applied sciences

MDPI

Article

Auto-Scoring Feature Based on Sentence Transformer Similarity Check with Korean Sentences Spoken by Foreigners

Aria Bisma Wahyutama and Mintae Hwang *

Department of Information & Communication Engineering, Changwon National University, 20 Changwondaehak-ro, Changwon-si 51140, Republic of Korea
* Correspondence: mthwang@cwnu.ac.kr; Tel.: +82-55-213-3832

Featured Application: An Android mobile application which shows the similarity score in percent based on users' recorded voice speaking specific Korean sentences being fed through Speech-To-Text Engine and Sentence Transformer library.

Abstract: This paper contains the development of a training service for foreigners to help them increase their ability to speak Korean. The service developed in this paper is implemented in the form of a mobile application that shows specific Korean sentences to the user for them to record themselves speaking the sentence. The objective is to generate the score automatically based on how similar the recorded voice with the actual sentence using Speech-To-Text (STT) engines and Sentence Transformers. The application is developed by selecting the four most commonly known STT engines with similar features, which are Google API, Microsoft Azure, Naver Clova, and IBM Watson, which are put into a Rest API along with the Sentence Transformer. The mobile application will record the user's voice and send it to the Rest API. The STT engines will transcribe the file into a text and then feed it into a Sentence Transformer to generate the score based on their similarity. After measuring the response time and consistency as the performance evaluation by simulating a scenario using an Android emulator, Microsoft Azure with 1.13 s is found to be the fastest STT engine and Naver Clova is found to be the least consistent engine with nine different transcribe results.

Keywords: Korean speaking proficiency; Speech-To-Text engine; sentence transformer; similarity check; automatic scoring

Citation: Wahyutama, A.B.; Hwang, M. Auto-Scoring Feature Based on Sentence Transformer Similarity Check with Korean Sentences Spoken by Foreigners. *Appl. Sci.* **2023**, *13*, 373. https://doi.org/10.3390/app13010373

Academic Editors: Yun Seop Yu, Kwang-Baek Kim, Dongsik Jo, Hee-Cheol Kim and Jeong Wook Seo

Received: 31 October 2022
Revised: 25 December 2022
Accepted: 26 December 2022
Published: 28 December 2022

1. Introduction

South Korea, officially known as the Republic of Korea (ROK), is a country that is surrounded by other countries that do not adopt the Latin alphabet. South Korea also has an alphabet called "Hangeul", which consists of 10 vowel letters and 14 consonant letters [1,2]. The fact that they use "Hangeul" instead of the Latin alphabet creates a particular problem for a foreigner who lives in South Korea because it means that they have to learn how to read this alphabet and learn new vocabulary, sentence structures, etc.

There are 1.96 million foreigners that have stayed in South Korea as of 2021, which includes 1.57 million long-term residents and 386,000 short-term residents [3]. With the high number of foreigners staying inside the country, South Korea created a test to measure foreigners' proficiency in the Korean language called the Test of Proficiency in Korean (TOPIK) which was first administered in 1997. This test has many purposes, including getting a job or an academic scholarship in South Korea. TOPIK has six proficiency levels, with level 1 being a beginner level and level 6 being an advanced level. According to [4], the TOPIK test is divided into two categories: regular and speaking. The standard category measures reading, writing, and listening comprehension, while the speaking category solemnly measures speaking proficiency. However, the TOPIK speaking test in 2021 only has two sessions, and it is still stated as a pilot operation for 1200 participants per session

in five countries (a pilot session in 2022 is not yet available, and a regular session will be implemented in the second half of 2022). In this situation, the problem is that the TOPIK speaking score will be manually measured by an expert panel, which will take a considerable amount of time to process, significantly when the number of participants is increasing as it will be available for more countries. Therefore, it requires an innovative way to solve the problem.

This paper is driven by NSDevil, a South Korean company that collaborates with the South Korean government to manage the TOPIK test across South Korea. The NSDevil plans to speed up and digitalize the scoring process of TOPIK speaking by developing a system that can generate the score automatically using AI. The motivation for this paper is to help reduce the overall time of the scoring process in the TOPIK speaking test. Therefore, implementing Artificial Intelligence (AI), such as Machine Learning (ML) or Deep Learning (DL), to generate the TOPIK speaking score automatically can help the expert panel to score the test and reduce the whole scoring process time. This idea also became a necessity of this study. The system requires a Speech-To-Text (STT) technology to transcribe the test participants' voices into the text to be fed into an ML or DL model to be evaluated further and then generate the score. Furthermore, one of the system's key features is high mobility and being lightweight. Thus the system has to be light enough to be carried and implemented anywhere without a complex configuration.

With the collaboration with NSDevil, the development of the ideal future system is divided into several phases. The first phase compares the performance of the open-source STT engine to find which STT is optimal for Korean. The second phase, also the main objective of this paper, is to develop the first version of the mobile application using STT engines and the Sentence Transformer framework. The application allows users to record themselves reading a Korean sentence and automatically checks the similarity as a score percentage. The mobile application will be the basis of the third phase, implementing an ML or DL technology to generate scores without needing a similarity check.

