Texcellence | Project Report

Songha Ban, Max Belitsky, Rozanne van den Berk, Ngoc Doan, Jakob Hauser, Zen van Riel

November 27, 2020

Contents

1 Problem definition and objectives 2
1.1 Problem Scenario 2
1.2 Proposed solution 2
1.2.1 Breaking down the proposed solution into subproblems 2
1.3 Objectives 2
1.3.1 KPIs 2
1.3.2 Organizational objectives 2
1.3.3 Leading objectives 3
1.3.4 Component (model) objectives 3

2 Data and AI model training 3
2.1 Environment/software 3
2.2 Data 3
2.2.1 Min Sentence length 3
2.2.2 Max sentence length 3
2.2.3 Data Cleaning 3
2.3 Training 3
2.3.1 MT system information 4
2.4 Testing 4

3 Requirements 4
3.1 General: 4
3.2 Functional requirements: statements of services the system should provide 4
3.3 Non-functional requirements: how the system performs a certain function 6
3.4 AI-Specific requirements 7

4 Implementation and version control 8
4.1 Sprint 1: Oct 25th - Nov 3rd 8
4.2 Sprint 2: Nov 3rd - Nov 10th 8
4.3 Sprint 3: Nov 10th - Nov 17th 8
4.4 Sprint 4: Nov 17th - Nov 24th 9
4.5 Sprint 5: Nov 24th - Nov 27th 9

5 Testing 9

6 Migration to the Cloud 11
6.1 Frontend 12
6.2 Backend 12

7 Teamwork 12

1 Problem definition and objectives

1.1 Problem Scenario

Holding effective meetings is incredibly important when it comes to successful communication within a team, and ultimately the success of a project. One factor that can greatly increase the effectiveness of a meeting is adequate note taking. Useful notes should provide a concise overview of future tasks, role division within the team, and more.

However, taking notes by hand during a meeting inconvenient and can be distracting. Furthermore, sharing English meeting notes with Dutch partners or clients is hindered by language barriers. This creates a necessity for an application that takes notes and translates them automatically, to ease the process of both note taking and sharing.

1.2 Proposed solution

Texellence is an action planning speech-to-text system that addresses the aforementioned problem. It is an AI empowered web-based meeting secretary with that provides transcription and translation functionality. With the help of this service, users are able to instantly create and translate short notes, such as to-do lists, either during or after the meeting.

1.2.1 Breaking down the proposed solution into subproblems

- Speech recognition
- English to Dutch machine translation
- Plan future meetings (calendar integration)
- Realize the business case by developing a full system (frontend and backend)

Input: Spoken words
Output: Meeting notes in English with a Dutch translation (side-by-side). The notes can be copied to the clipboard or exported as a follow-up meeting in iCal format.

1.3 Objectives

1.3.1 KPIs

- The adoption of the system by large enterprises (leading objectives)
- User growth (leading objectives)
- Words transcribed and translated per meeting (leading objectives)
- The accuracy and precision of the transcript generated by the speech-to-text model (component (model) objectives)
- The accuracy and precision of the translation generated by the machine translation model (component (model) objectives)

1.3.2 Organizational objectives

Simplify the meeting process; taking notes automatically, instantly and accurately.

1.3.3 Leading objectives

A large number of (satisfied) users and a high level of usage per user.

1.3.4 Component (model) objectives

High speed and high accuracy for the transcription and translation models.

2 Data and AI model training

2.1 Environment/software

Our Github is at: *Texcellence*

2.2 Data

The source data refers to sentences in English, whereas the target data refers to sentences in Dutch. We chose to combine EuroParl (all 2 million sentences for source and target) and OpenSubtitles (the first 2 million sentences for source and target, filtered as per the data alignment section), so our model would be trained with both formal and informal dialogue. We wanted the system to be able to work with both, as it can differ greatly from company to company whether formal or informal language is used during meetings. We want our system to be usable for any company.

