INTELLIGENT INTERPRETERS

Agile Project Management Methodologies

Group 7

Saptarshi Kundu

Jonie Caisip

Johari Garcia

Yasaman Mehralian

Krish Shah

Table of Content

Introduction

Not understanding what the other person is saying can always be annoying, whether it is due to a disability (deafness) or not knowing one's mother tongue.

The findings show that mother tongue interference has impact on the writing and spoken of English language among secondary school students. Another finding shows that mother tongue hinders effective communication among students in class. It also shows that, Mother tongue influence students' academic performance.

Problem Statement

The problem statement chosen for this project is to create a tool to Identify language, gender and subtitles for people who are deaf or hard of hearing and help people who does not know the language being spoken in the audio.

In this project, we provide transcription and translation of the audio to help both categories mentioned in the introduction.

Agile Methodology

In software development, agile practices include requirements discovery and solutions improvement through the collaborative effort of self-organizing and cross-functional teams with those customer(s)/end user(s), adaptive planning, evolutionary development, prompt delivery, continual improvement, and flexible responses to changes in requirements, capacity, and understanding of the problems to be solved.

Popularized in the 2001 Manifesto for Agile Software Development,[5] these values and principles were derived from and underpin a broad range of software development frameworks, including Scrum and Kanban. For our project, we decided to use the Scrum development framework.

While there is much anecdotal evidence that adopting agile practices and values improves the effectiveness of software professionals, teams and organizations, the empirical evidence is mixed and hard to find.

Roles

- **Developers**

The development team are the people that do the work. You may think the "development team" Means engineers. But that is not always the case. According to the Scrum Guide, the development team can be comprised of all kinds of people including designers, writers, programmers, etc.

In this project Yasaman and Krish played the role of Developers.

- **Stakeholder**

In this project Johari played the role of Product Owner.

- **Product Owner**

Agile teams are, by design, flexible and responsive, and it is the responsibility of the product owner to ensure that they are delivering the most value. The business is represented by the product owner who tells the development what is important to deliver. Trust between these two roles is crucial.

In this project Jonie played the role of Product Owner.

- **Scrum Master**

The Scrum Master is the role responsible for gluing everything together and ensuring that scrum is being done well. In practical terms, that means they help the product owner define value, the development team deliver the value, and the scrum team to get to get better. The scrum master is a servant leader who not only describes a supportive style of leadership but describes what they do on a day-to-day basis.

In this project Saptarshi played the role of Scrum Master.

Scope

The development period of this project takes 2-3 weeks.

Resources

This project is managed in Jira software. Jira is a software application used for issue tracking and project management. The tool, developed by the Australian software company Atlassian, has become widely used by agile development teams to track bugs, stories, epics, and other tasks.

We also have an existing model that predicts the language and gender of the speech in the audio.

Goal

The goal of this project is to create a tool to identify language and gender for people who are deaf and provide transcription and translate of audio for someone who could not recognize the speech properly.

Sprint 1

Sprint Planning

Date: Feb. 12, 2022

User Story	Task	Owner
User needs a flask app which can capture audio from their microphone and save it into a file	Get Flask app up and running	Krish
	Create a function that can access user's microphone	Jonie
	Create function that can record audio from user's microphone and save it to the server	Saptarshi
	Create a function that can read an audio file and pass it to the ML model	Johari
	Test the Flask app	Yasaman
	Create Github repo	Johari

Sprint Goal: Create a Flask app for recording user audio and a Github repo for the team to collaborate on the project.

Timeline: Feb. 14 – Feb 17

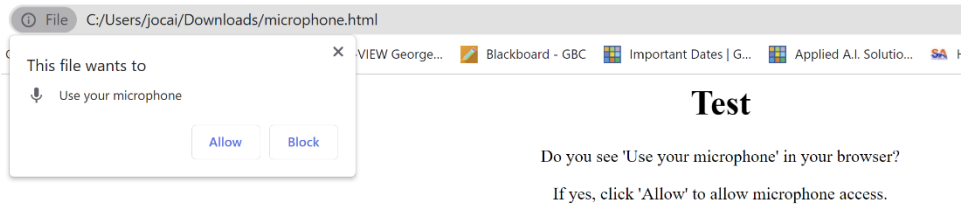
Tasks:

1. Get Flask app up and running:

   The packages for Flask and request are first installed. Then Flask is imported, and the methods are set. Flask server is then created with localhost using the default port 5000.

2. Create a function that can access user's microphone:

   In this task, the getUserMedia API from Mozilla is used to set a prompt to for microphone access to receive audio input. We set it to only get audio with ({audio: true}). This will display a popup showing an option to Allow or Block the use of the microphone.

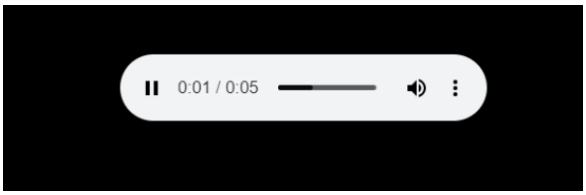


3. Create function that can record audio from user's microphone and save it to the server:

   This function will take the audio stream from the microphone after the user has allowed access in their browser. A Record button is set for the start recording function, and similarly for the Pause and Stop button it will pause and stop recording. Next, the recorded audio is exported to into a WAV file and passed to a function to create a download link and upload the audio file into the specified folder in the app files.