2. Related Works

With the significant growth of ML or DL, they can serve many purposes that benefit people, especially in Natural Language Processing (NLP). As a part of NLP, speech recognition is making a machine recognize a spoken word or sentence regardless of the different people, tone, variation, or pitch. As a product of speech recognition, STT is a process that transcribes spoken words into written text. Although STT is synonymous with speech recognition, the definition of speech recognition describes a broader process of speech understanding [5]. The architecture of STT, as stated in [6], is depicted in Figure 1.

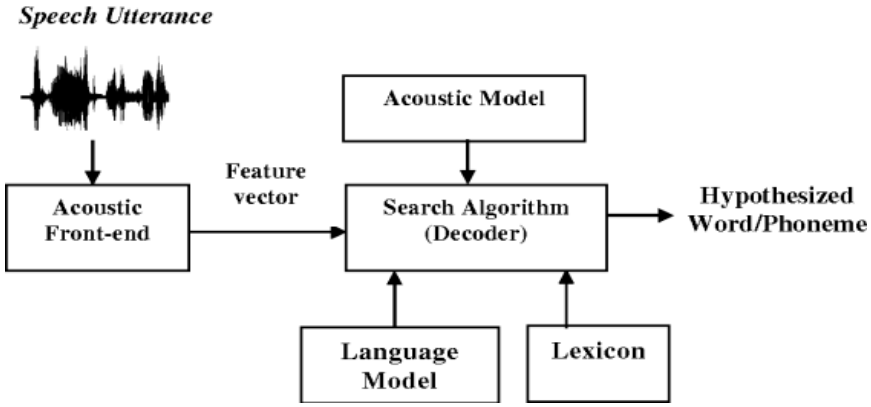

Figure 1. The Architecture of Speech-To-Text.

The STT became well-known to the public around 1990 after the speech recognition algorithm's performance based on the Hidden Markov Model (HMM) was deemed sufficient to provide a plausible transcript, although it is often inaccurate. STT is ubiquitous

and embedded into many applications, from smartphones to cars and household items. Furthermore, STT, especially the well-known open-source STT, could provide a cheap, reliable, easy-to-access, and quick way to automatically transcribe speech data in many languages regardless of gender, fluency, intonation, pitch, and voice [7].

As stated in [8], there are several algorithm approaches to performing an STT:

- Template-based (matching unknown speech with known information);
- Knowledge-based (variations in speech are saved into the system, and the interference system deduces a complex rule);
- Neural network-based (uses a neural network to detect speech based on the training data automatically);
- Dynamic time warping-based (matching sequences of the same speech with varying time or speed);
- Statistical-based (uses automatic learning procedure on large training data samples).

Many related previous works have discussed the use of STT to measure, evaluate, or score spoken sentences [9–15], as shown in Table 1.

Table 1. Previous Related Works Contribution as a Comparison.

Title	Contribution
Research on Open Oral English Scoring System Based on Neural Network [9]	A scoring system for open-spoken English using Neural Network (NN) based on the user's recorded voice at phonetic and text levels. The research separately scores the spoken content and spoken speech, in which the spoken content results from an external speech recognition engine and the spoken speech is scored using an in-house NN model.
A Convolutional Network-Based Intelligent Evaluation Algorithm for the Quality of Spoken English Pronunciation [10]	A Convolutional Neural Network (CNN) is used to evaluate the quality of spoken English pronunciation. The research directly feeds the voice signal into the CNN model to extract each feature rather than using any speech recognition engine to transcribe the voice into text.
Improving English Pronunciation: An Automated Instructional Approach [11]	An experiment that checks which group of children is willing to improve their English pronunciation using several English films and English-language learning software such as Speech-To-Text software engine
Automatic Proficiency Assessment of Korean Speech Read Aloud by Non-Natives Using Bidirectional LSTM-Based Speech Recognition [12]	The foreigners are told to speak specific Korean sentences and tested in two scenarios: with and without text. The spoken sentences are then fed into a Bidirectional Long Short-Term Memory (BLSTM)-based model that measures five holistic proficiency scores: segmental accuracy, phono-logical accuracy, fluency, pitch, and accent.
Review of hardware implementation of Speech-To-Text Engine for Jawi Character [13]	A hardware implementation for helping students to read Jawi characters more effectively using Speech-To-Text tools
Automated Scoring of Nonnative Speech Using the SpeechRater^{SM} v. 5.0 Engine [14]	This research shows the overview of multiple features of the SpeechRater engine to evaluate the speech performance of foreigners speaking English.
Fairseq S2T: Fast Speech-to-Text Modeling with Fairseq [15]	This research developed a FAIRSEQ extension for Speech-To-Text modelling tasks such as STT translation and speech recognition called FAIRSEQ S2T.

Although the previous works in Table 1 have shown a significant contribution, there is still a research gap that this paper intends to fill. The majority of the mentioned related works focused on English as the target language, except for [12], which targeted Korean and [13], which targeted the Jawi character (Malay writing script based on the Arabic script). A Speech Recognition model developed for a specific language cannot be directly used for another language because it needs much tweaking or a complete model rebuild.

