocr](#), resulting in improved reliability and seamless cloud deployment.

Image Upload Errors (File Pointer Issue)

✗ **Observed Error:** Encountered the error message *"No input detected. Please type something or upload a file."* even after successfully uploading image-based files, leading to confusion and failed translation attempts.

- **Root Cause:** The file pointer wasn't consistently reset prior to reading the files, leading to an empty buffer and subsequently, no text being passed to the OCR engine.
- **Resolution:** Implemented `uploaded_file.seek(0)` in [file_parser.py](#) to explicitly reset the file pointer before any read operation on uploaded images, ensuring all content is captured.

Text File Encoding Issue

A `UnicodeDecodeError` (specifically *`UnicodeDecodeError: 'utf-8' codec can't decode byte 0xb3 in position 622: invalid start byte`*) was encountered when attempting to decode content from certain .txt uploaded files . This prevented successful text extraction and subsequent translation for those specific files.

- **Root Cause:** The application's decoding logic for .txt files explicitly tried to interpret content using "utf-8" encoding. However, some .txt files were actually encoded using a

different character set causing the operation to fail when it encountered unfamiliar characters.

- **Resolution:** The text decoding strategy in `file_parser.py` for .txt files was updated. Instead of a strict "utf-8" decode, it now uses "latin1" (ISO-8859-1) encoding. latin1 is a more permissive encoding that can successfully decode any single byte sequence without throwing an error, significantly improving compatibility with a wider range of .txt files, particularly those originating from older systems or different regional defaults.

Tested Edge Cases for Error Handling

The following scenarios were specifically tested to confirm the application's robust error handling as designed, verifying that the appropriate user feedback messages (as detailed in [5.6 Robust Error Handling and User Feedback](#)) are displayed:

- **Empty Input:** Verified that the application correctly prevents processing when no text is entered or extracted from an uploaded file.
- **Non-English Input:** Confirmed that the application accurately detects non-English input and prompts the user to provide English text for translation.
- **API Quota Exhaustion:** During testing Gemini API quota was exceeded and the application gracefully handles this by displaying an informative error message.
- **Unsupported Languages for gTTS:** Verified that for target languages where gTTS lacks speech synthesis support, the application correctly omits audio generation and notifies the user of this limitation.

8.2 Streamlit Cloud Deployment Challenges & Solutions

We experienced deployment timeouts on Streamlit Cloud primarily due to the size of the **EasyOCR** models.

- **Challenge:** When `easyocr.Reader` initializes, it attempts to download large models (hundreds of MBs) from the internet if not found locally. Streamlit Cloud's free tier has a **strict startup timeout limit (typically a few minutes)**. The model download time often exceeded this limit, causing deployment failures.
- **Solution:** We addressed this by embedding the EasyOCR models directly into the Git repository via the directory `ml_models/easyocr`
- **How it Works:** When the Streamlit Community Cloud clones the repository, these model files are already present on the server. By configuring `easyocr.Reader` to load models from the local model storage directory and combining it with Streamlit's `@st.cache_resource` decorator, the time-consuming network download is completely bypassed. This ensures the (still memory-intensive) model loading operation happens only once upon application deployment, significantly improving startup efficiency and reliability on the deployment platform.

9. Limitations

- **gTTS Language Support** : For example, although the Gemini API successfully translated English into Zulu(another language widely spoken in South Africa); the library was unable to generate audio output for the translated text.
- **File Size Constraints** : Maximum upload size is 200MB. This is ideal for .txt, .csv or .xlsx formats but could be limiting if users upload large image-based or media rich scans or multipage documents.
- **No Real-Time Preview**: Users must wait for translation and audio generation before seeing results.
- **Translation Accuracy in Gemini 1.5 Flash**: The multimodal language model is known to be optimized for responsiveness and cost efficiency, it can however sometimes, generate overly literal translations(directly translates word for word) that fail to adequately capture idiomatic nuance and compromise the comprehensibility of the translation.
- **No Voice Customization**: gTTS does not support pitch, speed or voice selection.

10. Notable Challenges

- **File Parsing** : Handling diverse file formats and extracting clean text.
- **Image OCR Reliability**: The application can occasionally encounter inconsistent text extraction from image files (.png, .jpeg, .jpg) due to factors such as image quality, font style, text orientation, compression artifacts, and contrast levels. This can lead to the error message: "No text found in the image". The error is displayed in the selected language.
- **API Rate Limits**: Managing Gemini API usage within quota constraints.
- **Language Compatibility**: Ensuring gTTS supports the selected translation language.
- **No GPU available in Streamlit Community Cloud**: Most cloud-hosted Streamlit environments don't provide GPU access, this can tremendously slow down the application's performance.

11. Future Enhancements

- We can look to add **support for additional file types** (e.g., DOCX, RTF, HTML etc.)
- Look to enable **batch processing** for multiple files.
- Add a **real-time translation preview** before audio generation.
- **Store previous translations** for reuse or review.
- Offer **voice customization** options (pitch, speed, gender).

Conclusion

This project stands as a testament to the capabilities of Generative AI in creating impactful language tools. The application is well-positioned for continued growth and refinement, aiming to further expand its functionalities and reach.