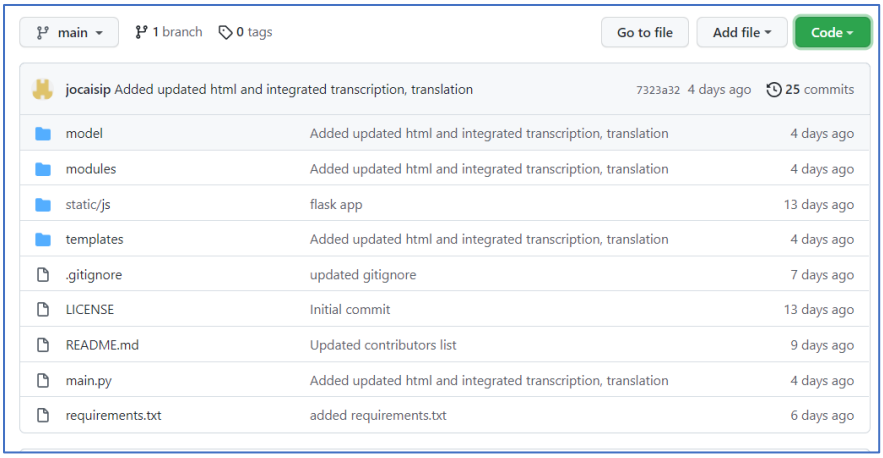
4. Create a function that can read an audio file and pass it to the ML model:

   In this task, we create a function that can read an audio file using response library. This is a sample of the output.



5. Test the Flask app:
   After the code was ready, we tested the app's recording functionality and no issues were found.

6. Create Github repo:
   For this task, we created a Github repository where we can all share our code. Github link:
   https://github.com/jgarciaporras/intelligent-interpreters.git



Sprint Stand-Up

Date: Feb. 15, 2022

Updates:

- We have set up the Flask app.
- The app interface and recording functionality is working.
- Github repo has been created for the project.

Blockers:

- Audio recording need to be modified to the requirements of the model.

Next Steps

- Configure the Javascript code to record 10 seconds add timestamp to file.
- Test the app after changes are added.

Sprint Review

Date: Feb. 17, 2022

Evaluation

In this sprint, the goal was to create a Flask app that can record the user's audio and save it into a file.

Tasks were divided among the team and updates were discussed during the stand-up.

At the end of the sprint, the team has completed the tasks and created a working app that can record audio.

Demo

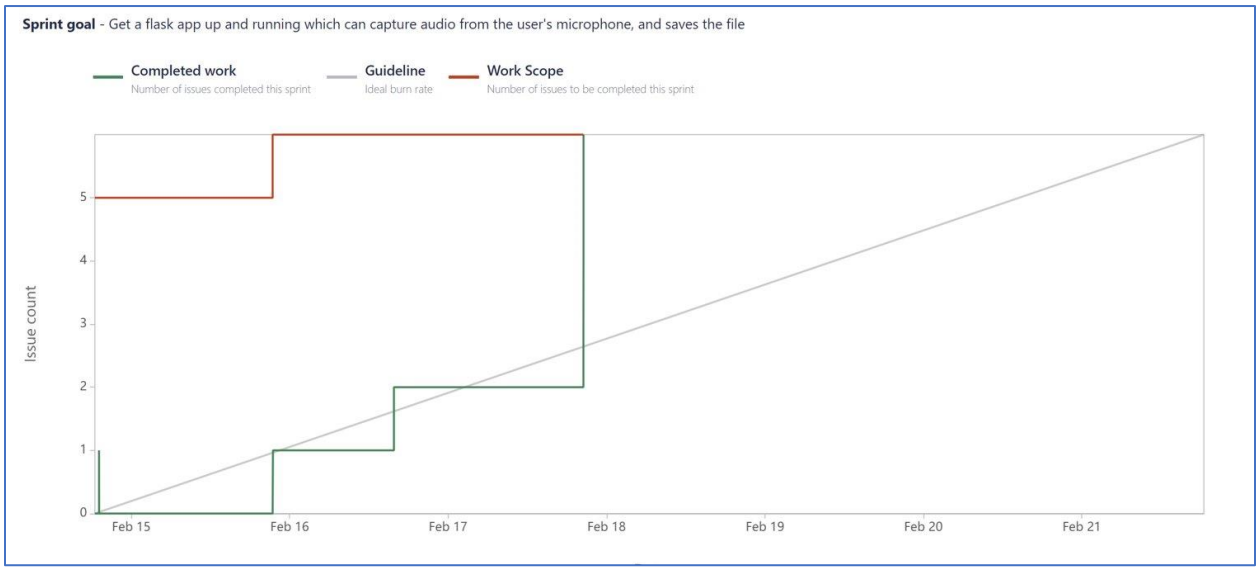
This is a sample demo of the recording app:

Feedback

Audio is being captured and recorded as intended and only small tweaks are left for the file name and record length.