EuroParl contains, for both the source and target 2027447 sentences in total, while OpenSubtitles contains 37200618 sentences.

2.2.1 Min Sentence length

EuroParl - English: First Part (2 tokens), Dutch: Proficiat (1 token)
OpenSubtitles - English: It changes (2 tokens), Dutch: Assistent. (1 token).
Note: we set the threshold for the shortest sentence to at least 11 characters.

2.2.2 Max sentence length

EuroParl - English: 3951 characters, Dutch: 3890 characters
OpenSubtitles - English: 5046 characters (lots of full stops, likely noisy data), Dutch: 1168 characters.

2.2.3 Data Cleaning

We sampled a selection of random lines and checked whether they are translations of one another to get an idea of whether the different lines are aligned correctly.

Some sentences have an indication of the name of the person talking, or other artifacts such as '[SINGING]'. We removed these sentences from the corpus.

After the first training run we also did some additional cleaning on the OpenSubtitles: Stripped the data of Sentences with too much punctuation and of instances where the word delta between the dutch version was too high. We then trained the final version of the model.

2.3 Training

We modified your tutorial code to suit our needs. In addition, the summary statistics of the datasets was generated by another script. We created a preprocessing script for the opensubtitles dataset as well. All of these can be found on the GitHub repository.

2.3.1 MT system information

GPU: NVIDIA Tesla K80,
Training time: approximately 7 hours,
Validation perplexity: 7.99733,
Validation accuracy: 57.8886,
Steps: 100000,
Translation time: 1000 example sentences of varying length are translated in 5.8 seconds.

2.4 Testing

BLUE score: 0.072
This BLUE score is based on a collection of 5000 sentences from the OpenSubs dataset. The chosen sentences were not part of the training set, and were cleaned the same way as the training set was.

3 Requirements

Software Requirements Specifications

3.1 General:

An AI-based meeting minute taker/a meeting secretary: a web-based application that recognizes speech and automatically creates and assigns tasks. Users should be able to create "todos" during the meeting or all at once after the meeting. We assume that the system is meant for simple notes not for large input. The source language is spoken English and the target language is written English with Dutch translation side-by-side.

3.2 Functional requirements: statements of services the system should provide

Requirements	User	System
Data	Spoken English audio data will be input and text (data) will be the output. The output will be a transcribed version of the English audio data in text, as well as a translated version of this in Dutch (by providing the original English transcription, translation inaccuracies are avoided). The system will support clear, fluent English with some slight accent.	Audio data (binary, in WAV format) and text data (in clear JSON format) is sent between services to allow for the transcribing and translation of text. To allow for more scenarios, output data can be copied to clipboard and converted in .ical format (a standardised way for calendar invites), which will serve as the next proposed meeting.

Operational	The system will be used during online meetings; compatible with Microsoft Teams and Skype. Notably, if your microphone is already used by such software, the app should be easily accessible using a separate device such as a smartphone.	The system will be able to run as a background process during online meetings as long as the user explicitly starts the recording themselves.
Workflow	User inputs notes using microphone Recorded notes are returned to the user in the form of a bullet list. Each bullet represents a separate recorded note in English and Dutch translation on the side. These notes can be edited in case transcription and/or translation errors occur, or simply if more information should be added. Once the notes are taken, user can add the notes to the next meeting by copying the notes to their clipboard or exporting a meeting to .ical format.	System receives user audio input in binary form (WAV) The system calls the speech to text API with this binary data, returning a transcribed version of the input The system calls a custom translation API (English to Dutch) to translate the transcribed version The transcribed text will be used for The system returns the transcribed text to the user (English and translated in Dutch) In practice, the system workflow will be a bunch of API calls between different microservices. Some of these API calls will respond immediately, other API calls may return a status ID and only return data when certain time-intensive processing is complete.
Assumptions/ Constraints	System is meant for to-do lists and simple notes, so there is likely going to be a maximum recording time limit. The Dutch translation quality will be limited, hence the original transcript is provided to make sure the notes are always readable and the transcripts can be edited. If a user is in the meeting on their phone and has no laptop available, their microphone may already be used by the meeting software and they might not be able to use our system.	NMT model was trained with a limited corpus of general language. It does not know domain-specific jargon.

