LINE TTS: Local Interactive Narration Environment for Text-To-Speech

General description of the project (2 pts)

This project seeks to leverage a lightweight, open-weight TTS model (https://huggingface.co/hexgrad/Kokoro-82M) to create a fully-functional Text-To-Speech (TTS) desktop application. The project will likely make use of libraries such as kokoro, pytorch, streamlit, and PyMuPDF. This project would require the following development paths to be successful, listed in order of importance:

1. Read in existing file or string input as text parseable by an open-source library leveraging an open-weight model
    a. Reading in user-provided text or even a ".txt" file should be trivial, but ingesting text from a ".pdf" would require working with a library such as PyMuPDF.
2. Provide method to ingest text and output audio files
    a. Will need to use "kpipeline"s, found in kokoro
    b. Will need to implement the capability to generate combine to a single file, as it seems most of the methods found automatically segment audio files during the generation process
    c. Will require GUI development using Streamlit
3. Enable the user to generate a unique voice for narration
    a. "Voice blending" allows for blending user-provided voices to generate a unique narrator from existing pytorch model file (.pt)
        i. Will provide users with some pre-configured voices found in Kokoro-82M, as well as referencing in the GUI (https://huggingface.co/hexgrad/Kokoro-82M/tree/main/voices)
    b. Could be achieved using several sliders or percentage inputs in a voice generation menu
    c. Will require additional GUI development using Streamlit
4. Audio summarization of provided text
    a. Provides a separate audio file as well as a text summary of the summarization
    b. Leverages the "OpenAI API" to create summary

In plain terms, upon completion of these paths, a user should be able to use this application to generate a narration of their submitted text using a narrator of their choosing or creation, with the option to generate an additional summary of the text.

Task Vignettes (User activity "flow") (4 pts)

An example user may want to listen to an article they were reading while driving to work. The user generates a voice, in this case a clear, articulate voice to narrate a scientific article. The user now has an audio file to listen to anywhere.

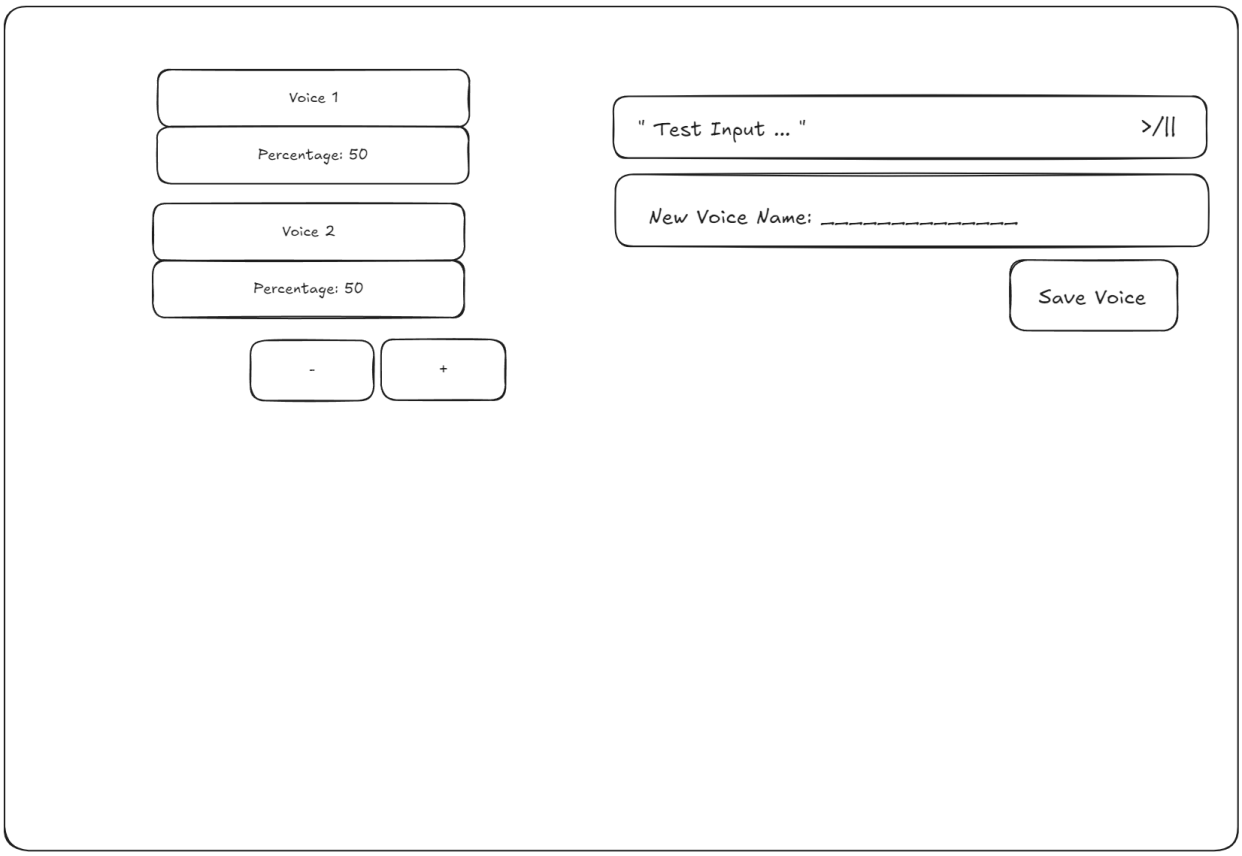
Task 1: Generating\selecting a voice for narration