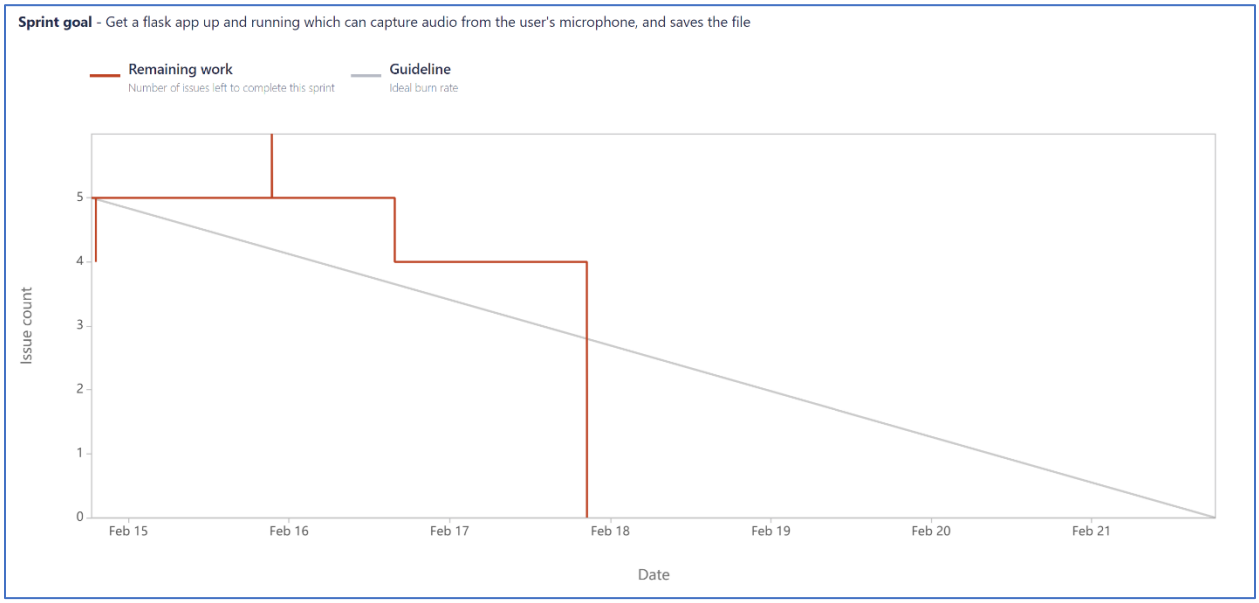
Results

Burnup Chart



For sprint 1, we used issue count in the chart because story points were not added to the tasks. The issue count represents the number of tasks in the sprint. When a task is done, the completed work line increases. The date for the sprint was originally set for a week, but the team has completed the tasks in four days and the sprint was completed early. We were also able to reach the ideal burn rate in this sprint.

Burndown Chart



The burndown chart shows an increase in issue count on the second day as a sixth task was added for the Github repo. Tasks were under development in the first two days and completed in the third and fourth day, where the issue count was decreased. Since this sprint is represented only from the count of tasks, the average amount of work completed may not be estimated properly.

Sprint 2

Sprint Planning

Date: Feb. 17, 2022

User Story	Task	Owner
User wants to read the audio into the app and return the prediction.	Divide ML Model Code for detecting gender and language into functions	Jonie
	Create a package for the ML Model which includes the functions for gender and language	Jonie
	Integrate ML model package for detecting gender and language with the Flask app	Yasaman
	Modify the Front-end to display output language and gender	Krish
	Integrate speech to text code with the Flask app	Saptarshi
	Research Google's translate API	Johari

Sprint Goal: Build functionality for predicting on the audio recording.

Timeline: Feb 18 – Feb 22

Tasks:

1. Divide ML Model Code for detecting gender and language into functions:

   The code for the model we are using contains functions for processing the data into features and retrieving the model's prediction. We take this existing code and divide it into separate functions for preprocessing input data and getting the prediction of the model for the data.

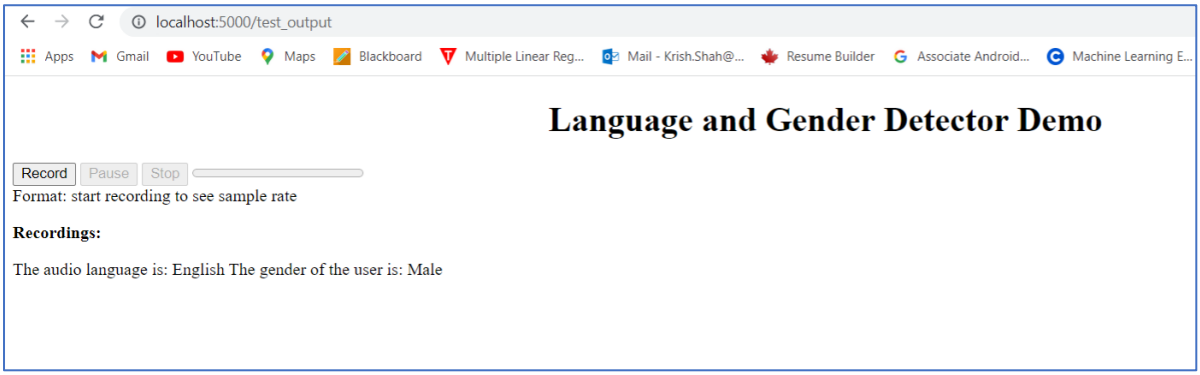
2. Create a package for the ML Model which includes the functions for gender and language:

   We put the functions for preprocessing and prediction into packages that can be called in the Flask code. The preprocess_sample.py package contains the function to take the audio file and return the features which will be used as input for the model. The load_model_and_predict.py package loads the trained model from the file path and predict takes the features and model and returns the predicted label.

3. Integrate ML model package for detecting gender and language with the Flask app:

   The functions from the python packages are imported to the main python file with the Flask app and a variable for preprocess_sample and prediction is set which calls these functions and adds the input file and file path.

4. Modify the Front-end to display output language and gender:
   The home page html file was modified to dynamically display the prediction of the model. In this case the language and gender predicted by the model are displayed on the home page.



5. Integrate speech to text code with the Flask app:

   The module created has a couple functions. One converts the recorded WAV file into MP3 (better for performance with Google Cloud Speech to Text API). The next one takes the converted MP3 file, passes the file and the required parameters to the cloud API, and accepts the response, which is the transcription. The parameter for the API includes the language as well, so we are good on that front. I have used PyDub for the WAV to MP3 conversion.