3.3 Non-functional requirements: how the system performs a certain function

Requirements	User	System
Hardware	Accessible and usable on any modern device with compatible hardware (microphone): Typical web browsers on iOS and Android, Typical web browsers for Windows/OSX/Linux.	The frontend and backend for the app should be hosted in the cloud online 24/7 for maximum availability. The MT model should be trained on a GPU machine.
Software	The application is accessed via a web application on a modern browser. It will be a single page application with simplistic design to ensure that the application does not distract from the meeting that users are attending. We will use well-tested and popular frontend design, such as *Material UI*.	OpenNMT is used for training the NMT model and for inference for English to Dutch translation. The inference process should be scalable and parallelized to allow for more users to use the service. Microsoft Azure Cognitive Services is used for the English voice to text transcription. While this service is very scalable as it is cloud native, we will likely be limited by using the free tier. The system is developed with Python (>=3.7) to align with the OpenNMT model and FastAPI for backend. The frontend will be built using React, allowing us to develop a modern "single page application".
Performance	Good internet connection is required in order to make sure no data is lost. We do not have plans at this moment to save data to e.g. a user account.	System should respond in a reasonable time, $<$ 10 seconds for transcribing and translation. The Azure Speech to Text likely takes the most time and is very dependent on how long the microphone recording is. This non-functional requirement is prone to change as we test our individual components under different loads. OpenNMT inference should take max 1 second. The frontend of the application must be very lightweight and quick to load (<400 ms).
Support-ability Requirements	The service can be accessed in any modern browser and does not need any app installations.	-

Security Requirements	The application can be accessed by anyone, and sessions should never be shared with others. At this moment we decided that it can be accessed without a password, although perhaps we will require users to login via single-sign on to prove we can lock the application behind a specific set of users.	Microphone data should be sent using secure HTTPS. No user data is kept after processing.
Interface Requirements	The user should be able to download the todo list in several formats (.txt, .csv, .ics if scheduled events (use a "Schedule event" button), .org) by pressing a corresponding button.	-
Availability Requirements	24/7	The individual system components should be available to each other 24/7. Realistically, there will be some downtime according to the SLA of the cloud service we will use.
Assumptions/ Constraints	-	There may be constraints on the amount of data that can be processed, as there is no budget to scale out this service indefinitely. Data can easily be lost if e.g. the user quits their browser since user accounts are not in the current scope of the project.

3.4 AI-Specific requirements

Requirements	User	System
Assumptions/ Constraints	-	The NMT model was trained with a corpus of general language. It does not know domain-specific jargon.
Hardware	No requirements. The AI part is only relevant to the back-end server, which the user does not need to worry about.	OpenNMT is used for training the NMT model and for inference for English to Dutch translation. The inference process should be scalable and parallelized to allow for more users to use the service. Microsoft Azure Cognitive Services is used for the English voice to text transcription. While this service is very scalable as it is cloud native, we will likely be limited by using the free tier.

4 Implementation and version control

Our sprint tasks can be seen in each respective project on our GitHub repository.

4.1 Sprint 1: Oct 25th - Nov 3rd

Scrum Master: Jakob

We created a Github repository at *Texcellence* and protected the master branch as to have at least a very simple branching model for feature additions and bug fixes. Due to the unavailability of team members and the possible dates for prospective sprint retrospective and sprint planning sessions we decided that our sprint should start on 27-10 and last until 3-11.

We met on the 27th for a sprint planning session. In this session we made use of the Kanban board integrated in github and created several GitHub issues which we both assigned as tasks for this sprint and left in the backlog for future sprints.

