Akash Verma
akashzsh08@gmail.com
Chhattisgarh, India
Timezone: Indian Standard Time (UTC+05:30)

Revolutionizing E-learning: Building a Cutting-Edge Audio/Video Transcription Generator and Editor

GSoC Project proposal for Kolibri Studio

Personal Details and Contact Information

➔ **Name:** Akash Verma
➔ **Github:** @akash5100
➔ **Email:** akashzsh08@gmail.com
➔ **Timezone:** Indian Standard Time (UTC+05:30)
➔ **University:** Shri Shankaracharya Institute of Professional Management and Technology.
➔ **Portfolio:** Resume / Linkedin / Website

Synopsis

The goal of this project is to add a feature to Kolibri Studio that allows for the automatic generation and editing of captions for uploaded audio and video resources. Currently, users can upload caption files in WebVTT format, but there is no way to edit or preview them within the Studio. To enhance the accessibility of Kolibri's resources for learners, this project will focus on two main tasks: (1) implementing an asynchronous, self-hosted solution for auto-generating captions in the same language as the video with the ability to be translated into different languages and (2) adding frontend support for previewing and editing uploaded or auto-generated captions.

To achieve the first task, we will explore various libraries and tools that support the languages already supported by Studio, as well as potentially expanding the number of supported languages. We will consider both proprietary and open-source options, including **Google Cloud Speech-to-Text** service and emerging open-source libraries like OpenAI's **Whisper**. The chosen solution will be integrated into Kolibri Studio's existing infrastructure to ensure seamless operation.

For the second task, we will build a frontend editor that allows users to preview and edit captions in real-time. This editor will support both uploaded and auto-generated captions and will allow for basic text editing and synchronization with the video or audio resource. The editor will also support multilanguage options, with automatic translation to be considered as a future enhancement.

Overall, the auto-generated captions and editor feature will significantly enhance Kolibri Studio's accessibility and usability, making it easier for learners to access and engage with educational resources.

I will be focusing on implementing the auto-generation of captions and editor feature within the Learning Equality Studio repository with **Blaine Jester** and **Richard Tibbles** as my mentors.

Benefits to the Community

Creating accurate WebVTT files for video lectures or audio resources can be a time-consuming and tedious task, especially for content creators who may not be familiar with the technical requirements and formatting rules. The absence of subtitles or transcriptions in an E-learning platform can be a significant barrier to learning, as it can hinder accessibility and comprehension for students who are deaf or hard of hearing, non-native speakers, or simply who prefer to learn with captions.

The addition of an auto-generate captions feature to Kolibri Studio will enhance the platform's accessibility and usability, making it easier for content creators to seamlessly create content without being worried about adding captions by themselves as well as for learners to access and engage with educational resources. This will help progress Studio by making it more inclusive and user-friendly, potentially increasing its usage and reach.

Anticipated benefits to the community:—

- **Increased language support** — A wider range of languages, potentially making Kolibri Studio resources more accessible to non-native speakers.
- **Better engagement** — Captioned videos can help learners stay engaged with educational content, as they are less likely to miss important information.
- **Increased usage** — The addition of an auto-generate captions feature may lead to an increased usage of Kolibri Studio, as it will make the platform more accessible and user-friendly.
- **Greater reach** — By making educational resources more accessible, Kolibri Studio will be able to reach a wider audience.

The implementation of an auto-generate captions feature in Kolibri Studio opens up possibilities for future development in several areas. For example, the automatic translation of captions could be considered a future enhancement, allowing for even greater language support and accessibility. The frontend editor could also be expanded to include more advanced features, such as the ability to **add annotations** or **notes** to the captions.

Additionally, we could consider adding a **regex search** feature to the transcriptions, which would allow learners to easily skip forward or backwards in the video or audio resources.

Overall, the addition of an auto-generate captions feature to Kolibri Studio provides a foundation for continued learning and improvement in the areas of accessibility and usability.

Current Status of the Project

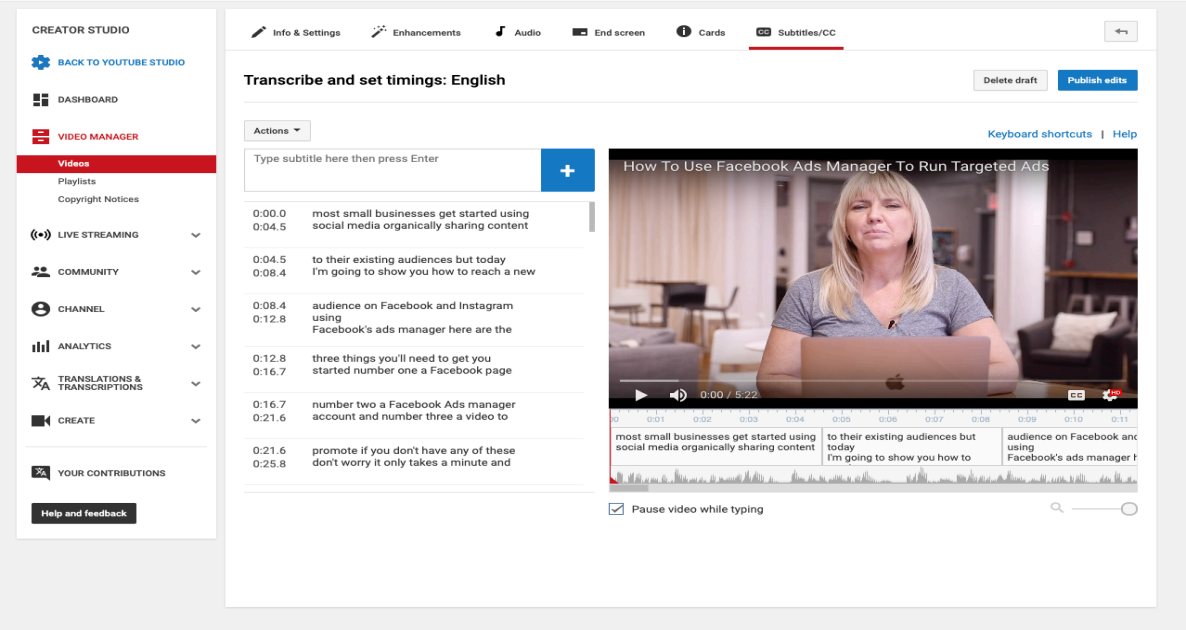
At present, the REST API endpoint “POST /api/file/upload_url” is responsible for handling the uploading of all types of files, including WebVTT subtitle files, as well as video and audio resources, to the database.

Regarding the project, the current frontend architecture exclusively supports uploading subtitle files that are associated with either a video or an audio file. Upon uploading, the file is stored in the PostgreSQL database. When the multimedia content is played in **Kolibri**, which utilizes **video.js**, it looks for a .vtt extension in the current ContentNode. If an associated subtitle file is found, it is added to the **<track>**.

Enabling Kolibri to provide auto-generated captions/subtitles without an internet connection

Kolibri is widely recognised for its ability to operate in regions where the internet is expensive, unreliable, or inaccessible. To facilitate the implementation of auto-generated captions, my proposal involves generating them automatically during the content creation process on Studio. It is important to note that the accuracy of the auto-generated captions may not always be reliable. Therefore, a Caption editor similar to the one provided by the popular Youtube Studio will be available for use.

Below is an image that depicts the caption editing feature in **YouTube Studio**:



We will use **Celery** to generate captions in the background without blocking the user interface. Studio uses Celery for executing asynchronous tasks, which are integral to Studio's channel editing architecture. We can use this current implementation to simplify the **asynchronous generation of captions**.

Goals

Goal 1 - Generating Valid WebVTT file

- **Generating the transcription from video input.**
  Starting the implementation with the Backend-First Development approach, we first implement the backend which will generate the subtitle/transcription for any multimedia input (audio, video).
- **Creating a WebVTT file using the transcription output.**
  The generated transcription is not really in the file format that we need so we need to generate content conforming to the WebVTT file format, which will be stored in the PostgreSQL database.
- **Creating a clear endpoint architecture to facilitate data transfer from the backend to the frontend.**
  Studio retrieves data in the frontend from read-only REST endpoints implemented using Django Rest Framework. I will decide the request

and response format of the API endpoints. The GET to populate the editor and POST/PUT to update the backend.