Otherwise, it will give a non-optimal result. The challenge from [9,10] is that the model was developed from scratch with NN or CNN. It was specifically designed for English, thus, cannot be used for Korean effectively and efficiently, unlike the open-source STT engines available for multiple languages. In [14], although the SpeechRater is used to conduct automatic scoring, it is not considered an STT engine and is mainly used to generate a TOEFL score. Thus, it is also not suitable for Korean. As for [12], although the BLSTM model is designed for Korean and can measure multiple holistic proficiency scores, the research only developed the model as is, without it being implemented to a particular platform or application to be used for everyday users. Even though the research results in [15] show that the FAIRSEQ S2T extension improves model performance, it is tested for English only. Thus it might show a different result if used in Korean. Apart from what has been mentioned in the table, there are other exciting research results, such as [16–19]. In [16], a method was developed to enrich speech recognition text results by processing the sentence boundaries and speech disfluencies that are proven suitable for English. In [17], a method was shown that measures the readability of STT transcribed text based on the correctness of responses to comprehension questions, speed of reading and responding to questions, and a subjective rating of passage difficulty. Although these methods can improve the STT result, it is not considered suitable for the case in this paper because the sentence is already provided, thus minimizing the user's possibility to use fillers such as 'uh,' 'um,' false starts, repeats, et cetera. In [18], a Part of Speech (POS) tagging was implemented in Odia, a low-resourced language from India with a unique morphology and script. The research resulted in an NLP system that can tag each word in Odia to its lexical classes using an HMM model to help users trying to learn Odia. A similar research result was shown in [19], where the authors developed a pre-trained model for the Khmer language, also a low-resourced language, and evaluated the model with POS tagging and news categorization. It also showed a significant result that can be improved regardless of its limited resources.

There has yet to be any research that combines STT engines and a semantic similarity check in the form of a mobile application to train users' Korean proficiency by generating a score automatically. This paper's differences and novelties compared to previous research results are that this paper uses an open-source STT engine. Instead of a DL, the score is generated using the semantic similarity check function provided by a lightweight framework for Python called Sentence Transformer, which will be implemented as a mobile application. The mobile application will show a Korean sentence, and the user can record themselves speaking it. Once done, the application will automatically show the score in percent value along with the transcribed text directly to the user. Furthermore, the research in this paper is an extension work of [20] that compared the performance of multiple open-source STT engines. The comparison results from the first phase of developing the ideal system mentioned in the previous section. Therefore, this paper focuses on developing the Rest API using Flask and mobile applications that take advantage of STT and Sentence Transformer to show the similarity score.

3. System Development and Design

This section describes the system architecture, the algorithm of both mobile applications and the Restful Application Programming Interface (Rest API), and the system requirements. Additionally, the initial User Interface (UI) design in the form of a storyboard is displayed to further visualize the application before its development.

3.1. System Architecture

The system architecture consists of the user, Rest API, and the mobile application. The Rest API will act as a communication bridge between the mobile application and the STT engine, along with a similarity check function. Both components are written in a different programming language. The system architecture is shown in Figure 2, along with a number to indicate the order.

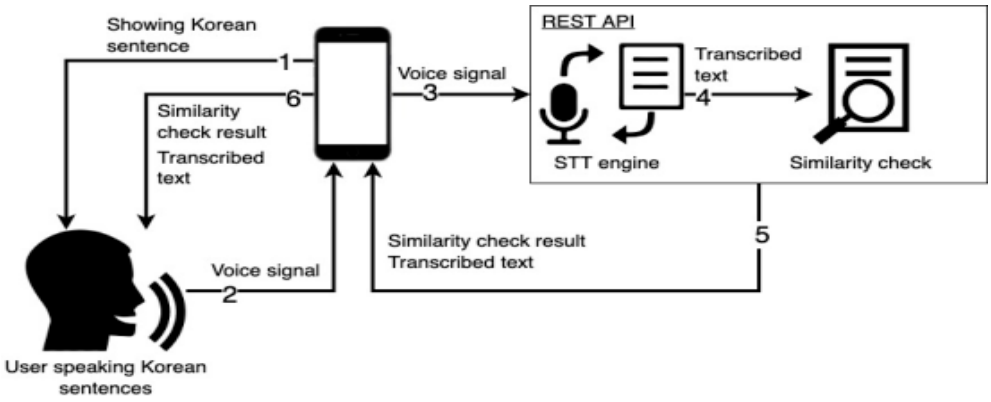

Figure 2. The Architecture of the Proposed System.

The system's flow starts with showing the pre-defined sentence in the application, and then the user can choose one of four STT engines based on [20] before beginning recording. The recorded voice is then sent to the server using Rest API to be transcribed into text using the selected STT engine. The transcribed result is then fed into a similarity check to produce the per cent value of how similar the transcribed text is compared to the pre-defined sentence. Finally, the per cent value and the transcribed text are returned to the mobile application via Rest API to be shown on the screen. Users can repeat the process with a different STT engine.

3.2. Flowchart Algorithm

This subsection will discuss the flowchart containing the algorithm to determine the application's input, process, and output before executing the coding stage. Additionally, this flowchart act as a problem-solving aid if errors appear.