One of these tasks was to settle on a backend framework, in which we are planning to implement the APIs that will be included in the final product. We farmilized ourselves with the APIs by following hands-on tutorials and ran some test cases for the transcribing API.

FastAPI was chosen for the backend for its simplicity in implementing micro applications/services compared to other frameworks. Preliminary tests of five words or less were performed on both the backend mvp and the API transcription. These tests showed that the application (1) transcribes correctly what has been said, and (2) took less than a second to return the transcription in all test cases (due to the speed of Azure Speech-to-text).

Apart from this, an important action on our agenda at this point in time was finding a way to "stream" the audio to our transcribing API. We also needed to figure out how the request between all the included services could be structured optimally.

Our approach to the project, albeit not strictly speaking test-first development at this moment, is rather similar to the approach in nature in the sense that we are adding functionalities one by one, each time testing it with all the test cases we can come up with.

4.2 Sprint 2: Nov 3rd - Nov 10th

Scrum Master: Max

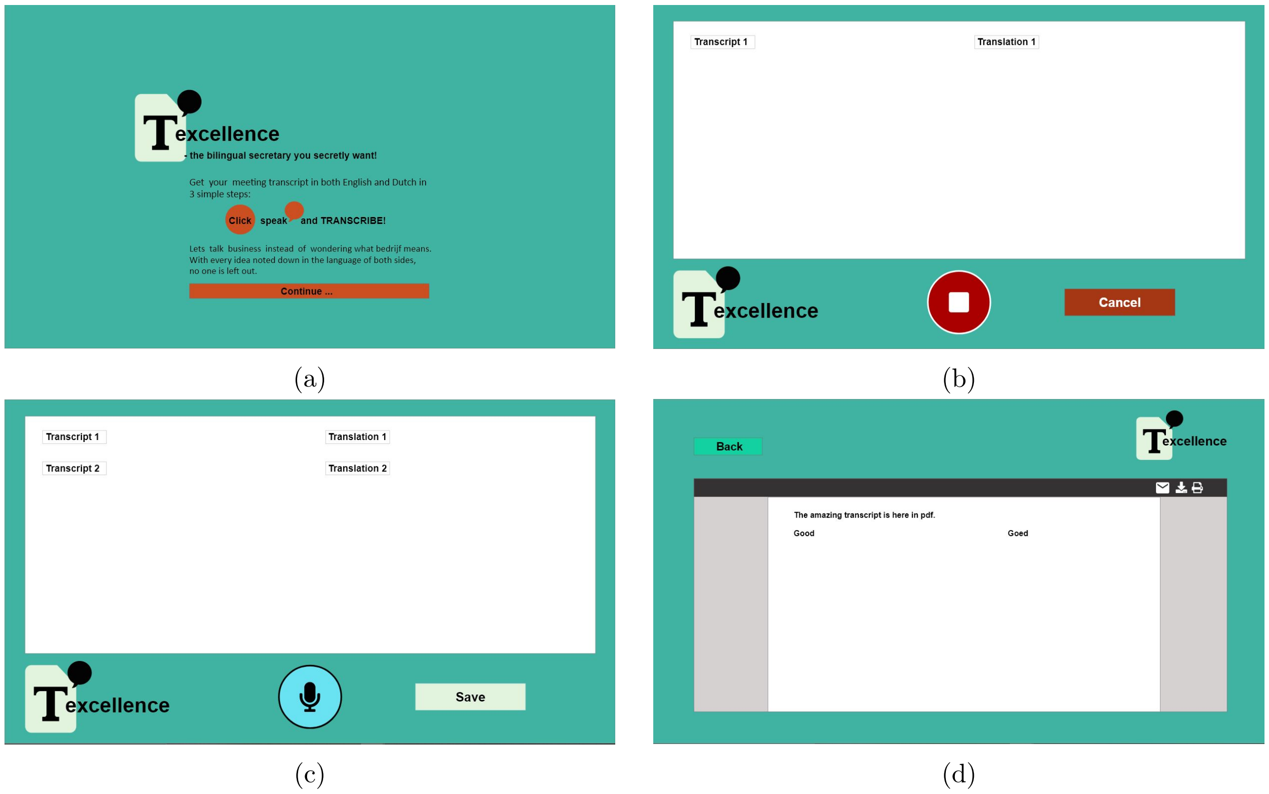
A couple of major goals for this sprint included: scaffolding the frontend and starting work on the React components used for the UI, wiring up the frontend with the backend using the browser's microphone, and adding unit tests to comply with task 5.