Goal 2 - Connecting Backend to Frontend

- **Implementing the GET API endpoint to populate the Editor.**
  Implementing the GET request API endpoint to populate the editor. We will use this endpoint when a user clicks the “edit caption” button in the frontend.
- **Implementing the PUT/POST request to update the changes made in the frontend to the database:**
  Implement a REST API endpoint for updating the WebVTT file in the database. This endpoint should accept a PUT or POST request with the updated content of the WebVTT file in the PostgreSQL database.

Goal 3 - Frontend Vue.js Component

- **Crafting the cool editor:**
  There must be some way to finetune the auto-generated WebVTT file. Clicking the “edit captions” button on the frontend vue.js component in the ChannelEdit Single Page Application will open a new component, namely the editor.
- **Populating the editor:**
  Populate the editor with the necessary REST API endpoint. And finally, allow the frontend changes to sync with the database when the “save” button is clicked.

Deliverables

Deliverable 1

Expected by Evaluation 1

- Implementation of the Transcriber function `def transcribe(video, audio)`
- Implementation of the error handling in the transcribe function.
- Testing

- Implement `def generate_vtt_file(data)` with the feature to upload the vtt file generated to Google Cloud Storage using “file_id” and a pre-signed URL.
- Django model `GeneratedCaptions` and the corresponding viewset `CaptionsViewSet`.
- Add necessary documentation.

Deliverable 2

Expected by Evaluation 2

- Implement create_from_changes, update_from_changes and delete_from_changes viewset action for `CaptionsViewSet`.
- **GET** request REST API endpoint to populate the editor, retrieving JSON file from PostgreSQL.
- **POST** request REST API endpoint for the “save” action, update the JSON file in the PostgreSQL database and update the already existing VTT blob in Google Cloud Storage.
- Initialize Frontend component **CaptionEditor** in ChannelEdit Vue.js SPA.

Deliverable 3

Expected by the end of the coding period

- Implement caption populating objecting with the help of the **GET** request API endpoint created earlier.
- Implement Video playback in the browser.
- Work on **syncing the Video and Caption** together.
- Integrate the **SAVE** feature in the frontend with the help of the **POST** request API endpoint created earlier.
- Add necessary documentation.

Expected Results

The final product of this project will be an enhanced version of Kolibri Studio that includes the following components and features:

1. Asynchronous, self-hosted solutions for auto-generating captions for audio or video resources.
    - ➔ The **backend** will be implemented first to generate subtitle/transcription for the input video.
    - ➔ The generated captions will be saved in JSON format and stored in the PostgreSQL database.
    - ➔ The JSON object will be used to convert into WebVTT and uploaded to Google Cloud Storage.
    - ➔ REST API endpoint will be implemented to allow for editing of the generated captions.

2. **Frontend** support for previewing and editing captions.
    - ➔ A new Vue.js component will be created to display the captions in a user-friendly way.
    - ➔ An editor component will be implemented to allow users to edit the captions.
    - ➔ The editor component will be connected to the REST API endpoint, so that changes made on the frontend will be synced to the backend.

In addition to the above components, we aim to implement the following stretch goals and features:

- ➔ Improved accessibility for Kolibri resources by automatically generating captions in **multiple languages**.

The final product will enable Kolibri Studio users to easily create and edit captions for their audio and video resources, improving accessibility and enhancing the overall learning experience for Kolibri users.

Approach

Overview of the approach

Before discussing the implementation, we need to determine the best tool available for caption generation. This may require exploring various speech-to-text APIs and libraries and evaluating their accuracy, usability and cost.

Once we have selected a suitable tool, we can begin the process of generating captions as an asynchronous task. This will involve using Celery, which is already integrated into Kolibri Studio, allowing us to process the transcription in the background without blocking the user interface. Studio also leverages django-celery-results to easily track the progress and status of tasks, as well as analyze the results of completed tasks. We can use this to store the task results in a database and use it to populate the frontend.

After generating the captions or transcriptions, it's important to provide users with an editor in the Vue.js frontend to allow them to make any necessary modifications. For this purpose, we'll need to create a REST API endpoint that accepts PUT or POST requests to update the captions in the database.

Finally, we need to create an editor that allows users to edit the captions in a user-friendly way. On the frontend side, Studio uses Vue.js, so we will use it to build a web-based editor. The editor should allow users to edit and maybe provide features like spell-checking and formatting.

Comparing Cloud-hosted and Self-hosted Solutions for Studio

Benefits and Drawbacks of Cloud-hosted solutions

I understand that the project summary favours a self-hosted solution. However, I believe it's still worth considering popular cloud-hosted solutions like Google Speech to Text. This is because it is easier to scale up or down,

and the cost of hosting a self-hosted solution can sometimes exceed that of a cloud-hosted solution. Therefore, we should explore every possible option to find the best solution for the project. Here is a quick summary of some popular Speech-to-Text (STT) services that I made:

API	Main Details	Language Supported	Pricing
Amazon Transcribe	Punctuation and formatting, telephony audio, customization and multiple speakers recognition	37	This doesn't meet the project requirement as language support is insufficient.
Google Cloud	Customization, batch and real-time modes, noise robustness, filters for wrong words relative to the context, flexibility in the source files storage	120	$0.024 per minute = **$1.44 per hour**
IBM Watson	Real-time mode, custom models, keywords spotting, speaker labels (in beta), word confidence, word timestamps, profanity filtering, word alternatives, smart formatting (in beta)	14	This doesn't meet the project requirement as language support is insufficient.

Microsoft Azure	Real-time mode, customization, formatting, profanity filtering, text normalization, integration with Azure LUIS, speech scenarios	58, if you filter **"audio + human-labelled transcript"**	**$1.40 per hour**

By leveraging the **Google Cloud Service**, we can easily integrate speech-to-text functionality into Kolibri Studio's existing infrastructure. As Kolibri Studio already uses Google Cloud Storage (used in utils/celery), this would be a good option as it would leverage the existing infrastructure and APIs, making it easier to integrate and maintain.

Benefits and Drawbacks of Self-hosted Solutions

A self-hosted solution have its own advantages, we have control over the infrastructure and security of data, and it can also be cost-effective in the long run compared to cloud hosting because we are **free from paying for plans** to the cloud service.

There are **two types of models** that can be used in Kolibri Studio's self-hosted solution. The **first** type is a **pre-trained model**, such as Google Speech-to-Text (although it is cloud-hosted, it's a pre-trained model), Mozilla DeepSpeech, or OpenAI Whisper, which does not require software developers to pay attention to training data because it is already pre-trained and can be fine-tuned according to the software's needs. The **second** type is a model that **requires training from datasets**, such as Kaldi ASR or CMU Sphinx, which have advantages such as good accuracy, and customizability, and may even support less common languages.

> **Radina Matic** 5 days ago
> Yes to both: depending on the platform Farsi and Persian are interchangeable, and various dialect of Fula are spoken in several territories across equatorial Africa, but the one we have Kolibri translated into (Fulfulde Mbororore) is from Cameroon, where our partner is located. This locale might be difficult to find in other libraries.

However, the **disadvantages of using a model that needs training** include being time-consuming, needing more storage resources as the new language is added, needing a large dataset for training, and requiring human testing to ensure correct translations (or can be done with training dataset). For Kolibri Studio, a **pre-trained** model would be best suited due to its ease of use and time efficiency.