3.2.1. Mobile Application

As mentioned previously, the service developed in this paper to train foreigners' Korean speaking proficiency is a mobile application. The user can record their voice reading one Korean sentence that will be shown randomly from a collection of sentences that is hard coded in the application. Once done, the user can upload the voice to the Rest API to generate the score. After the score has been generated, the result will be returned to the mobile application. It is worth noting that the user can take however many tries they want. The flowchart of the mobile application is shown in Figure 3.

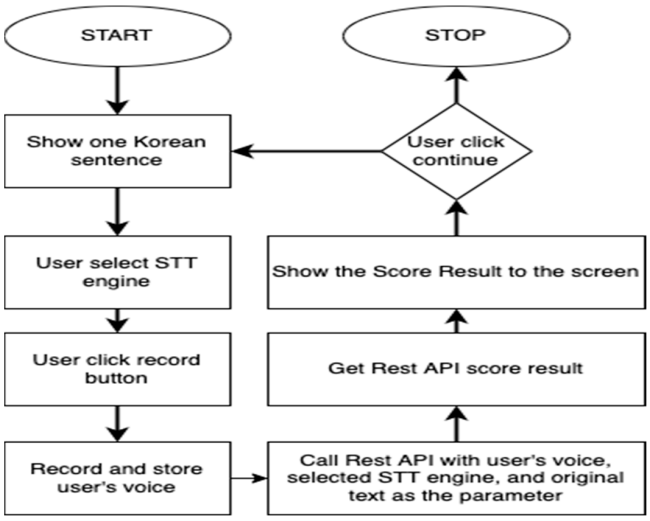

Figure 3. The Flowchart of Mobile Application to Illustrate the Algorithmic Flow.

3.2.2. Rest API

The Rest API will be slightly more complex than the flow of the mobile application because the Rest API is the backbone of the system and handles the communication between the mobile application and server along with its data. The mobile application will call the PHP Rest API and collect all the parameter information, which is raw voice data, selected STT, and original text. Then, the raw voice data is validated to ensure they are in the correct format and size. If the validation is failed, then it will return an error message. If the validation is successful, the raw voice data, selected STT, and the original text are converted into JSON to prepare to call the Python Rest API. When the Python Rest API is called, the raw audio file has to be converted into the correct format and specification for the STT engines. Otherwise, it would not transcribe the text. Voice data is given to the specific STT engines according to the user's selection (i.e., if the user selects Google, then the voice will only be given to the Google STT). After that, the transcribed text and the original text are fed into the Sentence Transformer to obtain the similarity score. Once done, the transcribed text, original text, and similarity score results are converted into JSON format and returned to the PHP Rest API. The PHP Rest API then parses the JSON data into a readable string to be shown on the mobile application.

Each Rest API present in this system has two different purposes. The PHP Rest API is responsible for getting the raw data, validating the data, formatting the data, and executing the Python Rest API. On the other hand, the Python Rest API is responsible for converting the raw data into the required specification, executing the STT engine along with the similarity checker, converting the final result into JSON, and returning it to the PHP Rest API. Based on the description, the PHP Rest API directly connects to the mobile application and the Python Rest API. In contrast, the Python Rest API directly connects to the PHP Rest API and the STT engine, along with the similarity checker. The flow of the Rest API will be shown in Figure 4.

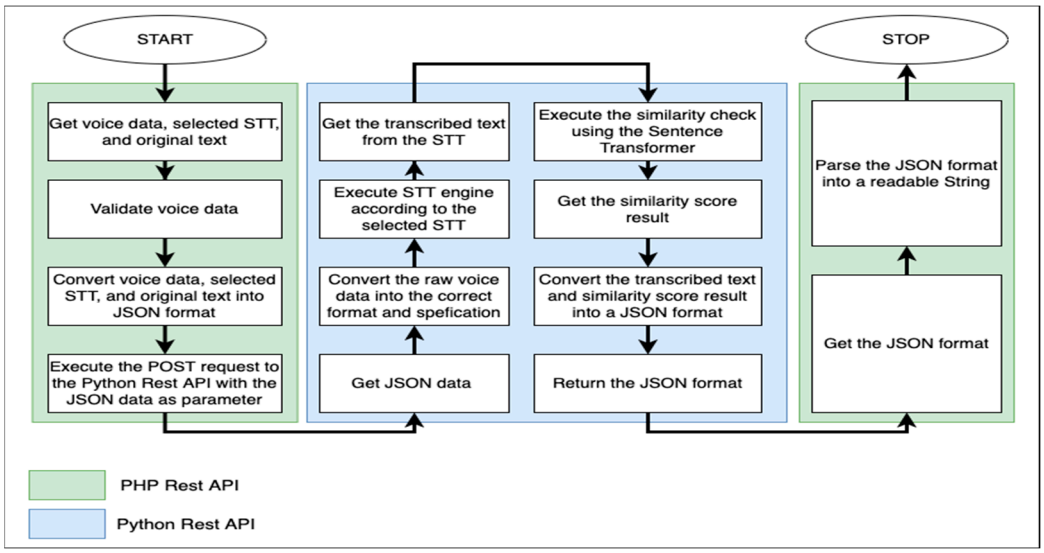

Figure 4. The Flowchart of Rest API to Illustrate the Algorithmic Flow.