6. Research Google's translate API:

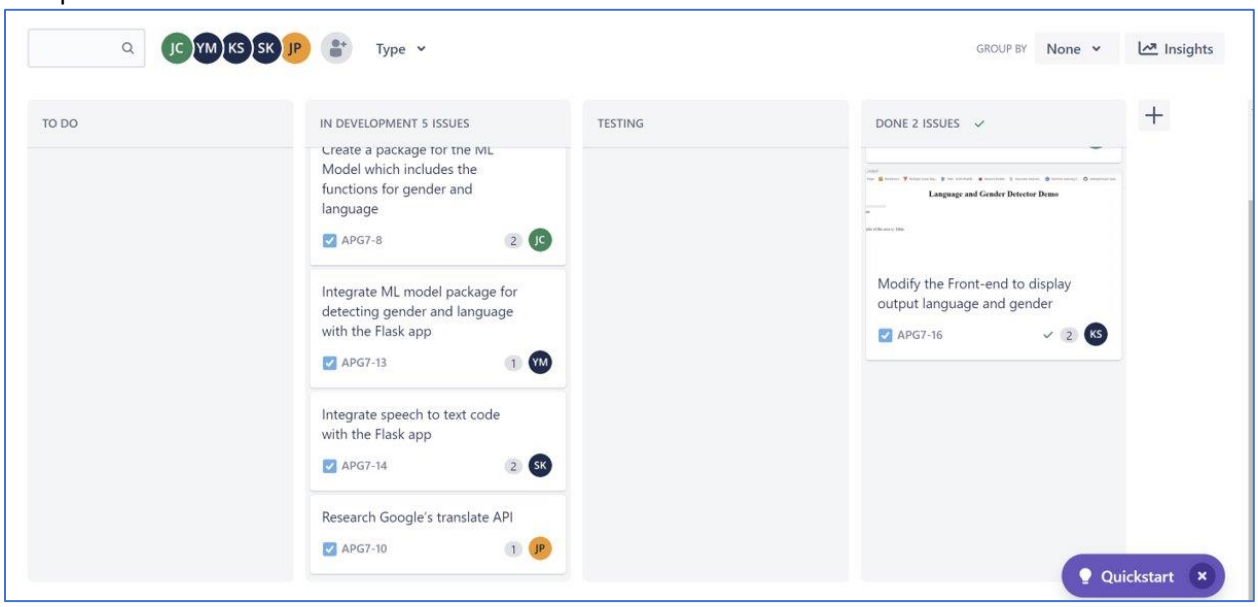
   Google translation API is going to be used in the next sprint. So, we need to know if the Google API has the option to set up the translation language. Actually, it has the option to add it in the function as a "dest", but the default is "English".

```
from googletrans import Translator

translator = Translator()
result = translator.translate('Mikä on nimesi', src='fi', dest='fr')

print(result.src)
print(result.dest)
print(result.text)
```

Output:

```
fi
fr
Quel est votre nom
```

Sample view of the Jira board:



Sprint Review

Date: Feb 22, 2022

Evaluation

For Sprint 2, we created the packages and coded the integration and front-end of the Flask app to get the predictions on the recording. Tasks were assigned after the sprint planning meeting and the team got to work.
Since this sprint's schedule fell on the weekend, there were no stand-up meetings. Individual tasks are submitted, and the code pushed into the Github repository. The packages and code are then put together for the prediction functionality.

Demo

Integration for model prediction:

```
@app.route('/predict', methods=['POST', 'GET'])
def predict():
    if request.method == "POST":
        if request.form.get('predictaction') == 'Predict':

            prediction = load_model_and_predict.predict_sample(features_file, model)
```

Feedback

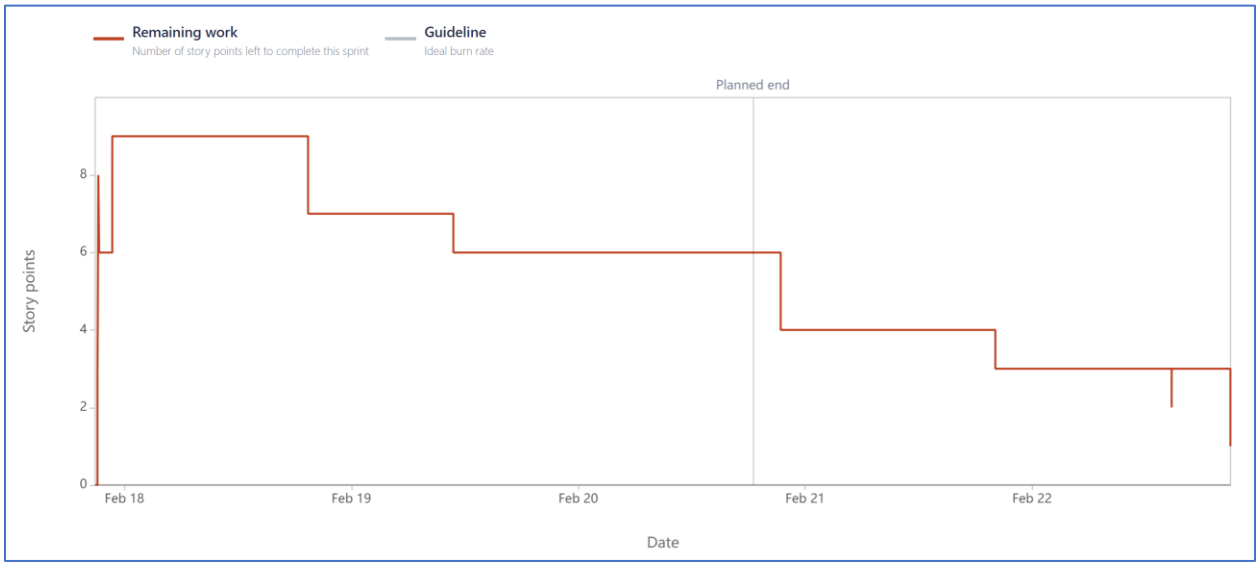
The Code and variable names need slight modification. Integration of the Flask app with packages and front-end is needed for testing.

Results

Burnup Chart

The burnup chart for sprint 2 is measured in story points as the tasks are now assigned with story point estimates. The chart shows a gradual completion rate for the first two days. We did not reach the ideal burn rate for the remaining days as the sprint end date has been slightly adjusted. One task was moved to the next sprint for testing and all other story points were completed.

Burndown Chart



At the start of sprint 2, the story points are set, and the amount of work left is shown. The midpoint for story points completion was reached on the fourth day, with the remaining work for the code integrations completed on the last two days.

Sprint 3

Sprint Planning

Date: Feb. 21, 2022

User Story	Task	Owner
User wants to transcribe audio from the app and return the translation	Create function that takes model's predicted language as input language in Google speech recognition API to transcribe text	Yasaman
	Write a function that takes transcribed text as input in Google translate API and outputs the translation English (or chosen language)	Krish
	Integrate google translate API function with the Flask app	Jonie
	Modify front-end to display option for transcribing speech and translating speech	Johari
	Research Cloud Platforms for our use case (CUDA preferred)	Saptarshi

Sprint Goal: Build functionality for transcribing speech from audio and translating the text with Google translate API.