- User will have access to list of voices
    - dropdown menu is one option
    - Potentially file explorer depending on how generated voices are stored
        - File explorer would be better for a user flow that more directly accounts for new voice generation
- User will be able to control the percentage the voice is “blended”
    - Potentially using sliders?
    - The voices are controlled in percentages, and will require enforcement to ensure this is a valid value. Could perform a hard check before saving
        - May be better to have the user manually input percentage numerically for more granular control, ensure desired outcome
- User should be able to store generated voice for later use if desired

Task 1 user activity:

- The user opens the application, and is greeted by a simple welcome menu with instructions
- The user selects the voice from their presaved .pt files (including some provided in the repository)
- The user adjusts the voices to a blending of their choosing
- The user sets a new name for the voice model they’ve created, and has the option to save & download it to a .pt if desired

Initial mental models, subject to change:


Voice 1
Percentage: 50
Voice 2
Percentage: 50
-
+
" Test Input ... "
>/||
New Voice Name: _____
Save Voice

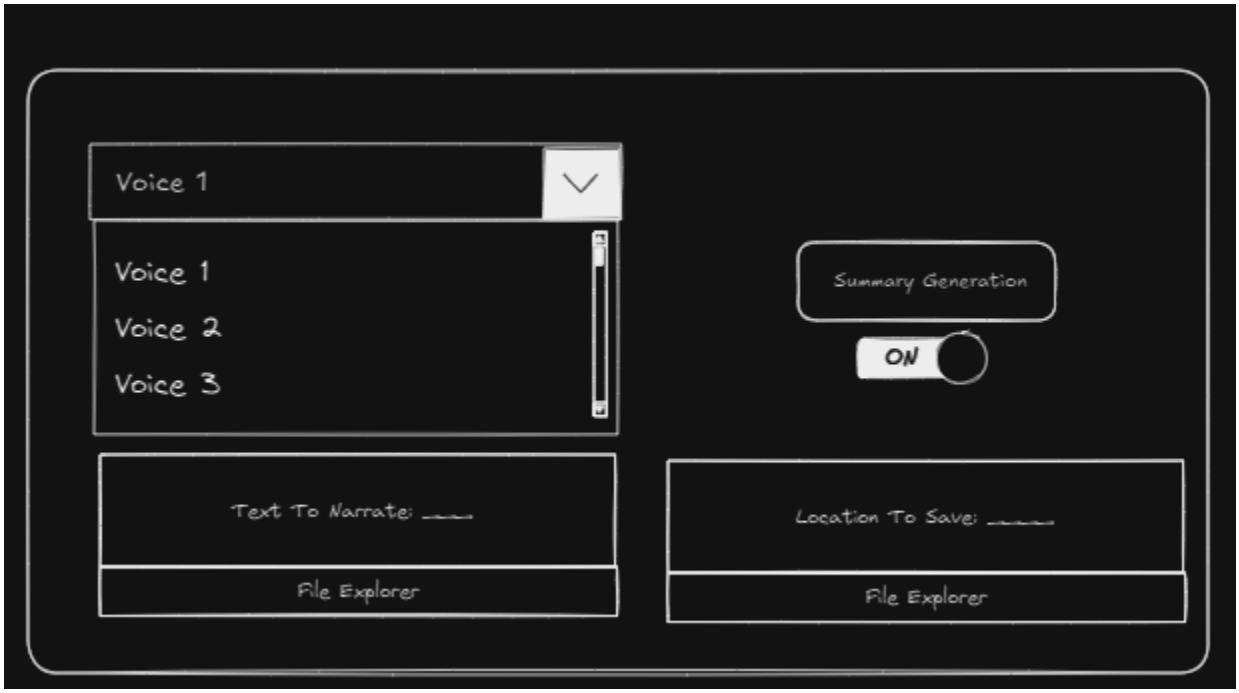
Task 2: Generate narration, with or without summary

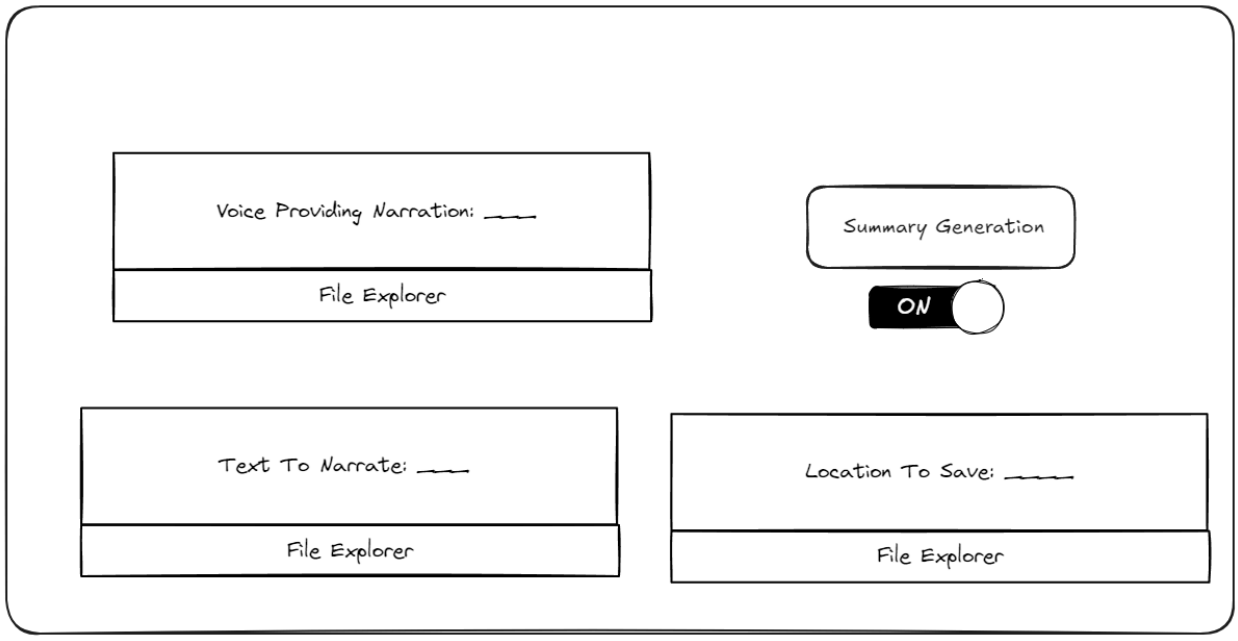
- User must select voice from list of voices or files
    - Again, drop down menu or file explorer could be used here
    - File explorer would be better for a user flow that more directly accounts for new voice generation
- Provide a way of selecting text to be narrated (.pdf, .txt, or provided text)
    - This will require parsing of the file if the user selects it, and processing using a TTS pipeline to generate narration
- Provide a way of selecting where to store output location in streamlit UI.
- Provide option to generate summary of text
    - Output as text for the user to read and narration of the summary
    - Utilizing OpenAI's GPT