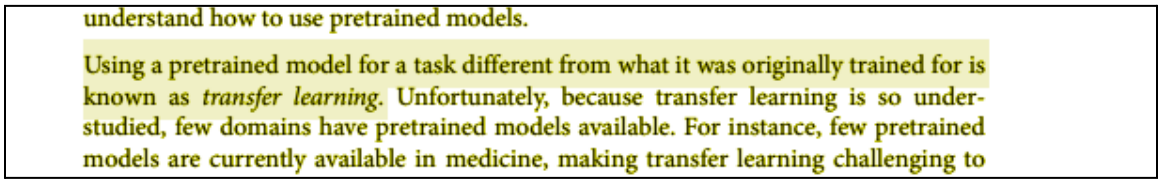
How can I say it requires more storage space?
Using Docker, I tried training the Kaldi ASR **Spanish** model (GitHub link to the example model that I trained) with the example dataset provided in the GitHub repository, and here are the results, the code snippet is an output of the terminal, shows the result of `**du -h**` command, stands for **Disk Usage**:

```
root@00bba0c337el:/opt/kaldi# ls egs/spanish_dimex100/
README.txt s5

root@00bba0c337e1:/opt/kaldi#                              ls
egs/spanish_dimex100/s5/exp/
make_mfcc   mono   mono_aligned   tril   tril_aligned   tri2b
tri2b_aligned tri2b_denlats tri2b_mmi_b0.05

root@00bba0c337el:/opt/kaldi# du -h
242M  ./egs/spanish_dimex100/s5/exp
4.5G  ./egs/spanish_dimex100/s5
4.5G  ./egs/spanish_dimex100
```

If the decision is made to integrate a self-hosted solution, I recommend considering a pre-trained model over a model that requires training. Pre-trained models can offer significant advantages like the speed of deployment, and ease of use, making them a better option for Kolibri Studio.

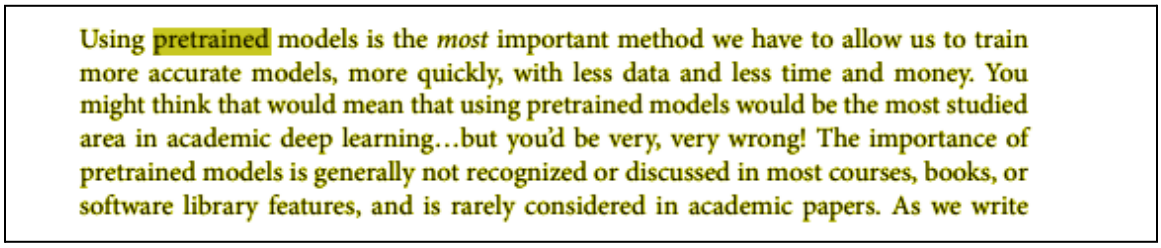
Key Takeaways from My Deep Learning Self-Study

I learned Deep Learning through the book "Deep Learning for Coders" by Jeremy Howard. It says that if we want a more accurate model with less data and less time and money, the most important method is using ***transfer-learning***. Here are some important points that I want to highlight through the images below.



understand how to use pretrained models.

Using a pretrained model for a task different from what it was originally trained for is known as *transfer learning*. Unfortunately, because transfer learning is so under-studied, few domains have pretrained models available. For instance, few pretrained models are currently available in medicine, making transfer learning challenging to



Using pretrained models is the *most* important method we have to allow us to train more accurate models, more quickly, with less data and less time and money. You might think that would mean that using pretrained models would be the most studied area in academic deep learning…but you'd be very, very wrong! The importance of pretrained models is generally not recognized or discussed in most courses, books, or software library features, and is rarely considered in academic papers. As we write

Researching the Best Speech-to-Text Solution for Studio

A Comparative Analysis of Self-Hosted Transcription Solutions for Studio: Identifying the Best Option Based on Research

One reasonable way to compare OpenAI's Whisper and Mozilla's DeepSpeech is by comparing their language support, resource requirements, and accuracy. As the OpenAI Whisper docs suggest, see tokenizer.py for the list of all available languages, it seems that the length of the LANGUAGES dictionary is **99**, whereas DeepSpeech releases only an **English-trained** model (says the community discussion) also there might be a possibility that currently, DeepSpeech has more supported language than OpenAI whisper, but Whisper provides the user to download the Multilingual model.

The accuracy of Speech-to-Text models is measured by **Word Error Rate** (WER). Word Error Rate is a metric used to measure the performance of speech recognition or text-to-speech systems. It is a measure of the difference between the transcription generated by a machine and the reference transcription, which is typically the actual text that was spoken.

A lower WER indicates better accuracy and performance of the speech recognition or speech-to-text system.

The DeepSpeech Research Paper shows a word error rate of **6.56%**

clean trained model and the noise trained model respectively. However, on the 100 noisy utterances the noisy model achieves 22.6% WER over the clean model's 28.7% WER, a 6.1% absolute and 21.3% relative improvement.

System	Clean (94)	Noisy (82)	Combined (176)
Apple Dictation	14.24	43.76	26.73
Bing Speech	11.73	36.12	22.05
Google API	6.64	30.47	16.72
wit.ai	7.94	35.06	19.41
Deep Speech	**6.56**	**19.06**	**11.85**

Table 4: Results (%WER) for 5 systems evaluated on the original audio. Scores are reported *only* for utterances with predictions given by all systems. The number in parentheses next to each dataset, e.g. Clean (94), is the number of utterances scored.

On the other hand, the OpenAI Whisper research paper shows a benchmark of Whisper running on different open-source datasets, revealing an average word error rate of 12.8%, which is higher than DeepSpeech's 6.56%.

Dataset	wav2vec 2.0 Large (no LM)	Whisper Large V2	RER (%)
LibriSpeech Clean	**2.7**	**2.7**	0.0
Artie	24.5	**6.2**	74.7
Common Voice	29.9	**9.0**	69.9
Fleurs En	14.6	**4.4**	69.9
Tedlium	10.5	**4.0**	61.9
CHiME6	65.8	**25.5**	61.2
VoxPopuli En	17.9	**7.3**	59.2
CORAAL	35.6	**16.2**	54.5
AMI IHM	37.0	**16.9**	54.3
Switchboard	28.3	**13.8**	51.2
CallHome	34.8	**17.6**	49.4
WSJ	7.7	**3.9**	49.4
AMI SDM1	67.6	**36.4**	46.2
LibriSpeech Other	6.2	**5.2**	16.1
Average	29.3	**12.8**	55.2

Table 2. **Detailed comparison of effective robustness across various datasets.** Although both models perform within 0.1% of each other on LibriSpeech, a zero-shot Whisper model performs much better on other datasets than expected for its LibriSpeech performance and makes 55.2% less errors on average. Results reported in word error rate (WER) for both models after applying our text normalizer.

However, it is important to note that the 12.8% WER is an average across multiple datasets, while DeepSpeech's performance was tested specifically on the LibriSpeech Dataset. In this dataset, Whisper's WER is 5.2%, lower than DeepSpeech's result.

The research also compares Whisper with other transcription systems:

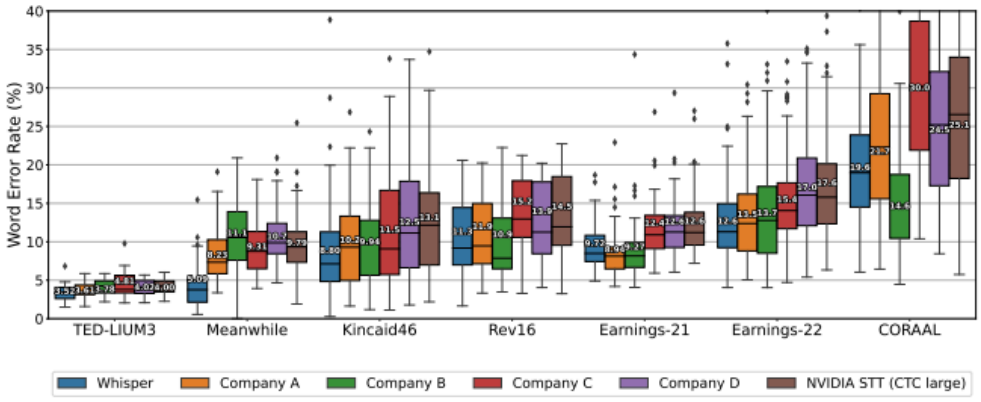
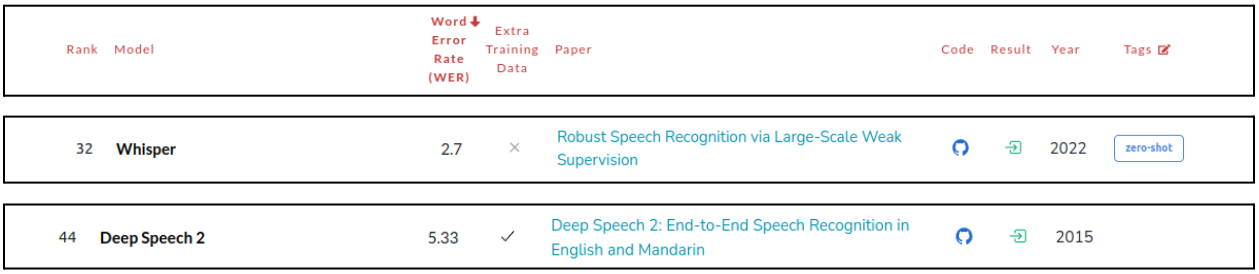


Figure 6. **Whisper is competitive with state-of-the-art commercial and open-source ASR systems in long-form transcription.** The distribution of word error rates from six ASR systems on seven long-form datasets are compared, where the input lengths range from a few minutes to a few hours. The boxes show the quartiles of per-example WERs, and the per-dataset aggregate WERs are annotated on each box. Our model outperforms the best open source model (NVIDIA STT) on all datasets, and in most cases, commercial ASR systems as well.