During this sprint we finished our low-fidelity prototype in Adobe XD. The process of creating the low-fidelity prototype consisted of three stages: (1) a rough sketch showcasing the global outline of what the application should look like, (2) the first version created in Adobe XD, in which the logo was created and ideas for a front page, loading screen and pdf preview were added, and (3) the final version in Adobe XD in which the ideas outlined in the first two stages were combined into one Adobe XD low-fidelity prototype. Screen captures of the finished low-fidelity prototype can be found in figure 1.

4.3 Sprint 3: Nov 10th - Nov 17th

Scrum Master: Zen

This sprint had several goals: further work on implementing unit testing, creating a single script to start both frontend and backend as well as exploring possibilities of merging the python files into a single module or package, implementing the UI based on the low-fidelity prototype, hosting the backend in the cloud and the frontend in GitHub Pages, and investigating whether tokenization will improve the translation model's performance.



Texcellence

- the bilingual secretary you secretly want!

Get your meeting transcript in both English and Dutch in 3 simple steps:

Click speak and TRANSCRIBE!

Lets talk business instead of wondering what bedrijf means. With every idea noted down in the language of both sides, no one is left out.

The amazing transcript is here in pdf.

Good	Goed

Figure 1: Final version of the low-fidelity prototype of the GUI. (a) The cover page. (b) The application as it is recording. (c) The application as it is not recording. (d) A pdf preview of the notes taken. This function was later replaced with a 'copy to clipboard' function.

4.4 Sprint 4: Nov 17th - Nov 24th

Scrum Master: Songha

The aims in this sprint were for the most part a continuation of the goals set in sprint 3. In addition to this, we investigated whether tokenization will improve the translation model's performance. While the frontend was soon successfully hosted (for screen captures of the finished GUI, see figure [tbd]), the backend experienced a short delay as we only obtained an AWS login this sprint. After the short delay, the backend was successfully hosted during this sprint.

4.5 Sprint 5: Nov 24th - Nov 27th

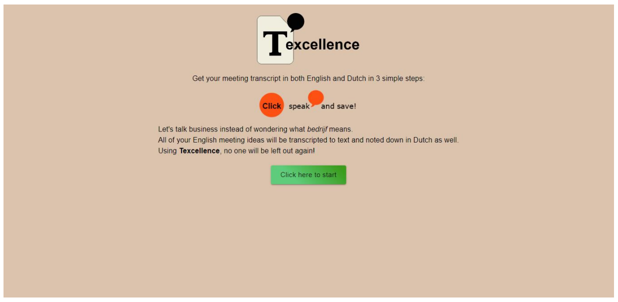
Scrum Master: Rozanne

This sprint was an effort to clean up the code repository, ensure all components remained functional, and to ensure that the final report was of great quality. To assure this, the report was converted into LaTeX(Overleaf) and all members of the team were tasked to refactor the contents.