The developed system requires two Rest APIs because the Sentence Transformer, which conducts the similarity checker, can only be run in a Python environment. In contrast, the PHP Rest API is required because the PHP environment offers a more manageable, lighter, and more optimized way to get the raw voice data from the mobile application and do the validation. The downside of this scheme is that the device requires an active internet connection for the mobile application to connect with the Rest API.

3.3. System Requirements

The requirements for mobile applications and Rest API can be derived from the flowchart described in the previous section. The main requirements for the mobile application are to show the Korean sentence, choose the STT engine, record the voice, store the voice, call the Rest API URL, and retrieve the similarity value along with the transcribed text from the Rest API to be shown on the screen. For the Rest API, the main requirements are retrieving and validating the original recorded raw voice data, converting them to the appropriate format and specification as determined in [20], executing the STT engine along with the similarity check, and returning the similarity value along with the transcribed text. The requirements of Rest API are depicted in Table 2, whereas the requirements for the mobile application are depicted in Table 3.

Table 2. Mobile Application.

Requirements	Description
Show pre-defined Korean sentence	The application can show the pre-defined sentence by the user
Users can choose which STT to use	The application can give the user the options of the available STT
Record users' voices and save them into a file	The application can record the user's voice and save it into a file in the storage
Play the recorded voice	The application can play the last recorded voice by the user
Call the Rest API with the required parameters	The application can call the Rest API and pass the required parameters: a raw voice file, selected STT, and original text.
Retrieve the result from the server	The application can retrieve and show the similarity score and transcribed text via Rest API

Table 3. Rest API.

Requirements	Description
Retrieve and validate the raw voice data	Rest API can retrieve and validate the raw voice data to ensure it is in the acceptable format.
Convert the original recorded file	Rest API can convert the raw voice data into the correct specification for the STT engine to work.
Execute the STT engine and similarity check	Rest API can execute the STT engine based on what the user had selected, then pass the transcribed text to the similarity check to get the score.
Return the final results	Rest API can return the similarity score and the transcribed text in the form of a readable JSON to the mobile application

3.4. Initial User Interface Design

Before beginning the construction process, a UI design also needed to be done to visualize the application. There are two core UI designs of the application: the home page and the result page. The UI design is depicted in Figure 5.

In Figure 5, the image on the left is the home page, whereas the image on the right is the result page. The home page has text for the user to read and a radio button to choose the STT engine. Underneath the text is three buttons to record the voice, upload the file to be checked, and play the latest recorded voice. However, the upload and play button will only be active if the user has recorded their voice. A similar UI is shown on the result page, with the addition of the transcribed text and the score as a result from the STT engine and a similarity check for the user to see. The user is free to choose another STT engine and repeat the process.

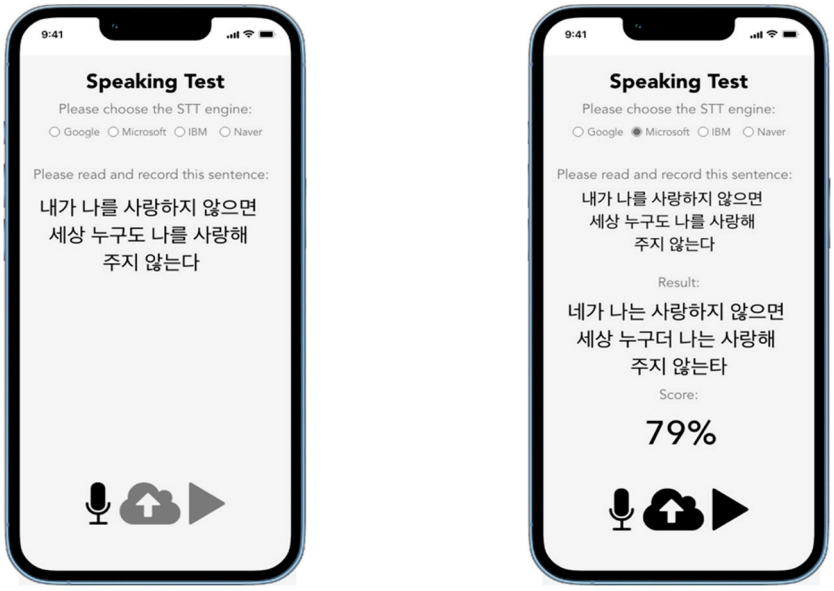

Figure 5. Initial UI Design of the Application. (Note: The Korean in the **left** figure is the chosen sentence users require to speak. The Korean, located at the top in the **right** figure, is the chosen sentence that users require to speak, while the Korean, located at the bottom in the right figure, is the transcribed text from the STT engine).

4. System Implementation

This section will discuss the environment used to develop the system and the implementation results of the Rest API and mobile application.

4.1. Implementation Environment

The implementation environment, which in this case is the Personal Computer (PC) used to code the mobile application, and the Rest API is equipped with Windows 11 version 21H2 to build 22000.588, Intel i7-10700, and 32 GB of RAM. In terms of software, the tools to build the application is Android Studio Bumblebee, version 2021.1.1 Patch 2, build number #AI-211.7628.21.2111.8193401 and to build the Rest API is Visual Studio Code version 1.68.1 (for PHP Rest API), and with Jupyter Notebook version 6.4.5 for the Python Rest API.

