Deep Learners: Learning Tools for Transcribed Lecture Audio – Application Design Document –

Cyrus Asgari, cyrusasgari@college.harvard.edu
Ben Ray, benray@college.harvard.edu
Caleb Saul, cbsaul@college.harvard.edu
Warren Sunada Wong, wsunadawong@college.harvard.edu
Chase Van Amburg, cvanamburg@college.harvard.edu

1. Architecture

We divide our architecture design into a solution and technical component. The solution architecture helps identify the building blocks in the app, while the technical architecture captures

Solution Architecture:

Our solution architecture is divided into three classes of components – Process, Execution, and State. A summary of the basic functionality of these components is provided below.

Process:

- Data processing
- Model training/tuning
- Build app
- User can upload video
- View key-word highlighted transcription and generated quiz results

Execution:

- Take a lecture video and apply the same preprocessing to acquire audio for transcription
- Use the best fine-tuned Distilbert model to generate key-word predictions
- Return resulting highlighted transcript to user
- Return set of questions and answers generated based on transcribed text
- Track the best model

State:

- Save lecture videos and transcriptions to a common store
- Save model weights
- Information on preprocessing
- Source code repository for collaboration

The interactions between these three classes can be visualized with the provided diagram in Figure 1.

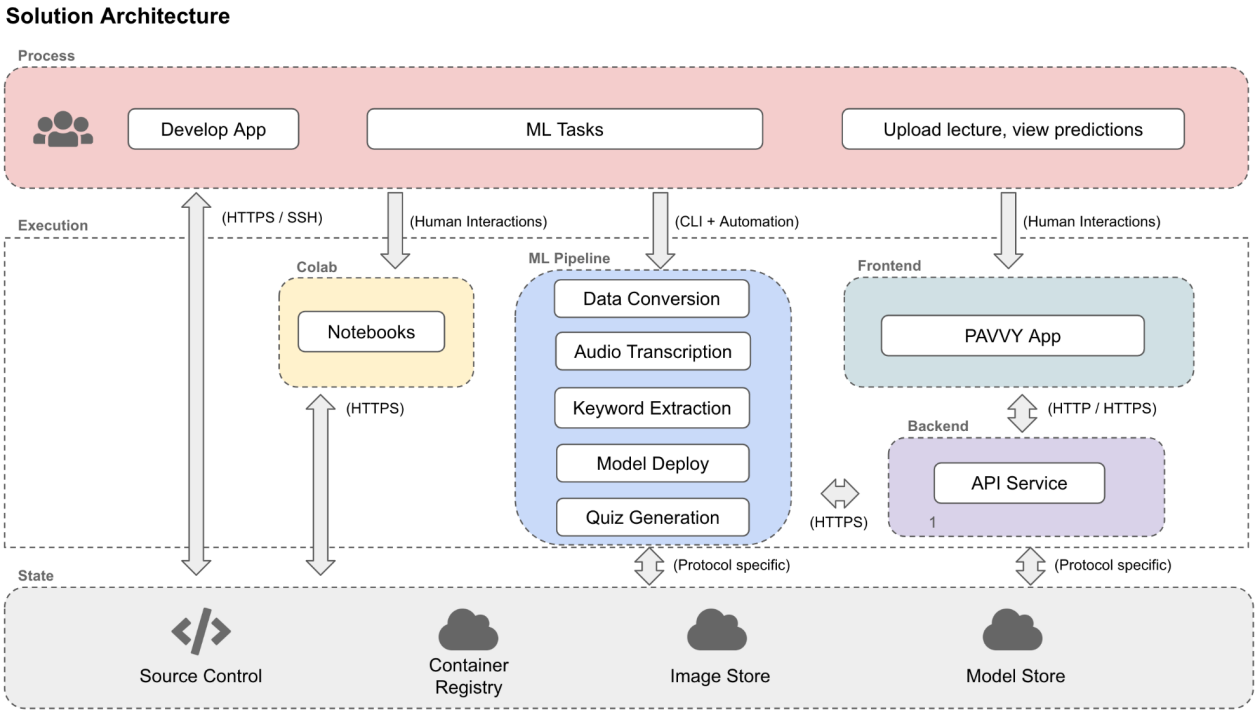

Figure 1: Solution Architecture

Technical architecture:

A high level view of the app from development to deployment, illustrating the interactions between our components and containers. Our solution is broken into three groups of people: Developers, Data Scientists, and Users. A summary of the three components is provided below:

Developers:

- Use IDE (VSCode), CLI to build web-app
- Expose the frontend and backend APIs to the outside so the user can access them using the browser
- NGINX container acts as a reverse proxy to funnel requests from outside to the processing containers.
- App built with React framework to provide a nice user interface
- All development is containerized
- Data storage: since the application is stateless, we have no need for a persistent store, and we will instead use a new GCP bucket store to transfer intermediate data between containers.

Data Scientists:

- Use Colab for exploratory data analysis and model selection
- Use Vertex AI for model training.

- Whisper JAX for audio transcription, ChatGPT for quiz generation, data conversion from MP4 to MP3, and keyword extraction with BERT
- Use IDE (VSCode), CLI to build ML tasks

Users:

- Upload lecture using the browser
- Receive lecture transcript with keywords highlighted, and a generated quiz.

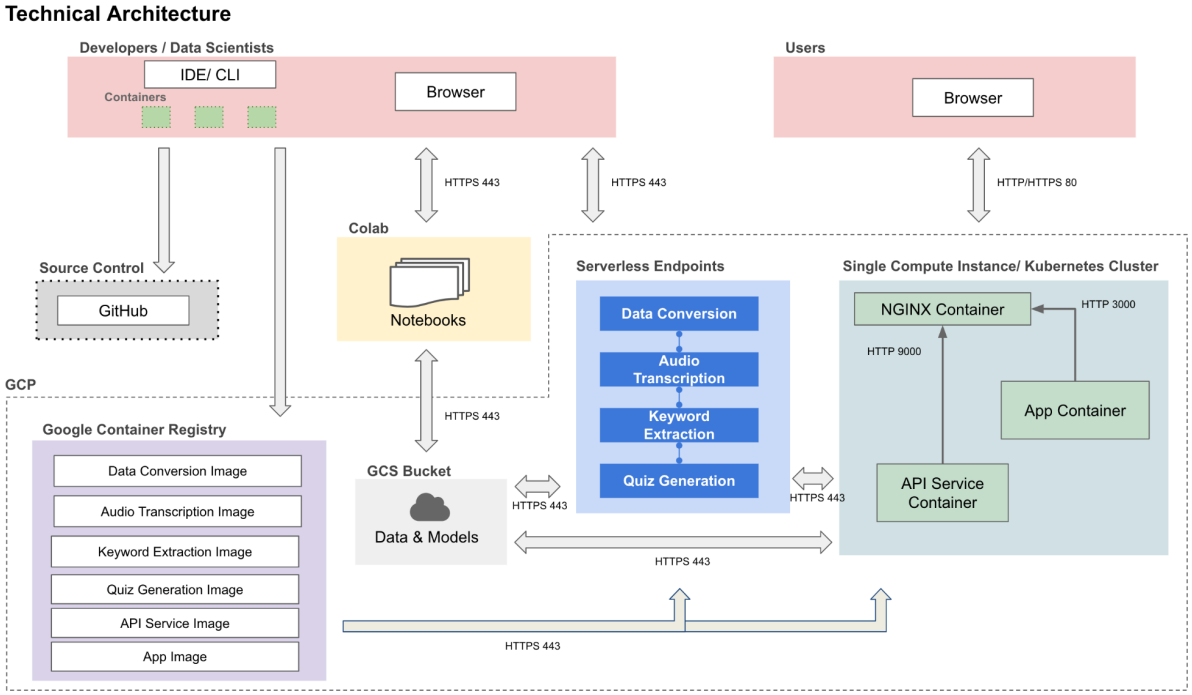

Figure 2: Technical Architecture

2. User Interface

A mockup of the planned user interface is shown below. We opt for a minimalist design with two pages. In the first page, the user uploads the lecture video they want to use. After a successful upload, the app transitions to the second page where the user can read the lecture transcript and comprehension quiz.