Timeline: Feb. 21 – Feb 23

Tasks:

1. Create function that takes model's predicted language as input language in Google speech recognition API to transcribe text:

   This is the transcription of the speech in the audio, and we will use the speech recognition library by Google to create this function. The input language for the is taken from the language predicted by the model and passed to the recognize function. The output returned is the transcribed text.

2. Write a function that takes transcribed text as input in Google translate API and outputs the translation English (or chosen language):

   After obtaining the transcribed text, we import the google translate API library and create a function that calls the translate function and pass this text. The source language is also the predicted language and translation is set to English. We then return the translated text in the output.

3. Integrate google translate API function with the Flask app:

   The function for getting the translation using the Google translate API is added in the translate package, and we will integrate it with Flask to translate the output into English. In Flask, we set a variable *google_translate* which calls the translate function to take the transcribed text and predicted language from the model and translate it into the target language English. The next variable *translation* gets the translated text to be displayed in the app.

```
        #translation language
        source = get_source_language(language)

        target = 'en'

        #translate audio
        google_translate = translate(transcription, source, target)
        translation = google_translate.text

return render_template('index.html', prediction=prediction, transcri
```

4. Modify front-end to display option for transcribing speech and translating speech:

   The HTML front-end was modified to display the output of the variables for the transcription of the audio and the translation in English so far.

5. Research Cloud Platforms for our use case (CUDA preferred):

Researched a variety of different options for deployment. So far, Linode seems to be the best option for our use case, since the GPU servers are cheap,and also it is simple and easy to use. I would recommend it for our test deployment.

Sprint Stand-Up

Date: Feb. 23, 2022

Updates:

- Code has been integrated with Flask and HTML frontend is updated and pushed to Github.

Blockers:

- Errors occurred during testing the app's prediction functionality.

Next Steps

- Modify the functions for the prediction and the integration code in the Flask app.
- Retest the app after modifications.

Sprint Review

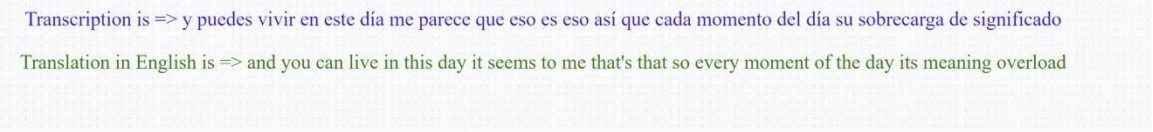
Date: Feb. 23, 2022

Evaluation

Functionality to transcribe and translate audio speech was integrated and tested.

Demo

Transcription and translation output:

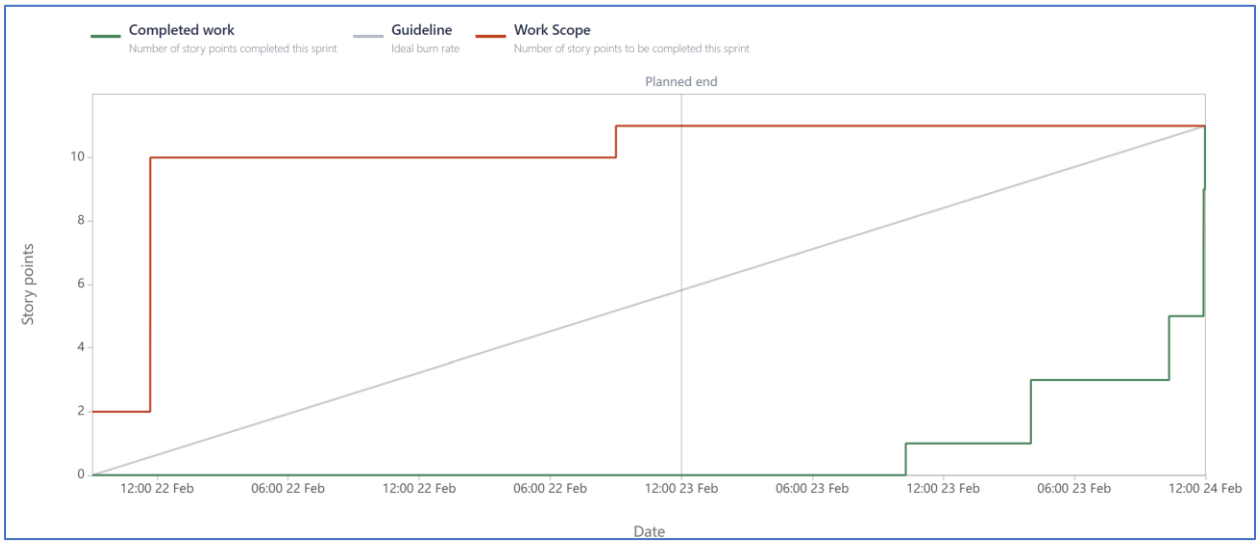


Feedback

There are some limitations with retrieving translations, and the transcription language may be wrong if prediction is wrong.