The image below shows the result of comparing both models to the same dataset (LibriSpeech). Whisper has 2.7 WER while DeepSpeech has 5.33 WER.



Rank	Model	Word Error Rate (WER)	Extra Training Data	Paper	Code	Result	Year	Tags
32	**Whisper**	2.7	×	Robust Speech Recognition via Large-Scale Weak Supervision			2022	zero-shot
44	**Deep Speech 2**	5.33	✓	Deep Speech 2: End-to-End Speech Recognition in English and Mandarin			2015	

OpenAI Whisper is made open-source on September 21, 2022, and it demonstrates a strong ability to generalise to many datasets and domains without the need for fine-tuning, giving tough competition to most of the existing open-source as well as Cloud services like Google Cloud, Azure etc. OpenAI Whisper provides a Multilingual model which means the developer should not worry about training separate models for different languages.

Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	`tiny.en`	`tiny`	~1 GB	~32x
base	74 M	`base.en`	`base`	~1 GB	~16x
small	244 M	`small.en`	`small`	~2 GB	~6x
medium	769 M	`medium.en`	`medium`	~5 GB	~2x
large	1550 M	N/A	`large`	~10 GB	1x

Conclusion: Unveiling the Ultimate Speech-to-Text Solution

For Studio, a self-hosted solution has more advantages over using a cloud-hosted solution. We have control over the infrastructure and security of data, and it can also be cost-effective in the long run compared to cloud hosting because we are free from paying for plans to the cloud service.

To address this challenge, we can explore hosting solution like HuggingFace, which simplifies hosting inference models by turning them into APIs. By leveraging HuggingFace, we can spin up and down hardware for the

autogenerate captioning feature as needed, reducing the burden of maintaining and scaling our own servers.

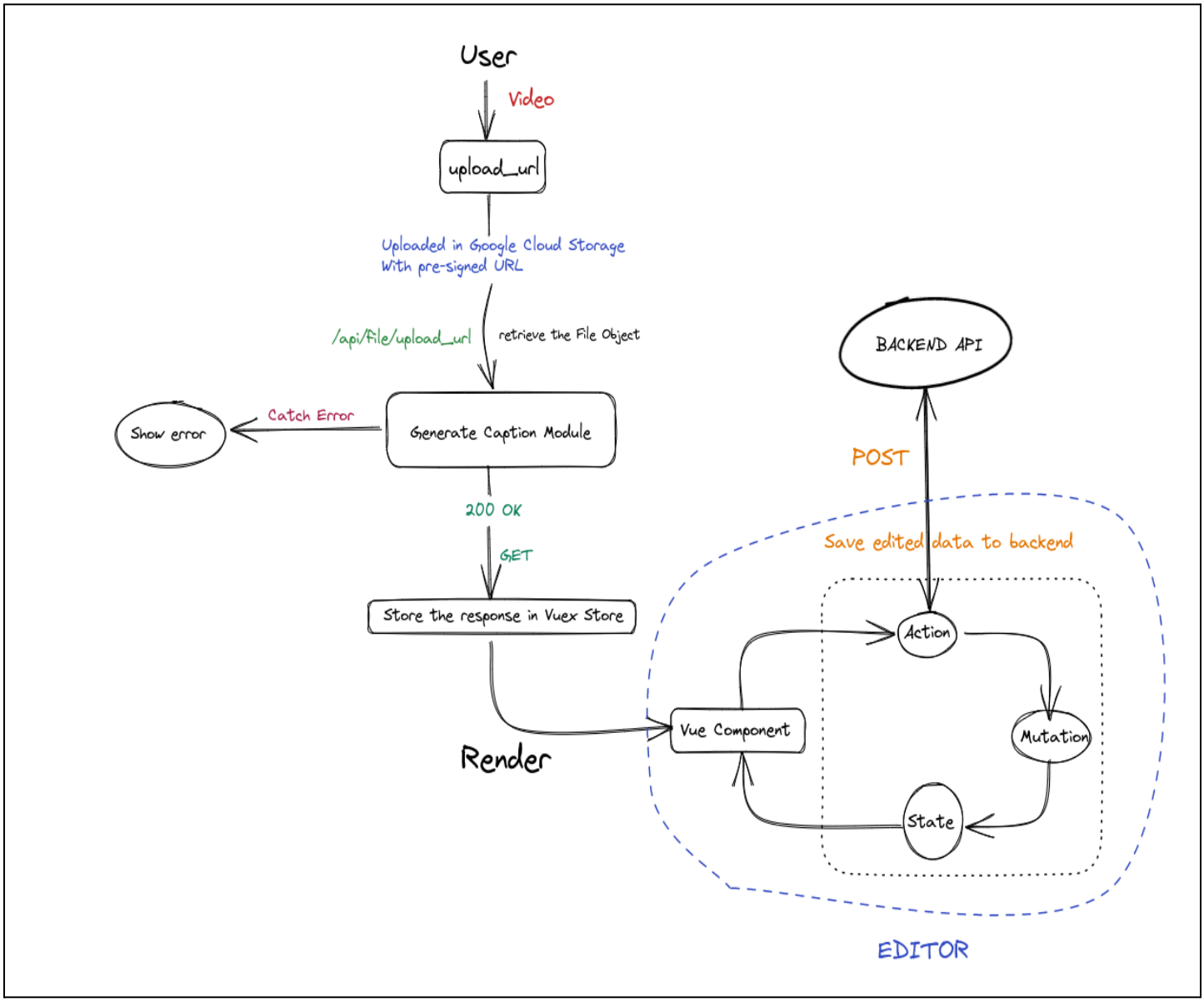
Initially, if we have fewer requests for autogenerate captioning features, we could use the **base** or **small** model which requires less computing resources — as the demand increases we can shift gears to **medium** or **large** models.

If we want to run the self-hosted service in Google Cloud VM Instance, the price can range from **$0.010 hourly +** additional **GPU** cost for **2 vCPU, 1GB memory** up to **$0.040 hourly + GPU** cost for **2vCPU, 4GB memory**. We can go even higher as the traffic increases. If Kolibri Studio already has a VM Instance running that has enough spare resources (idle resources) we can use that which would reduce the cost to zero. I am including GPU because the documentation says "**Required VRAM**".

> *> The hourly rate includes running the server on 27/7. I cannot calculate GPU cost because it's not available in my region, for some reason.*

In conclusion, a self-hosted solution with the option to leverage hosting solutions like HuggingFace can provide the ultimate speech-to-text solution for Studio, offering control over infrastructure, data security, and cost-effectiveness. With careful consideration of computing resources and maintenance, we can provide high-quality autogenerate captioning features to our users.