The PHP Rest API is written in PHP programming language version 8.1.7. In contrast, the Python Rest API and the similarity check and STT are written in Python version 3.8.9 from Anaconda version 2.1.4 with a Flask framework version 2.1. Xampp version 8.1.6 is used to create a local server to test the functionality, and Postman version 9.22.2 is used to check the Rest API result.

4.2. Rest API Implementation

As mentioned previously, the PHP Rest API is responsible for retrieving and validating raw voice data to ensure the file is in the correct format and size. Once everything is in order, the PHP Rest API uses PHP built-in functions, which are curl_init() and curl_setopt(), to call the Python Rest API. After calling the Python REST API, the result is retrieved and converted into a JSON format that includes the message, status, transcribed text, and the similarity score for showing it to the mobile application. The code snippet for the PHP Rest API is depicted in Figure 6. Line 40 is where the Python Rest API URL is defined. Line 48 is where the POST method is defined, along with passing the parameters. Line 55 is where the result from Python Rest API is returned.

```php
if(!isset($errorMSG)) {
    $url = 'http://127.0.0.1:5000/resultcheck';
    $ch = curl_init($url);
    $payload = json_encode(array(
        'stt' => $_POST['sttOption'],
        'fileName' => $fileName,
        'originalText' => $_POST['originalText']
        )
    );
    curl_setopt($ch, CURLOPT_POSTFIELDS, $payload);
    curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type: application/json'));
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    $result = curl_exec($ch);
    $status = curl_getinfo($ch, CURLINFO_HTTP_CODE);
    curl_close($ch);
    $jsonResult = json_decode($result, true);
    echo json_encode(array(
        "message" => "Success",
        "convertedText" => $jsonResult['convertedText'],
        "score" => $jsonResult['score'],
        "status" => true));
}
```

Figure 6. Code Snippet of the PHP Rest API.

The STT engine and similarity check procedure are done using Python in Jupyter Notebook because this paper continues [20]. Although each of the STT engines used in this system offers an API to directly use the service in the various platforms, including mobile applications, it would be deemed inconvenient because it means that the application has to hit four different APIs for different STT engines. Although, in the end, the Rest API will execute the STT procedure by calling the corresponding API designed by each STT engine developer, the similarity check function is only available for Python since it was developed as a framework for Python. Thus, the Python Rest API will transcribe the voice into text and directly execute the similarity check function to return the similarity score and the transcribed text. A recommended framework to build the Rest API in Python is using Flask, specifically Flask-RESTful. According to its official website, Flask is a lightweight Web Server Gateway Interface (WSGI) framework optimized for Python. Flask-Restful is an extension for Flask that offers an extra function to quickly build Rest API and a lightweight abstraction that is friendly with popular libraries.

To install the Flask, the user simply creates and activates a virtual environment that runs Python 3 (as recommended) in the terminal and installs the framework using the PIP command, 'pip install flask'. After the installation is completed, it is just a matter of importing the library into the Python file. The code inside the Python file is mainly combined with the pieces of code done in [13] by adding the audio file converting functionality and Flask configuration. The libraries used for converting the audio file are AudioSegment by PyDub and SoundFile. To run the Python code, execute the 'python3 [insert_python_filename].py' command in the terminal after activating the virtual environment. If successful, a terminal will return a URL indicating that the local server is alive.

The Python Rest API will convert the raw voice data into the correct specification, such as the extension, frame rate, channels, and sample width. The converted file will be stored on the server temporarily, and the converted file name will be stored inside a variable for the STT engine. As mentioned previously, the users must choose from four different STT engines, which is also a required parameter to call the Python Rest API. Each STT engine holds a number, either 1, 2, 3, or 4, with 1 being for Google STT; 2 being for Microsoft Azure; 3 being for IBM Watson; and 4 being for Naver Clova. Using a simple If-Then-Else algorithm, the Python Rest API will only execute the chosen STT engines. After the STT engine transcribed text, it directly fed into the similarity checker and the original text (which is also the required parameter for the Python Rest API). Once done, the transcribed text and similarity score will be returned in JSON format. The code snippet for Python Rest API is shown in Figure 7 below.

```python
@app.route('/resultcheck', methods=['POST'])
def post():
    data = json.loads(request.data)
    finalScore = 0.0
    STTOption = data['stt']
    fileName = data['fileName']
    originalText = data['originalText']
    textResult = ""
    newAudio = AudioSegment.from_file('upload/' + fileName, "3gp", frame_rate=44100,
        channels=1, sample_width=2)
    fileName = fileName.split(".")[0]
    newAudio.export('upload/convert_' + fileName + '.wav', format="wav")
    data, samplerate = soundfile.read('upload/convert_' + fileName + '.wav')
    soundfile.write('upload/convert_' + fileName + '.wav', data, samplerate, subtype='PCM_16
        ')
    fileName = 'upload/convert_' + fileName + '.wav'

    if STTOption == "1":
        client = speech.SpeechClient()
        with io.open(fileName, 'rb') as audio_file:
            content = audio_file.read()
            audio = speech.RecognitionAudio(content=content)

            config = speech.RecognitionConfig(
            encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,
            language_code='ko-KR',
            audio_channel_count = 1,
            enable_word_time_offsets=True)

            response = client.recognize(config=config, audio=audio)
            for result in response.results:
                textResult = str(result.alternatives[0].transcript)
                client = speech.SpeechClient()
                print(str(result.alternatives[0].transcript))

    queries  = [originalText]
    corpus  = [textResult]
    corpus_embeddings = embedder.encode(corpus, convert_to_tensor=True)
    top_k = len(corpus)
    for query in queries:
        query_embedding = embedder.encode(query, convert_to_tensor=True)
        cos_scores = util.pytorch_cos_sim(query_embedding, corpus_embeddings)[0]
        cos_scores = cos_scores.cpu()

        for idx in range(top_k):
            score = "%.3f" % (cos_scores[idx])
            finalScore = float(score)*100

    # return (str(finalScore) + "%")
    data = {
        "convertedText": textResult,
        "score": str(finalScore)
        }
    response = app.response_class(response=json.dumps(data), status=200, mimetype='
        application/json')
    return response
```