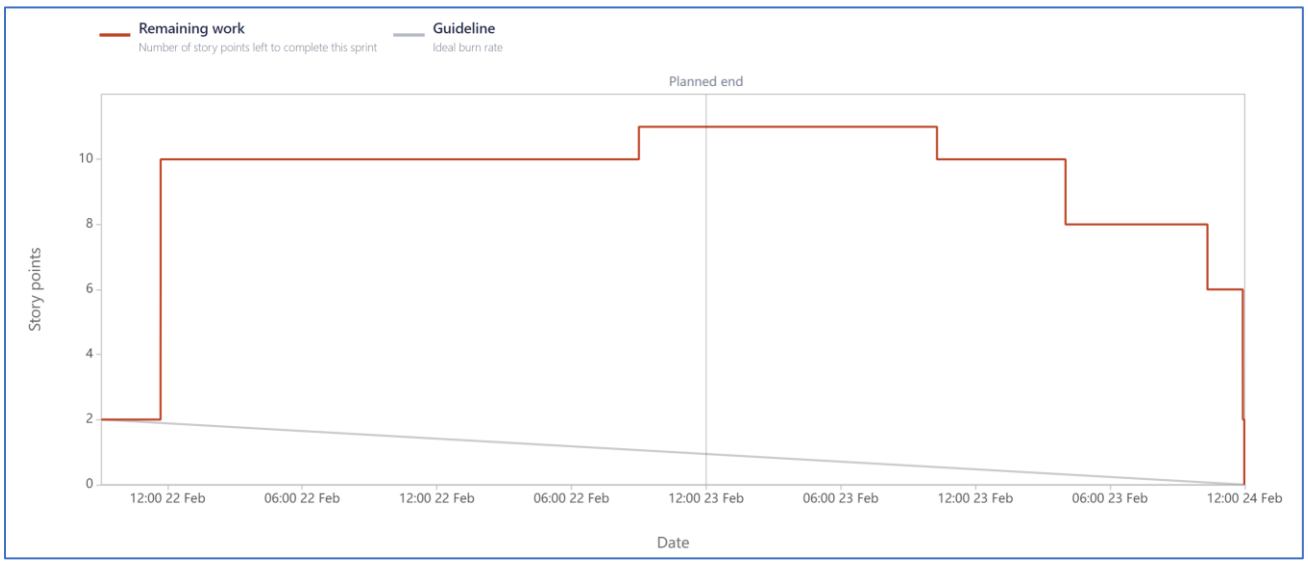
Results

Burnup Chart



The burn up chart is a tool that can helps us to track how much work has been completed during the sprint and show the total amount of work for the spring. The completed work and total work are shown on the vertical axis in story points. The horizontal shows the duration of the sprint, in this case days unit. In sprint 3, most of the work in the story points were completed at the end of the last day so the completed work line increased around that time.

Burndown Chart



The burndown chart is easy to read for stakeholders or any project members. . It represents the remaining work during the sprint and how it has been completed during the sprint. The story points have been added and set during the sprint planning on Feb 22. The tasks were under development for two days, so it does

not match the ideal burn rate. On the end of Feb 23, the work was put together and story points were completed.

Sprint 4

Sprint Planning

Date: Feb. 24, 2022

User Story	Task	Owner
User wants to access the web application in a remote server so they can access it online	Get linode account up and running with GPU server access	Johari
	Create a GPU Cloud Compute instance for deployment	Jo
	Setup the Cloud Compute Instance with CUDA and CuDNN	Krish
	Setup the Python Virtual Environment with the complete stack for the Flask app	Saptarshi
	Deploy the code and setup port forwarding and configure firewall	Yasaman

Goal: Set up server and deploy Flask app into the cloud server

Timeline: Feb 24 – Mar 2

Tasks

1. Get Linode Account up and running with GPU server access:

This was a prerequisite for the rest of the sprint, so it needed to be done first. For this step, we first needed to get a Linode account setup. After doing that, we had to communicate with Linode to activate GPU server access for our account. These servers are available for a limited number of customers only, since GPU servers are in high demand. Our account had a good standing, so we had no issues and within a day our account was set up with GPU server access.

2. Create a GPU Cloud Commpute Instance for deployment:

Once we had access to the GPU servers on Linode, we had to set up a Cloud Compute Instance with GPU access. We needed the GPU servers since our Keras model requires CUDA to run. Since our model was already trained, we only needed the GPU for inferencing, so we went with just one NVIDIA RTX 6000 on our configuration, to keep the costs down. For our OS, we went with Ubuntu 20.04 LTS, since Linux or UNIX in general is known to be more reliable.

3. Setup the Cloud Compute instance with CUDA and CuDNN

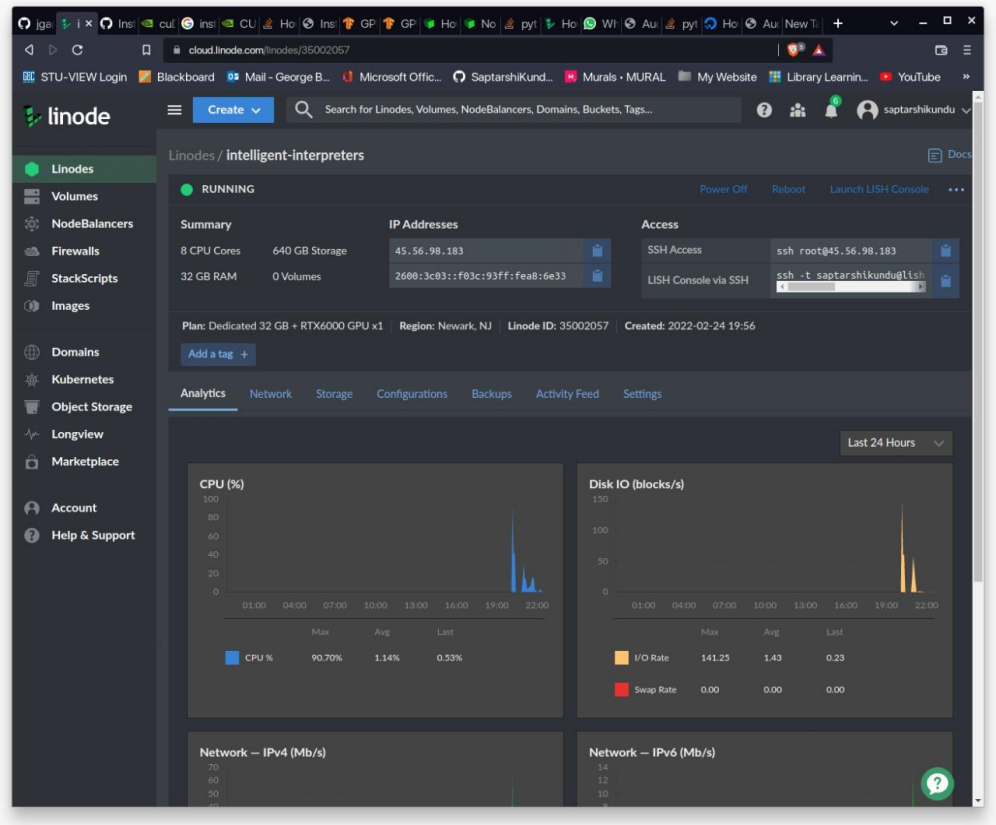
For our Keras model to load and run, we needed to setup CUDA and CuDNN to be setup properly. This task took longer than initially expected, since we ran into various driver related issues, more specifically with the GPUfree drivers, and installed NVIDIA proprietary drivers, then installed the CUDA and CuDNN libraries, and at that point it was working flawlessly.