Generating captions for Audio and Video



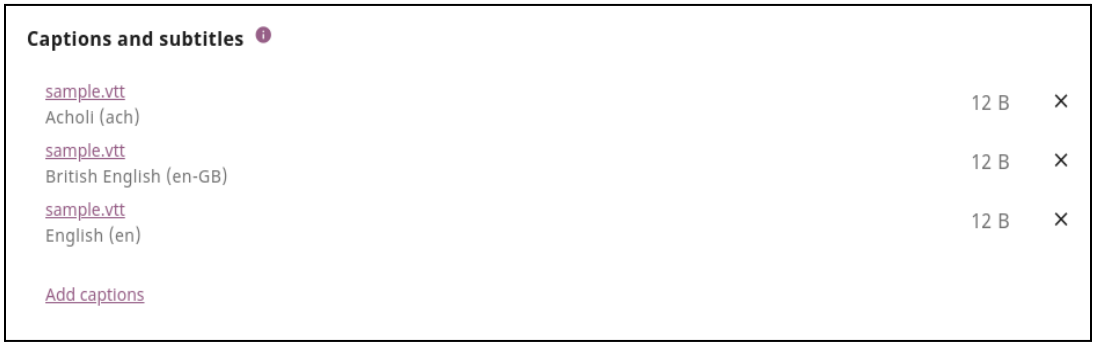
[Link to the diagram]

To create the backend of the project, we need to follow the steps outlined below:

1. Automated Caption Generation for Video and Audio
2. WebVTT File Creation: Ensuring Validity
3. Implementation of Restful API Endpoints for Frontend-Backend Communication
4. The Development of the Editor

Automated caption generation for video and audio

This image shows the currently available feature of uploading captions. It allows the user to upload a WebVTT file.



Instead, users will now see a new button called "**Generate captions**" alongside "Add captions". Clicking this button will ask users the following information:

- The spoken language in the video, which reduces the backend complexity of detecting language, and
- Any request to add additional language for translating the captions, for translation.

The current implementation uses the "**api/file/upload_url"** endpoint to upload the user-provided file directly to Google Cloud Storage, using a pre-signed URL generated by the "**get_presigned_upload_url**" function. The response of "api/file/upload_url" contains a "**file_id"** pointer to the uploaded file and "**uploadURL"** which is the pre-signed URL that can be used to access the uploaded file**.**

To implement the “generate captions” feature, we will create a new model and a ViewSet in “**viewsets/sync/base.py**”. We will create a “GeneratedCaptions” model that contains a foreign key to the “file_id” pointing to the associate multimedia file and a “JSONField” that will store the generated VTT file in a JSON format which can be helpful to render the VTT content in the frontend editor as well as generate a valid VTT file, and a corresponding “CaptionsViewSet”. The CaptionsViewSet will have support for

- CREATED: "create_from_changes"
- UPDATED: "update_from_changes"
- DELETED: "delete_from_changes"

When a user uploads a file and clicks “generate captions”, we will store information in the IndexedDB that specifies “**file_id”** and its corresponding “**uploadURL”**. In the next “**api/sync”** call, we will pass this information (i.e. “file_id” and “uploadURL”) to the “**create_from_changes”** function from our “CaptionsViewSet”. This function will then call the “**generate_transcription**” function which in turn will execute the OpenAI Whisper model to generate the transcription.

The backend on receiving the information of a new task enqueues an async operation with the help of Celery.

Initializing – create_from_changes ViewSet Action

The create_from_changes action will retrieve the binary format of the uploaded file with the help of “file_id” and “uploadURL” and pass it to the OpenAI model. We can place the AI model in the **studio/contentcuration/contentcuration/utils/** folder, and name it something like transcriber.py or caption_generator.py.

This code snippet provides an example of how **OpenAI Whisper** can be used in the backend to generate a transcription for a video:

```
model = whisper.load_model("base")
result = model.transcribe("dummy_files/video.mp4")
```

And the result is


```
{
    'text':" CDC is working to help keep you and your community safe from the threat of a novel or new coronavirus. There are steps you can take now to get ready if an outbreak occurs in your community. Make a household plan. Learn how to prepare and to take quick action if someone gets sick. Older adults and people with chronic medical conditions are at greater risk. Take extra steps to protect them. Think about what you will do if there are changes to your work schedule. And remember to always practice good health habits such as frequently washing hands with soap and water, staying home when sick and covering coughs and sneezes. For more information, visit CDC.gov.",
'segments':[
    {
        'id':0,
        'seek':0,
        'start':0.0,
        'end':6.0,
         'text':' CDC is working to help keep you and your community safe from the threat of a novel or new coronavirus.',
        'tokens':[...],
        'temperature':0.0,
        'avg_logprob':-0.09554368478280527,
        'compression_ratio':1.6453900709219857,
        'no_speech_prob':0.0052330042235553265
    },
    {
        'id':1,
        'seek':0,
        'start':6.0,
```

```
            'end':11.0,
             'text':' There are steps you can take now to
    get ready if an outbreak occurs in your community.',
            'tokens':[...],
            'temperature':0.0,
            'avg_logprob':-0.09554368478280527,
            'compression_ratio':1.6453900709219857,
            'no_speech_prob':0.0052330042235553265
        },
    ...
    }
```

The resulting data will contain additional information, but we are specifically interested in the following:

- **segments**: A list of objects representing each segment of the audio that was transcribed. Each segment contains the following keys:
    - **id**: A unique identifier for the segment.
    - **start**: The start time of the segment in seconds.
    - **end**: The end time of the segment in seconds.
    - **text**: The transcribed text for the segment.
    - **avg_logprob**: The average log probability of the transcribed text.
    - **no_speech_prob**: The probability that there was no speech in the audio during the segment.

For Step 1, the next objective is to **format the data into a JSON object** that contains only the relevant information needed to create a WebVTT file such as the **id**, **start**, **end**, and **text**. This JSON object will then be uploaded to the GeneratedCaptions model.

This creates a level of abstraction to the **generate_transcription** function and enables developers to easily switch to a different model if required in future.

> *If, for any reason during the community bonding period, the mentors decide to change the AI model, only this specific step would require*

modification, and the rest of the proposal would remain unaffected by this decision.

WebVTT File Creation: Ensuring Validity

In the next step, we need to convert the transcription generated from step 1 into a valid WebVTT file. We could create a function called generate_vtt_file. This function will take the "JSONField" value and the "file_id", available from the **generate_transcription** function.

A valid WebVTT file is a text file that follows the WebVTT format, which is a format for displaying timed text in connection with a multimedia presentation. A valid WebVTT file typically has the following features (Mozilla):

- It begins with a header
- Each cue (a timed text unit) is introduced by a cue identifier and a timestamp and is followed by the text content of the cue.
- Cue timestamps are expressed in a specific format: hours, minutes, and seconds, separated by colons and commas, e.g. "00:01:26".

So, the format would be

```
WEBVTT

0
00:01:18 --> 00:01:20
- Hello Learning Equality!
```

We can convert the "start" and "end" values to the **HH:MM:SS** format using python datatime.timedelta.

```
start = datetime.timedelta(seconds=seconds)
```

This repeating format can be easily created with a python script.

```
WEBVTT

id
Start --> end
- Text

id
Start --> end
- Text

id
Start --> end
- Text
```

Many AI models that transcribe multimedia files such as videos and audio, including Mozilla **DeepSpeech** and OpenAI **Whisper**, provide timestamps in their output.

Finally, after the successful generation of the WebVTT, we can save it as a file blob to Google Cloud Storage.

Implementation of Restful API Endpoints for Frontend-Backend Communication

To facilitate communication between the frontend and backend of the application, we must create API endpoints in the "***contentcuration/contentcuration/urls.py***" file and define the necessary views for each endpoint in the "***/views***" directory. The first API endpoint, which will be a GET request, will be used to populate the frontend editor with the JSON object stored in the "GeneratedCaptions" model. We will query the PostgreSQL database with the "file_id" obtained from the frontend to retrieve the corresponding JSON object.

When the user submits changes to the frontend editor, we need to update the captions file in two locations: the JSONField in the "GeneratedCaptions" model and the stored .vtt blob in Google Cloud Storage.

Frontend Save Action Handling

Upon clicking "save," the form will submit a JSON response containing the captions file JSON object with edits. For the first part, we can directly update the JSONField in the "GeneratedCaptions" model.

To ensure that any updates to GeneratedCaptions trigger the "**generate_vtt_file**" function, we will set up a system whereby a Celery task is enqueued for any "update_from_changes" made in the model. The function will be responsible for updating the file in Google Cloud Storage.