Figure 7. Code Snippet of the Python Rest API.

Postman is used to requesting dummy data to the Rest API and check the result. In this case, a dummy voice file, a random original text, and a random STT engine option are sent, and the result is depicted in Figure 8.


```
{
    "message": "Success",
    "convertedText": "랄랄랄랄라",
    "score": "0.0",
    "status": true
}
```


Figure 8. PHP and Python Rest API Result. (Note: The Korean in the figure is the transcribed text from the STT engine based on users' voice recording).

It is shown in Figure 6 that the Rest API returns a result in JSON format containing the message, the converted text, the score, and the status. It is worth mentioning that message and status are only complementary results to give additional information if a particular error has occurred. At this point, the Rest API can be considered ready for the mobile application.

4.3. Mobile Application Implementation

The mobile application is developed in Android Studio using the Java programming language. The main reason Android was chosen is that according to [21], the market share for Android was 68.96% compared to iOS, which only reached 30.66%, meaning that in South Korea, there are more Android users than iOS users. Furthermore, Android applications can be developed on a Windows-based Personal Computer (PC), whereas iOS application development can only be done on a macOS-based PC. The mobile application must be run on a device using at least Android version 5.0 (Lollipop).

The first thing to do is to import the dependency to interact with Rest API, which is called Retrofit. In this case, Retrofit version 2.6.1 is used along with its complementary dependencies, such as JSON converter (converter-gson) version 2.3.0 and log interceptor (logging-interceptor) version 4.10.0. The required dependencies can be imported into the application-level Gradle configuration file in the Android Studio. After importing the dependencies, another configuration required is to add permission to write a file into external storage, record audio using a microphone, access local storage, and access the internet. These permissions can be done by modifying the AndroidManifest.xml in the Android Studio.

For developing an Android application inside Android Studio, the interface layout can be arranged by coding the XML file inside the folder called 'layout', whereas the function code of the application is written in the Java file located in a folder called 'java'. The most important part of the code in this application is calling the Rest API using Retrofit, which consists of three separate files to handle the different tasks. The first is a file responsible for configuring the base URL of the Rest API and defining the Rest API method, form, and parameters. The second is a Java file responsible for retrieving the server response, which mainly contains a variable and its constructor to hold the value from the server. The third is the main file that calls the function defined in the interface file and its required parameter. As mentioned previously, this function will return the two results from the Rest API. The operation procedure of the application is shown in Figure 9, and the screenshot of the final result is shown in Figure 10.

Figure 9. The Operation Procedure of the Application. (Note: The Korean in this figure is the selected sentence users require to speak).

It can be seen from Figure 9 that the operation starts with selecting the desired STT Engine because the application will not record if the user has not chosen the STT engine. A pre-defined Korean sentence is shown at the top of the screen, along with three buttons

underneath it: record, play/stop, and upload. However, only the record button is available to press when first launched, while the other remains unavailable. After selecting the STT engine, the user can tap the record button and begin to record while reading the sentence. If a user taps the button again to stop recording, the audio file will be stored inside the device's storage. If the play/stop button is pressed, it will play or stop the last recorded voice, whereas the upload button is to upload the voice to the Rest API and initiate the automatic scoring procedure. Once generated, the transcribed text and the similarity score will be shown below the three buttons. As mentioned previously, the audio file is uploaded to the server to gather the dataset for the next phase ideal system's development. Therefore, the audio filename, transcribed text from the STT engine, and the similarity score are stored in a database.