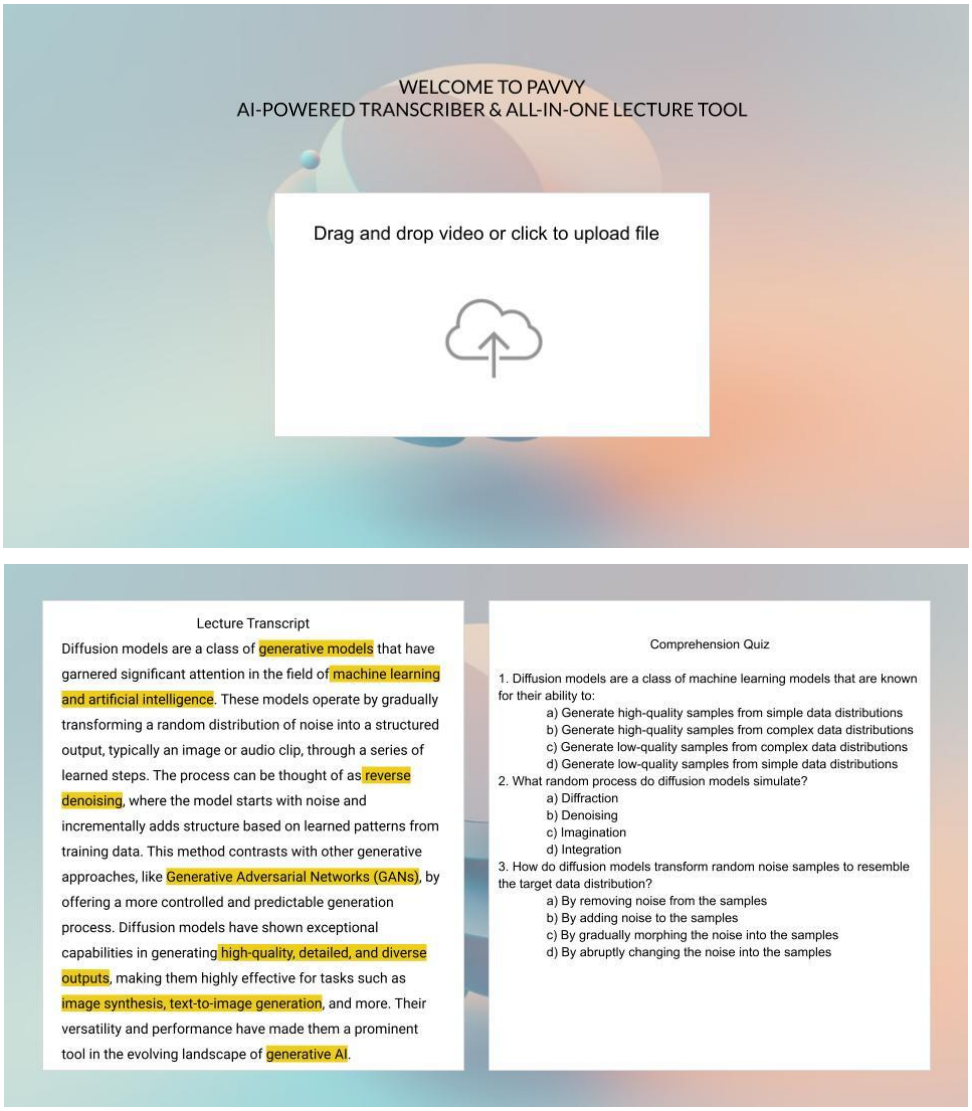

Figure 3: App Mockup

3. Code Organization Principles

- Fast iteration cycles and testing using Jupyter notebooks
- Containerization for machine independent usability
- Github for source control
- Encapsulation of different components, divided into the front-end and back
- Prioritize readability over conciseness