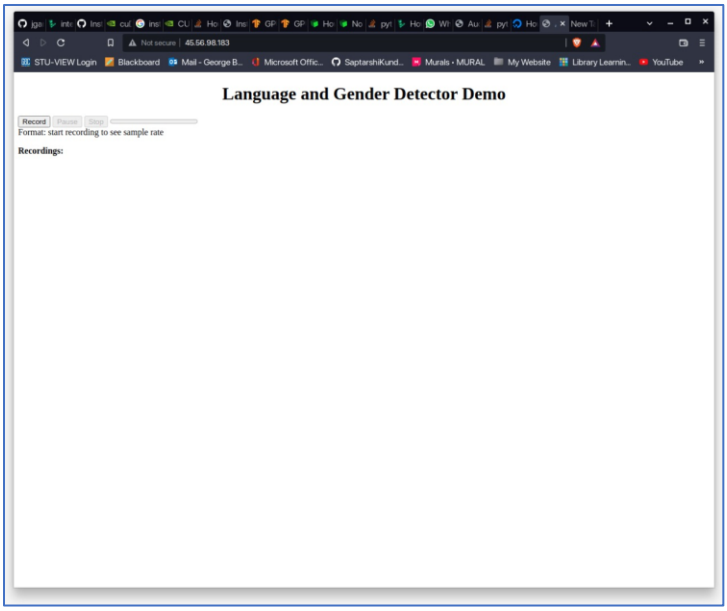
4. Setup the Python Virtual Environment with the complete stack for the Flask app

Once we had CUDA and CuDNN setup, we installed pip and Virtualenv, we created a Python environment for our app to run and installed all the dependencies with their exact versions from our requirements.txt file. For testing, we tried importing all the packages, and everything was working.

5. Deploy the code and setup port forwarding and configure firewall

Once the stack was installed and the environment was setup, we deployed our code by cloning our Git Repo. To deploy the app publicly, we had to setup NGINX and forward the ports from our Flask app to Port 80. This was achieved through a package called Gunicorn, which listened from our flask app and forwarded the response to port 80 on NGINX. Once all of these were setup, we had to forward the ports and configure our firewall for our server. We achived this by setting up network rules using IPTABLES, which is Linux's built in Firewall. Once the Firewall was configured on Port 80 properly, we were able to access the app from any machine on the cloud, using the IP Address of the server.





Sprint Review

Date: Mar 2, 2022

Evaluation

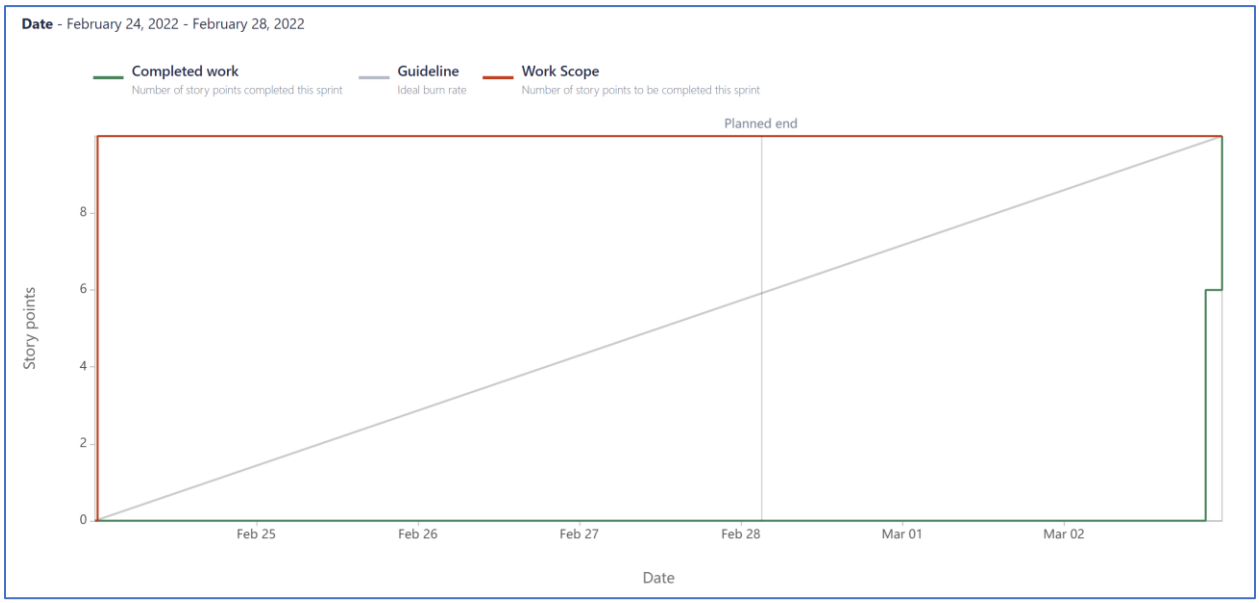
Completed app code is pushed to Github. The server in linode is set up and deployment is done.

Feedback

Deployment is working as expected. GPU credits is limited which shortens the time it can be deployed in the server.