5 Testing

For this task we have implemented 10 tests of which 4 are integration tests and 6 are unit tests. The unit tests were written for several functions including tokenize(), detokenize(), translate(), create_upload_file():

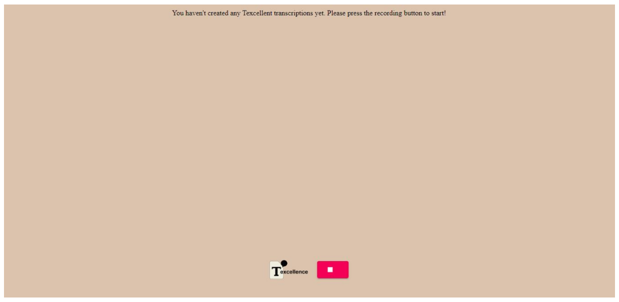
- tokenize():



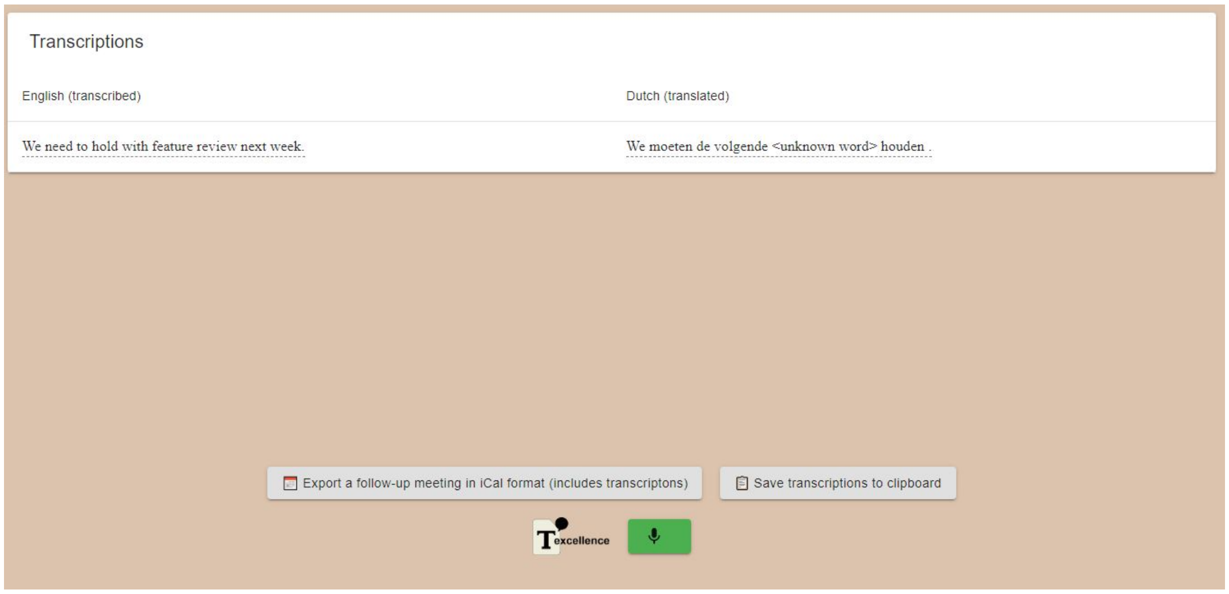
Get your meeting transcript in both English and Dutch in 3 simple steps:

Let's talk business instead of wondering what *bedrijf* means.
All of your English meeting ideas will be transcripted to text and noted down in Dutch as well.
Using **Texcellence**, no one will be left out again!

(a)


You haven't created any Texcellent transcriptions yet. Please press the recording button to start!