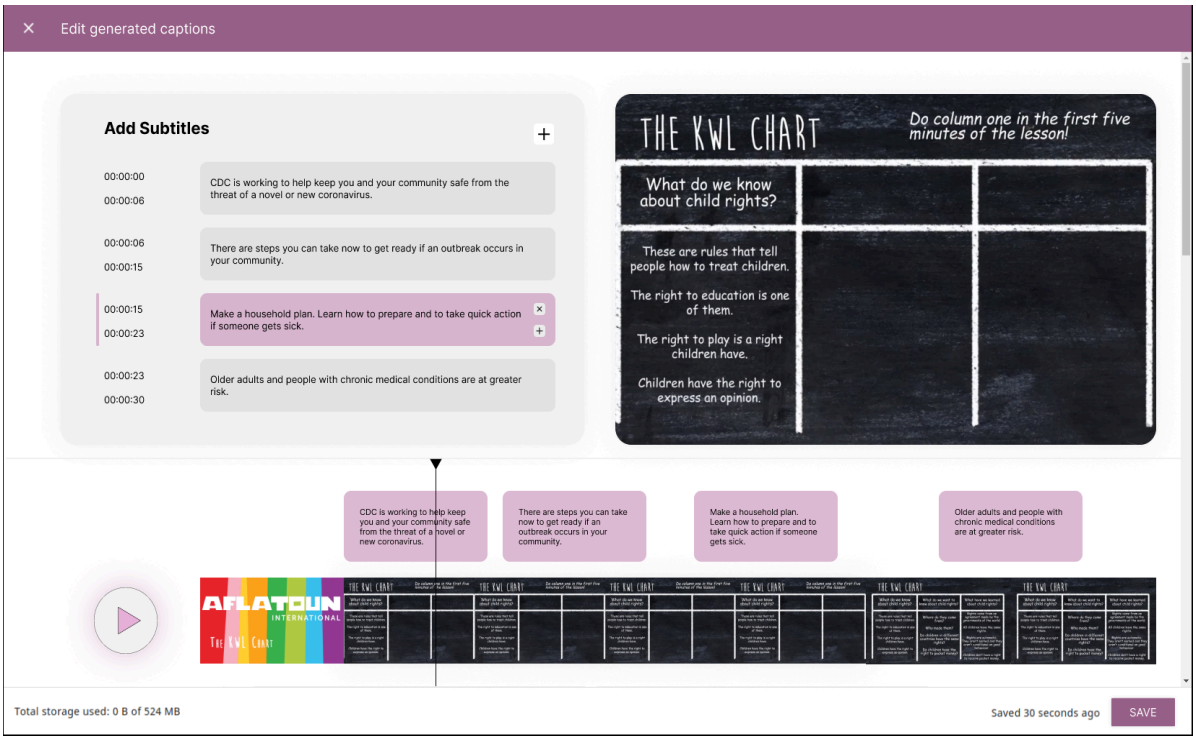
The Development of the Editor

Now that we have a way by which we can communicate from frontend to backend. I will try to explain what we need to do in the frontend, specifically in the **ChannelEdit** Single-Page-Application (SPA).

We will need to create a new component in the “component” directory for the caption editor. This component will contain all the necessary form inputs and buttons for creating and editing captions.

Add a new route for the caption editor in the “router.js” file in the “views” directory. This will allow the user to access the caption editor component. The link to this component will be visible after either uploading the captions (allowing users to edit the already existing .vtt file) or after the “Generate captions” task is successful.

I have prepared a Figma file, which is available for your review at [link]. For optimal viewing experience, please click on the **'Play'** button located in the top right corner of the screen.



How can we make this frontend?

1. First, we create a **time-synced mapping**: Create a mapping of the video duration to the corresponding capti ons. For example, if the video is 2 minutes long, and we have captions at 0:30, 0:45, 1:15, and 1:45, we would create a mapping like this:

```
{
  '0:30' : 0,
  '0:45' : 1,
  '1:15' : 2,
  '1:45' : 3
}
```

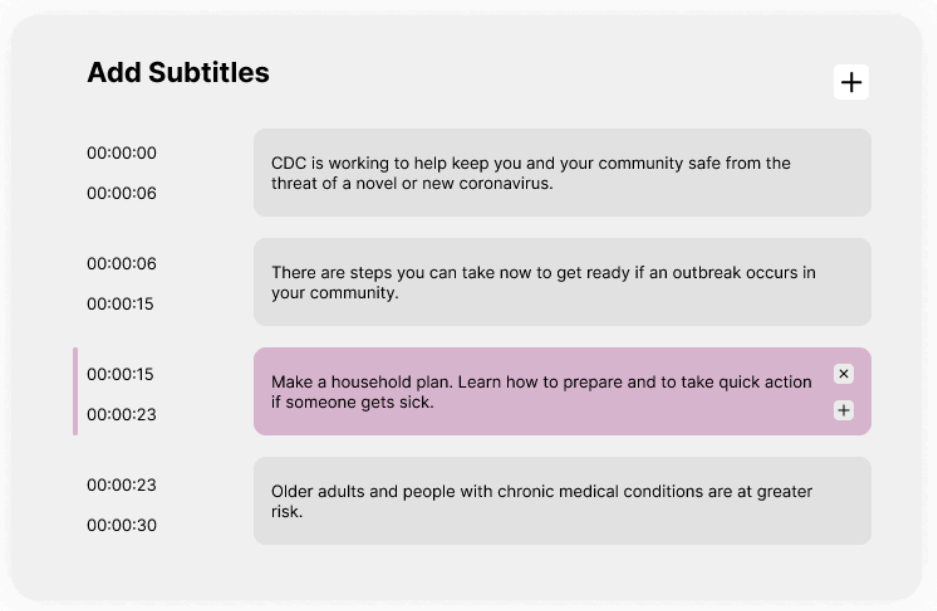
   This mapping will allow us to easily look up the correct subtitle to display at any given time during playback.

2. **Play the video**: We can use a video player library to play the video on the left side of the screen.

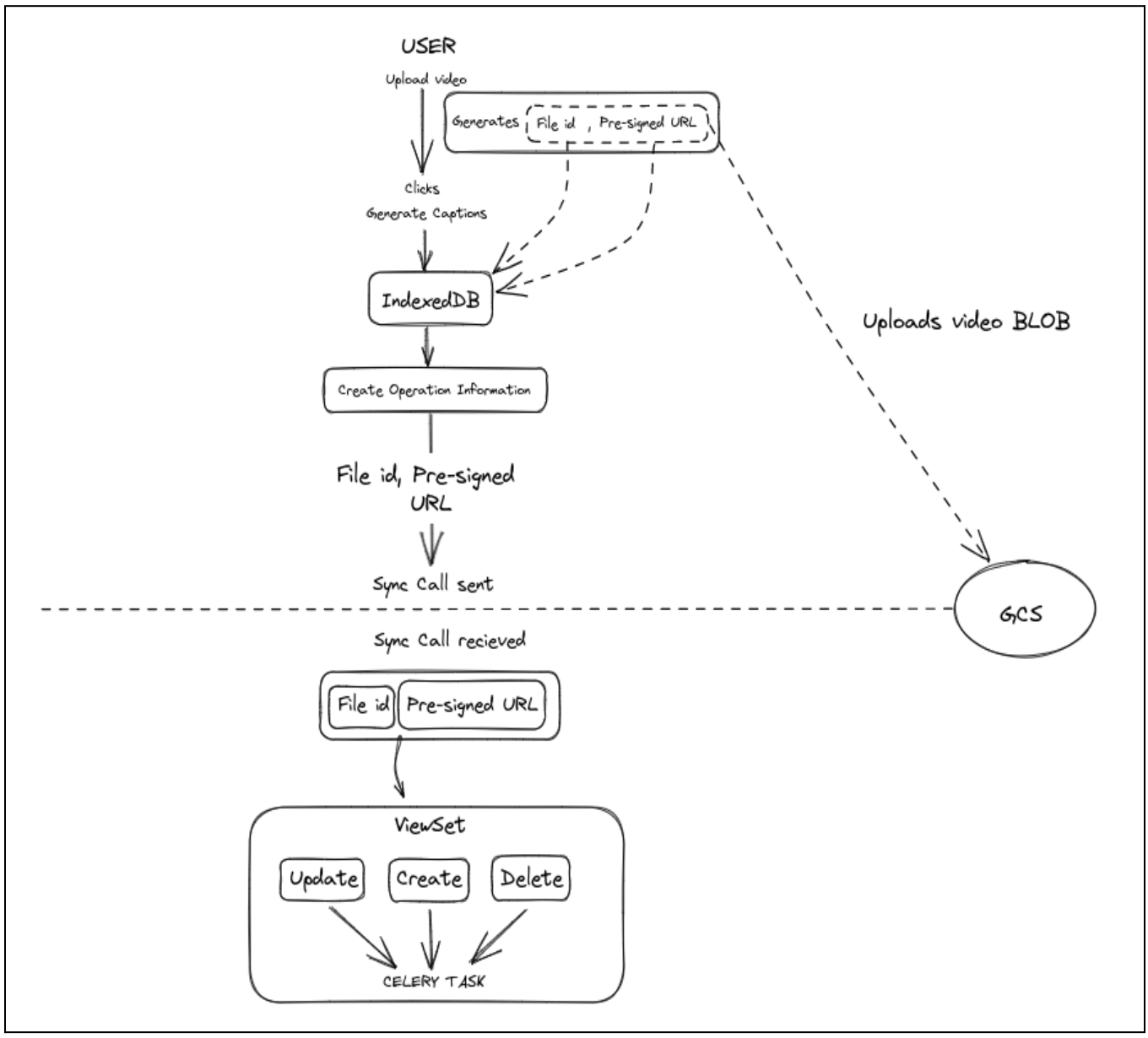
3. **Highlight the current subtitle**: We can add an event listener to the video player to detect when the video is playing and update the display on the right side of the screen with the current subtitle. To display the subtitle based on the current time of the video, we can use the mapping created earlier. To highlight the current subtitle in the caption editor, we can change its CSS style.