Task 2 user activity:

- The user selects the voice (possibly a previously blended one) for the narration
- The user is prompted to upload the text in an accepted format (.txt, .pdf at minimum)
- The user decides to opt into or out of summary generation
- The user selects the file name & location to save
- The user runs the application, is given feedback displaying that the application is currently generating their audio narration
- The user is prompted (likely using a toast) that their narration has been generated
    - Possibly provide a way to listen to the audio/summary within the GUI?

Initial mental model, subject to change:



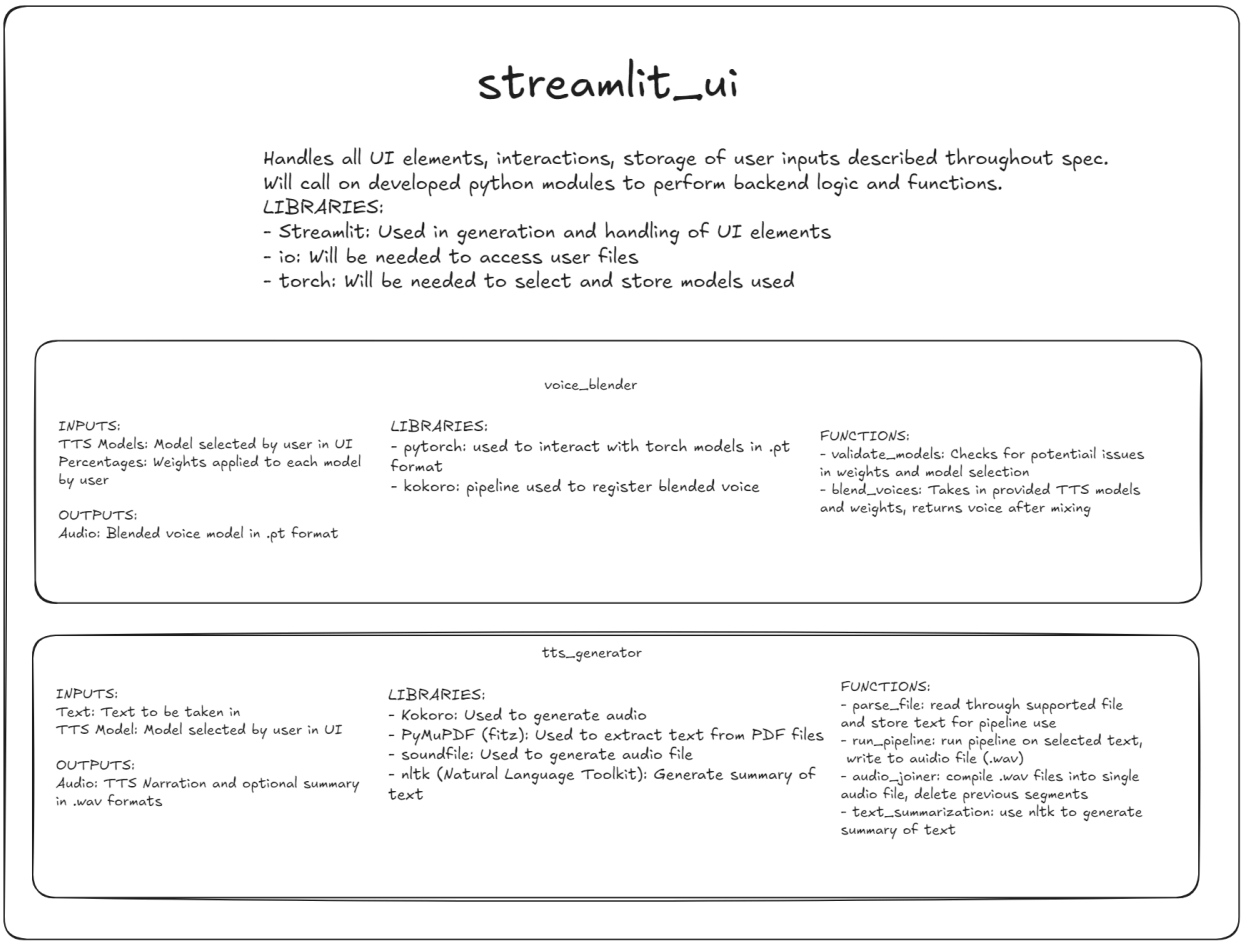


Version 1 should at minimum be able to read in text, be capable of blending voice models to generate a new one, and be capable of generating a compiled audio narration file based on the text provided.
By version 2, I should have the streamlit side of the application fully cleaned and usable, as well as a completed text summarization technique.

Technical "flow" (3 pts)

- **Data Input:** Voice Model(s)
    - User will be provided several pretrained models provided by Kokoro-82M
    - Kokoro and PyTorch will be the primary libraries responsible for interacting with the models used for text-to-speech narration
        - The open-weight models provided by Kokoro-82M are stored in .pt files, meaning they're already stored as PyTorch models, so working with the PyTorch library will be required
        - User will need to specify percentages in some way
            - Will need to at minimum check and warn user if there is an issue with selected weights, e.g. invalid values
                - *A function for validating voice weights*
        - "Pipeline" to allow for processing of text will be created using "KPipeline" from the kokoro library
            - *A function that processes using model*