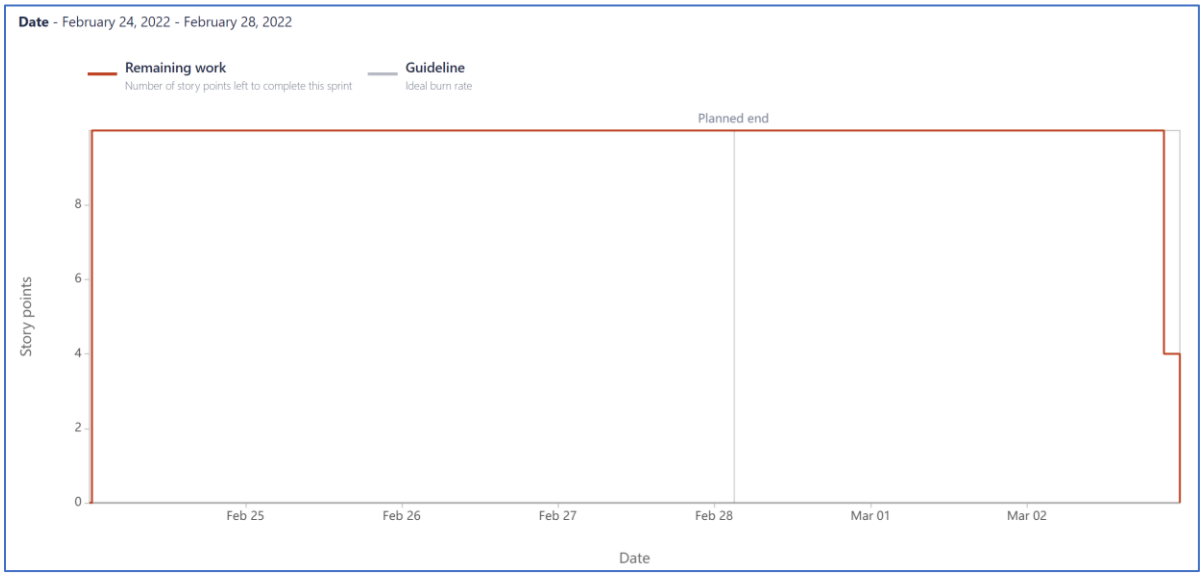
Results

Burnup Chart



Sprint 4 was extended to 1 week so in this burnup chart the planned end shows an earlier date. The tasks for the server deployment follows each task so each has been set to under development until all tasks were completed and the deployment is complete at the end of the sprint.

Burndown Chart



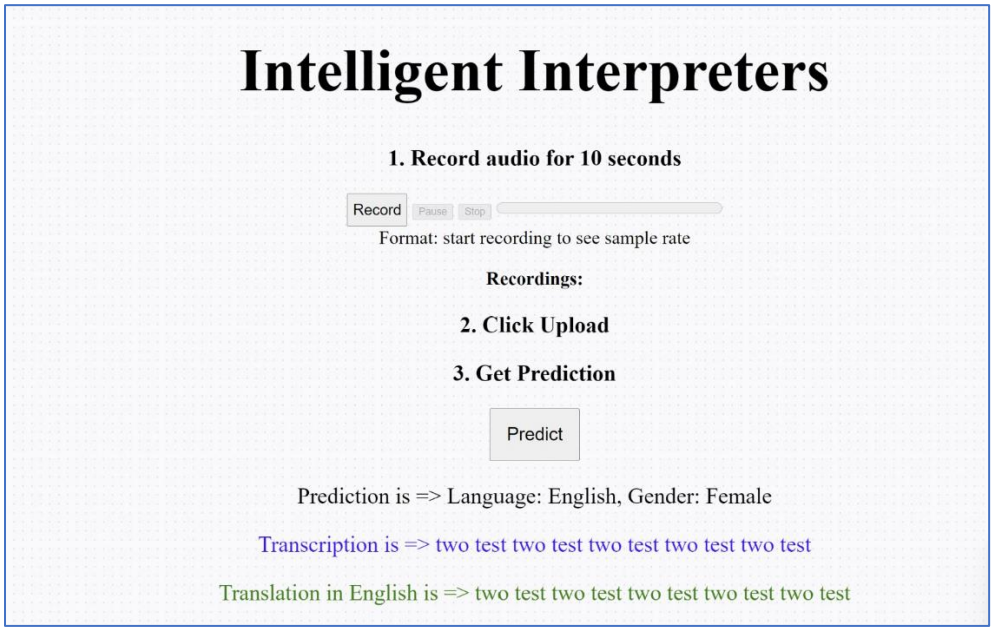
In the burndown chart, the story points have been set at the start of the sprint and the remaining tasks are set to completed when the deployment is done, so the red line remains flat and then decreases at the end of the sprint.

All Sprints

Velocity Chart

This velocity chart shows the velocity in story points with commitment in gray and completed work in green. Sprint 1 does not have a measurement as we did not use story points in that sprint. The highest story points are in sprint 3 with 11 story points and a commitment of 2 story points from the previous sprint. The chart displays the average amount of work the team completes and story points range from 8 to 11 from sprint 2 to 4.

Final Product



This is the interface of the final app design and code. The name of the app is added at the top of the page and then we added the instructions on how to use the app. First the user records a 10 second audio by clicking on the record button. A progress bar is shown in the horizontal line beside the buttons. After recording, the output audio file is displayed as a playback with the option to save and download the audio file or upload. The user can then click on the Upload button to upload the file to the app's files. This is passed on the to function to preprocess and predict on the audio. Lastly, the user can click on the Predict button to display the model's prediction, the transcription of the audio, and the translation into English. Here we can see a sample of the output from one of the tests.

Conclusion

The project was completed over duration of 3 weeks with 4 sprints each sprint being of 4 days, every sprint had daily stand-up call with the team members and each sprint completion was followed by a sprint review and evaluation by the stakeholders. At the high-level goal of sprint 1 was on developing the front end followed by developing and integrating the python scripts for the model prediction, the goal of the sprint 3 was to enhance the project utility by providing transcribing the speech to text and translating the text to the desired language. The final goal of the project was to deploy it to the cloud server which was completed in the sprint 4. All the sprints and backlogs were completed on time along the burn down and other reports in line with the expectations.