4. **Sync the subtitle display**: When the video is paused or seeked to a different time, update the display on the right side of the screen to show the correct subtitle. Again, use the mapping created earlier to look up the correct subtitle to display based on the current time of the video.

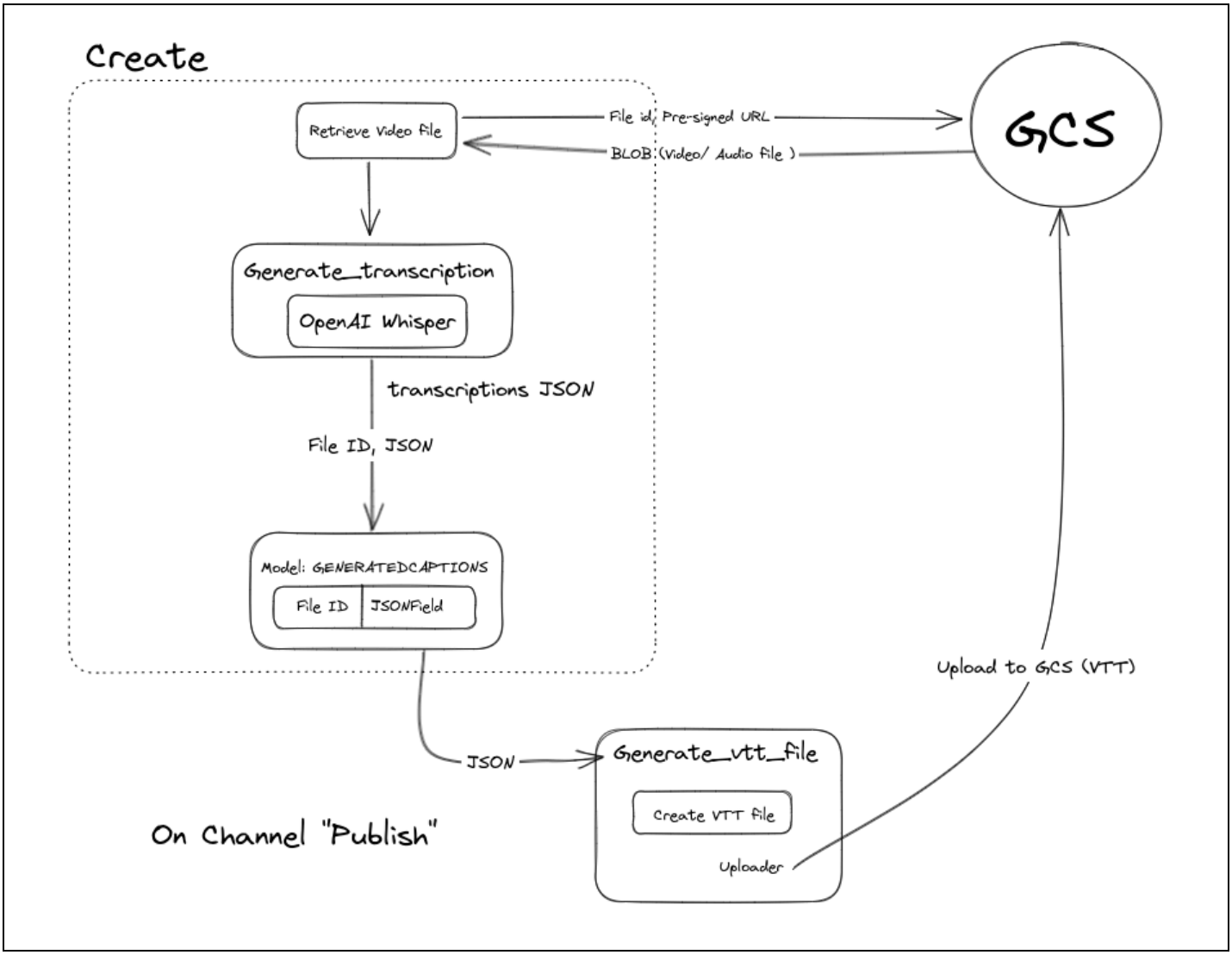
5. **Edit the subtitles**: Each block of subtitles that you see on the left-hand side of the image will be the '<input>' field. This will allow the user to edit the text of the subtitles in the text editor. When the user makes changes, update the subtitle text in the mapping and re-sync the display to show the correct subtitle at the current time.