    - Note: the weights could be 100% for one model, allowing the user to entirely use a preselected model without any blending
    - **Data Output**: Model for narration, saved in a .pt format
- **Data Input:** Text
    - Provided in the form of text typed into a field, or a file uploaded in an acceptable format (such as .txt, .pdf)
        - The PyMuPDF used here
    - Will require processing of text
        - *A function for text ingestion*
        - Results in summary generation
        - Results in generation of audio narration using the model created by the user, referenced above
    - **Data Output**: Summary of text (optional)
    - **Data Output**: Audio narration
- **Data Input:** Summary of text (optional)
    - Boolean controlled by a button, "summary_requested"
    - Check after text is provided and ingested. If this boolean is true…
        - *A "text_summarization" function*
        - Leveraging OpenAI's API
    - **Data Output:** Summary of text, provided in text and audio format

Initial Structure Mockup:



Final (self) assessment (1 pts)

There were two large unexpected changes from my initial spec:

1. I wasn't expecting the recommendation to include a summary of the text, which I think I understand how to implement but may require some learning that I wasn't accounting for.
2. I wasn't expecting to want to work with streamlit, which will require that I learn a new framework for GUI design.

- These are also the biggest anticipated problem - tying it together into one cohesive product tied together with a proper frontend will require the most effort. This is likely the piece I will come to you for if any, although I am relatively confident I can at least get something working as the streamlit library seems relatively well documented.
- There is some crossover in work I've done for my job and PyTorch that should aid in understanding.
- I've never worked with the kokoro, fitz, streamlit, or NLTK libraries before.

Edits post-version 1 implementation in red