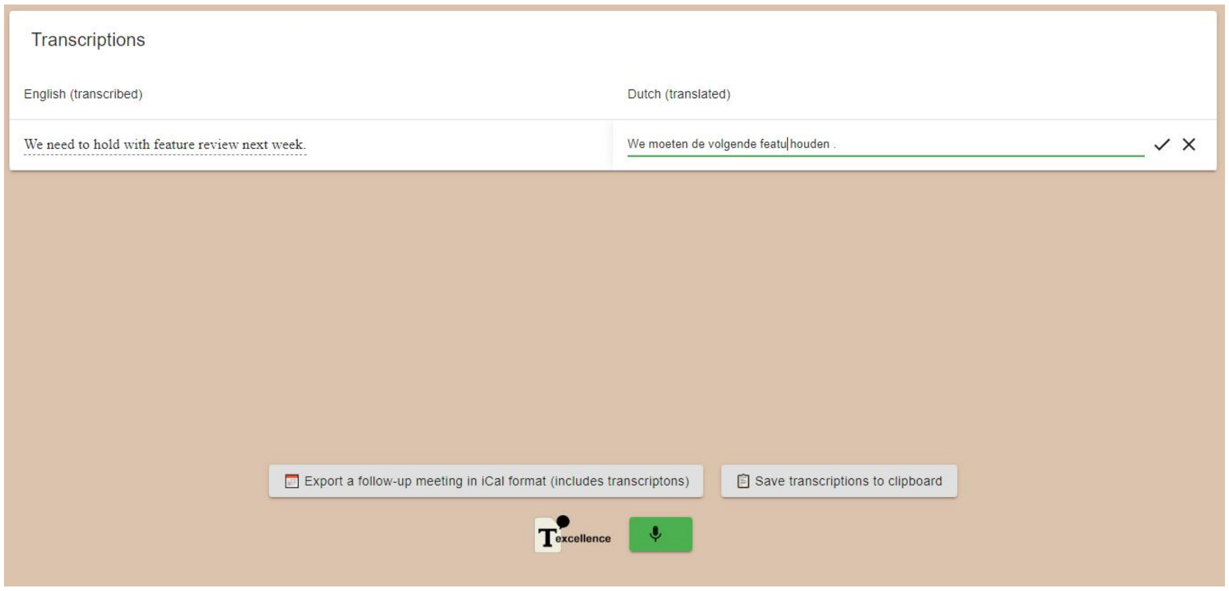
(b)


Transcriptions

English (transcribed)	Dutch (translated)
We need to hold with feature review next week.	We moeten de volgende <unknown word> houden .

Export a follow-up meeting in iCal format (includes transcriptons)
Save transcriptions to clipboard
Texcellence

(c)


Transcriptions

English (transcribed)	Dutch (translated)
We need to hold with feature review next week.	We moeten de volgende featu|houden .

Export a follow-up meeting in iCal format (includes transcriptons)
Save transcriptions to clipboard
Texcellence

(d)

Figure 2: Final version of the GUI. (a) The cover page. The background color was changed to beige so it would be easier on the eyes. (b) The application as it is recording. (c) The application as it is not recording. (d) The GUI has an edit function, so imperfect translations or transcriptions can be corrected before they are exported.

    - tested if the input is tokenized correctly

- detokenize():
    - tested if the input is detokenized correctly

- translate():
    - tested if an exception is raised when the type of the input is other than string
    - tested if there is no exception when unusual characters are passed as an input (unicode emoji and foreign characters)
    - tested if the model returns <unk> for the unexpected input (e.g."dgsgsddgsda")

- create_upload_file():
    - tested if the status code of the GET request equals 405 (since the function only accepts POST requests)

Since the create_upload_file() function implements most of the functionality of the application including the call to Azure Speech API, file uploading and translation, we have performed the integration tests on this function. In order to perform the integration tests we have generated some test data, including empty.wav, test.wav and longtest.wav audio files. In particular, we have:

- tested if the translation is correct
- tested if uploading the empty .wav file raises the HTTPException with the status code 500 (since the empty file produces the empty transcriptions, and the translate function has to receive something as input)
- using the test.wav file as input, we tested if the file is uploaded correctly and passed to the Azure API, if it produces correct transcription of the audio and if the transcription is translated by the model
- in the third test we have done the same as in test 2, but on a long file (longtest.wav, 33 seconds) to test how the system deals with large inputs without errors

It has to be mentioned that our tests have generated 5 warnings (all in the functions where the translation model was invoked). These are the warnings generated by PyTorch because of the deprecation of some functionality that is used by openNMT (*UserWarning: An output with one or more elements was resized since it had shape [5], which does not match the required output shape [1, 5].This behavior is deprecated, and in a future PyTorch release outputs will not be resized unless they have zero elements. You can explicitly reuse an out tensor t by resizing it, inplace, to zero elements with t.resize_(0)*).