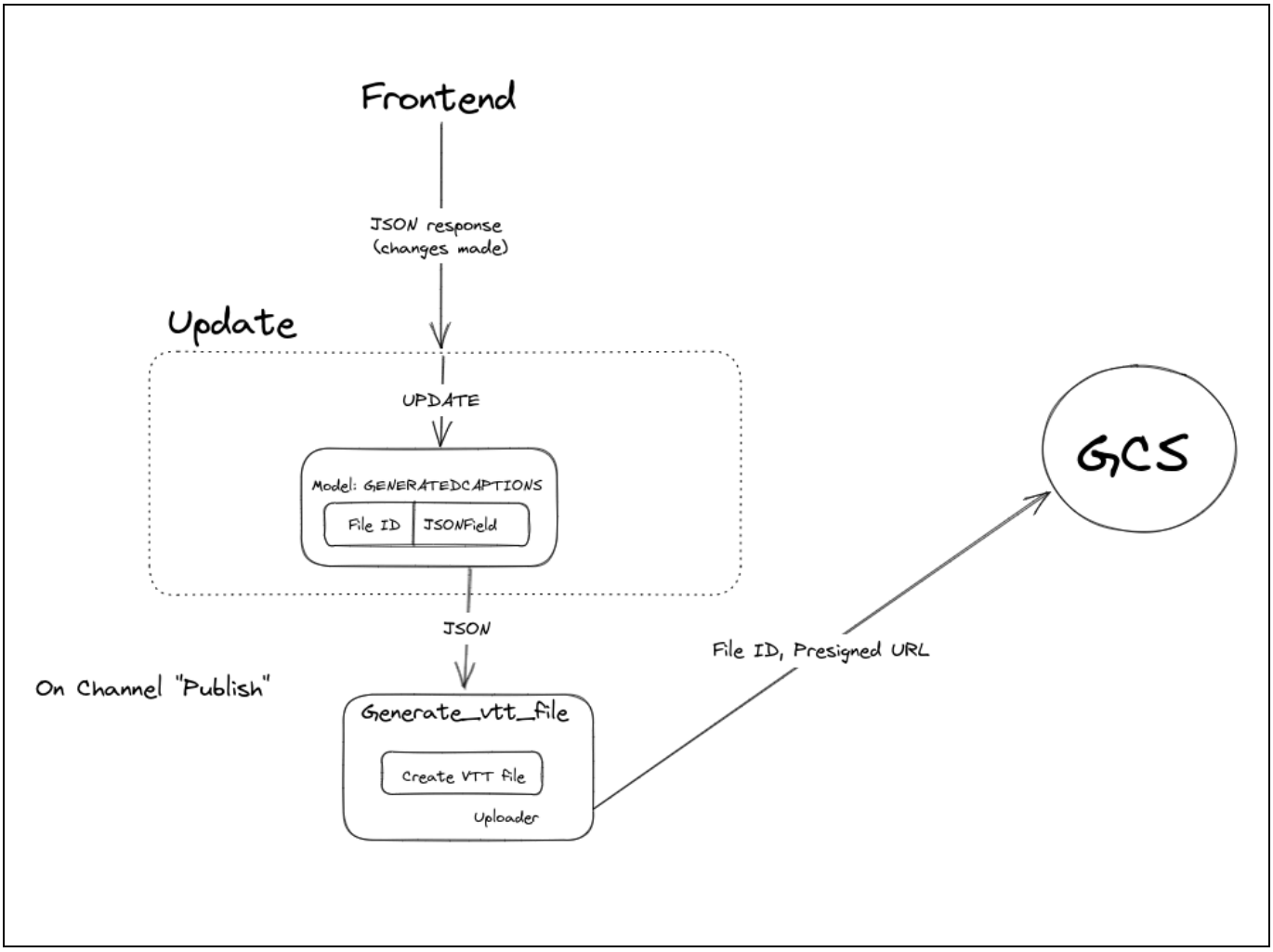
Backend Architecture Diagram



[Link to Diagram]



[Link to Create Diagram]



[Link to Update Diagram]

Timeline

Period	Task
After proposal submission [April 4 - May 4]	- Study more about Django Rest Framework. - Study celery. - Learn about async behaviour with celery task - Continue contribution by solving issues and staying connected with mentors.
Community Bonding Period [May 4 - May 28]	
Week 0 and Week 1 [May 4 - May 21]	- Meet with mentors and request which part of the proposal needs work. - Investigate and research the part of the project which requires the most work. - Understand the edge cases. - Fix any performance issues in the architecture. - Improve and finalize the complete backend architecture.
Week 2 [May 22 - May 28]	- Finalizing the frontend architecture. - Discuss the testing

First Coding Period [May 29 - July 14]	
Week 3 [May 29 - June 4]	- Set up the project with all configurations required. - Kickstart with the implementation of the transcriber.
Week 4, 5 [June 5 - June 18]	- Development of the transcriber using AI Model and related libraries. - implementing audio file handling, speech-to-text transcription, and error handling.
Week 6 [June 19 - June 25]	- Test until satisfactory result. - Implement error handling and exception handling for transcriber - fine-tuning AI model.
Week 7 [June 26 - July 2]	- Complete the function Generate VTT File, to generate WebVTT file. - Work on uploading VTT files to Google Cloud Storage.
Week 8, 9 [July 3 - July 14]	- Start with the implementation of the Django Model and its “viewsets”. - Implement the utils functions like JSON generator for the transcriber.

First Evaluation Date [July 10]	
Second Coding Period [July 14 - August 21]	
Week 10 [July 14 - July 23]	- Work on the implementation of CREATE viewset action
Week 11, 12 [July 24 - August 6]	- Implementation of UPDATE and DELETE viewset action of the Django models
Week 13 [August 7 - August 13]	- Start working on the GET request of the REST API to populate the editor. - Retrieving JSON file from PostgreSQL.
Week 14, 15 [August 14 - August 27]	- Start working on the POST request of the REST API to support “save” from the frontend to the backend. - Uploading the new JSON to Model and also to the Google Cloud Storage.

Week 16 [August 28 - September 3]	- Start working on the frontend development using VueJS.
Week 17, 18 [Sept 4 - Sept 17]	- Continue working on the frontend development and integrating it with the backend - Implement the user interface for managing transcriptions
Week 19 [Sept 18 - September 24]	- Working on the implementation of populating the frontend editor. - Testing the “save” endpoint. - Work on the implementation of the Translate viewset, which will translate the transcription.
Week 20, 21 [September 25 - October 8]	- Test the entire system and fix any bugs or issues. - Optimize the system for better performance and scalability.
Week 22, 23 [October 9 - October 22]	- Prepare the system for deployment to production. - Deploy the system to the production environment and test it thoroughly.

	- Provide user training and documentation for the web application.

About Me

I, Akash Verma, am pursuing a B.Tech in Computer Science from India. My aspiration is to become a proficient backend programmer. I have a keen interest in physics, mathematics, and astronomy, among other subjects. My coding journey began with open-source contributions, and I enjoy exploring open-source projects.

I also enjoy working with artificial intelligence and backend projects.

Proficiencies and Experience

I am skilled in **Python**, **JavaScript** and **C++** language and possess a good knowledge of data structures and algorithms. I have always been fascinated with backend programming and have completed various useful projects.

I had the privilege of being mentored by **Audrey Roy Greenfeld** and **Daniel Roy Greenfeld** authors of the famous Django book "**Two Scoops Of Django**" and creators of the project **Cookiecutter**. Under their guidance, I worked as an intern at Feldroy.com on a project called **BookBuilder** to automate the book-building process using Python for their upcoming book titled "**A Wedge of Django**" (a newer version yet to be released). This project aimed to take one or more Google document files, pull them down to a server, and render them as PDF and ePUB. I also worked on a project titled **Voice-helper**, which was a minimum viable product that utilized artificial intelligence (OpenAI GPT-3) to improve voice recognition software integration with Discord and/or other desktop apps. Here is a demo.

I received a letter of recommendation from both authors.

I also worked for their NGO called Margarita Humanitarian Foundation and started a project called HelpMeSpeak.

Soon after, I was selected for Google Summer of Code 2022, where I worked on the Helioviewer-project. My project was to create a Python package for helioviewer.org, which is successfully hosted in PyPI and is used by research students and professors. With this project, I collaborated with astrophysicists and NASA senior software developers, which was once my dream and it was an amazing experience because I love physics and astronomy, as I mentioned earlier.

Why do I choose to work with Learning Equality?

Learning Equality was first introduced to me by one of my dearest friends, Vivek Agrawal, in 2021, and I made some contributions back then. My motivation to work for Learning Equality is that by contributing to an organization like Learning Equality, I can help millions of children in their learning. Millions of people will be using my implemented features, which I can be proud of.

Along with artificial intelligence, I also love projects with backend development. When I read about how Kolibri and Kolibri-Studio are connected, I was surprised by how the different codebases are interconnected beautifully.

Through the implementation of this elegant backend-frontend architecture in Kolibri and Studio. In the 'auto-generate captions and editor' project, I aim to acquire valuable experience in my passion for Artificial Intelligence. Ever since the commencement of my computer science education, I have been captivated by the limitless potential of AI. Given the recent surge of AI research and groundbreaking inventions, my curiosity to delve deeper into the field has only intensified.

My Contributions

- Studio:
    - Split **deferred_user_data** into two API endpoints
    - Admin Page Editor Link Fix
- Kolibri:
    - Fixed text-align: right for rtl languages
    - Fixed view-as-grid and as-list button active state

Plans after GSoC

I have a strong interest in software architecture and its scaling and I am eager to contribute in any way I can to the organization's future plans and implementation of new features. I find my mentors' work very inspiring and I would be thrilled to continue learning and gaining experience under their guidance and expertise.

How do I plan to stay on track and finish the project successfully?

I will study Celery docs, research more about OpenAI Whisper's increased performance repositories available on GitHub, and solve issues in Kolibri Studio. I will also study methods to test the accuracy of the model with the available public Kolibri database.

How much time will I devote to the project?

I am able to commit four to six hours per day to the project.

Other commitments during summer

I have no other commitments during the summer, only Google Summer of Code.