6 Migration to the Cloud

Sprint 4: From Nov 17 to Nov 24

6.1 Frontend

We used Github Pages to host our website. The React frontend builds into a static website (pre-made HTML, JS and CSS files). Since it communicates with the backend only through a REST API, this was a logical decision as it saves our time and resources for running a separate frontend server. In addition, we do not have to worry about stability of the website while deploying a new version since we rely on the CDN architecture of GitHub. The front-end automatically builds and deploys on new changes pushed to the master branch by executing a CICD pipeline.

- Website URL: *`https://zenulous.github.io/Texcellence`*

6.2 Backend

For backend hosting, we used AWS EC2. The instance is t2.medium (2v Core, 4G RAM) with Ubuntu platform. We allocated one Elastic IP and associated it with the server. For security reasons we opened only necessary ports, 22(SSH), 80(HTTP), and 443(HTTPS), in the inbound rules of the instance.

We created our own backend server to use the translation model in an optimal environment. The size of the model is not small, and it will take longer if we have to load the translation model on every request. Therefore, we are running the model as a server locally so that the model is loaded at the beginning, and later it returns the translated response faster without loading time. Also, in case the traffic goes higher and the number of requests increases, we can run the multiple models at the same time to handle multiple requests more efficiently.

In addition, we can scale up and out easily by using AWS EC2. It is enough now to have only one backend server running, but in case the server needs to be upgraded to a more powerful machine or be able to handle more requests in parallel, it can be done by just changing the setting or adding more servers in the AWS console. We cloned the EC2 instance and created a load balancing target group for the future work. (The second server is not running now.)

Route53 was used to assign a domain to the server. We simply created a subdomain of a domain we already have to create a hosted zone, created an A record in the hosted zone, and routed traffic to the public ip address of the EC2.

To deploy our code, we used Gunicorn as an ASGI server and Nginx for reverse proxy and SSL handling. Gunicorn starts the FastAPI app on port 8000 by running the main python file. Nginx listens to port 443 and passes requests to port 8000 for API handling. Nginx also listens to 80, but then it redirects to port 443 for SSL encryption. For SSL/TLS encryption, we used *Let's Encrypt* to obtain a free CA certificate. We can easily monitor the web logs by checking Nginx logs, and because both gunicorn and Nginx are running as systemd services, they will automatically start when the server reboots, and in case the backend app unexpectedly exits, gunicorn will manage to keep it running by creating a new worker.

- API URL: *`https://api.texcellence.songhaban.com/uploadaudiofile`*

7 Teamwork

The roles of each team member were as follows:

- Songha: Cloud migration
- Max: Backend and testing
- Rozanne: UX and data cleaning
- Bo: UX

- Jakob: Backend and data cleaning
- Zen: Frontend and model training

These roles were not strict in the sense that each member was only involved in the role listed above, however. They merely give an indication of which aspect each member was most active in. Each member had some part in each aspect of the project.

Good teamwork and communication was ensured by holding weekly meetings, as well as clearly assigning tasks both verbally during the meetings and via GitHub Issues.

One challenge was miscommunication about meeting times. We solved this by allowing members who missed the meetings to assign themselves to tasks using GitHub Issues. Another challenge was weeks in which not all team members were equally available. During these weeks it was important to keep into account the team members' availability while still keeping the distribution of the work fair.

One thing that made this team work well, was that while members with more experience in a certain area often were responsible for the work in that area, less experienced members would still be involved in these tasks so everyone would make optimal use of this learning experience.