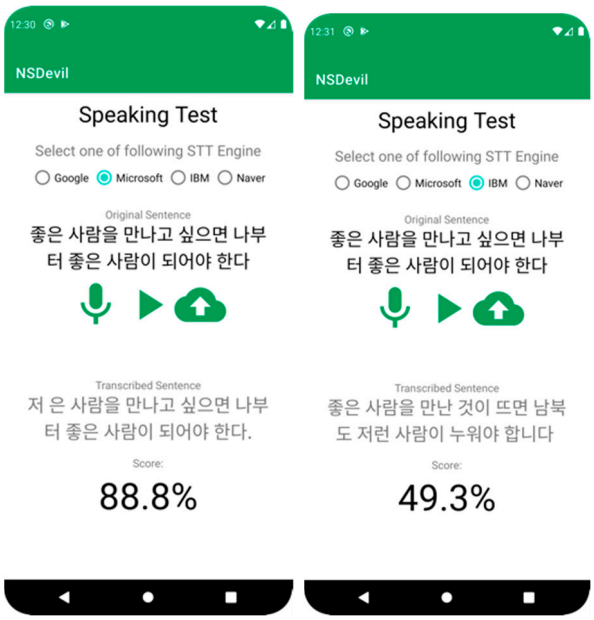

Figure 10. The Screenshot of the Application. (Note: The Korean at the top of these figures is the selected sentence that users require to speak, while the Korean at the bottom of these figures is the transcribed text from the STT engine).

5. Performance Evaluation

For evaluating the performance of the proposed system, two types of evaluation are conducted: measuring the response time for generating the score and measuring its consistency in delivering both the transcribed text and the score. The performance evaluation method sends 100 requests to generate the transcribed text along with the score for each STT engine. Then, the time it takes from tapping the "Upload" button until the results appear on the screen is recorded in milliseconds as the response time. The shown transcribed text and score are also recorded to measure its consistency. Specifically for measuring consistency, the same file will be used in all 100 requests.

For the response time, the average time from 100 requests for each STT is calculated along with the overall response time average from all STT engines. For consistency, each different transcribed text and score result will be counted as 1. Thus a bigger number means less consistency.

The evaluation environment is carried out in an Android Emulator provided by Android Studio with a 5.7" screen carrying 1080 × 220 pixels, four processor cores, 1536 MB of RAM, 512 MB of internal storage, and Android 10. As for the internet connection, a ground internet via LAN cable provided by KT with download and upload speeds clocked around 824 Mbps and 540 Mbps, respectively, is used at the time of the test.

The first test is conducted using the Google STT, and out of 100 requests, the average score is measured at 2232 ms or 2.23 s. The second test is carried out using Microsoft

Azure STT, and out of 100 requests, the average score is calculated as a faster response time compared to Google STT, which is 1137 ms or 1.13 s. The subsequent measurement is done using IBM Watson. Out of 100 requests, it resulted in the slowest average time, which is 6088 ms or 6.08 s, resulting almost three times slower than Google STT and nearly six times slower compared to Microsoft Azure. The last response time test is done using Naver Clova, and out of 100 requests, the average response time is 5488 ms or 5.48 s. Although not as slow as IBM Watson, it is still considerably slower than Google STT and IBM. Therefore, the overall average response time of every STT combined is 3736 ms or 3.7 s. Because the transcribing process for each STT is done in the cloud using their respective server (every STT engine is a cloud-based service), the speed of their server plays a significant role in evaluating the performance of the mobile applications. Therefore, the response time could fluctuate from time to time, depending on the server's traffic. The response time graph is depicted in Figure 11.

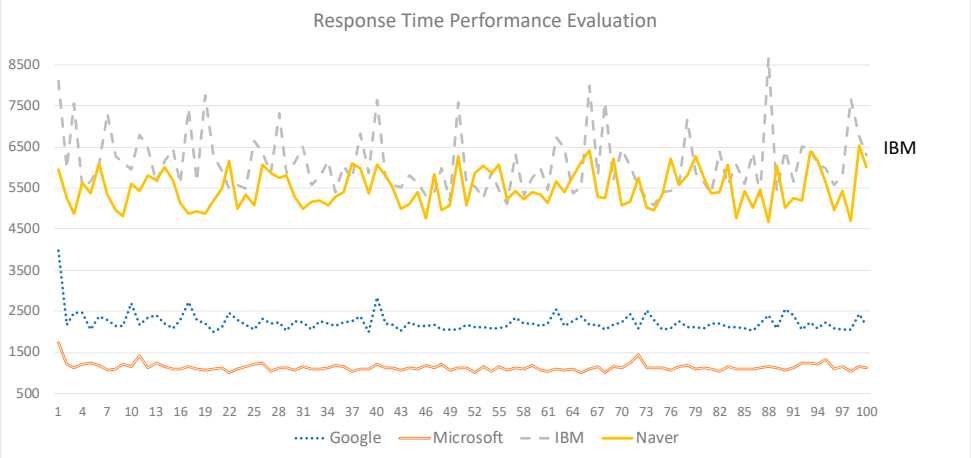

Figure 11. The Response Time Evaluation Graph of Every STT Engine.

As for the consistency evaluation, Google STT, Microsoft Azure, and IBM Watson are consistent across 100 iterations scoring only 1, meaning that every request resulted in the exact transcribed text and score. However, for Naver Clova, out of 100 requests, it resulted in eight different results, indicating that Naver Clova is the least consistent among other STT engines tested in this paper. The consistency score is depicted in Figure 12.

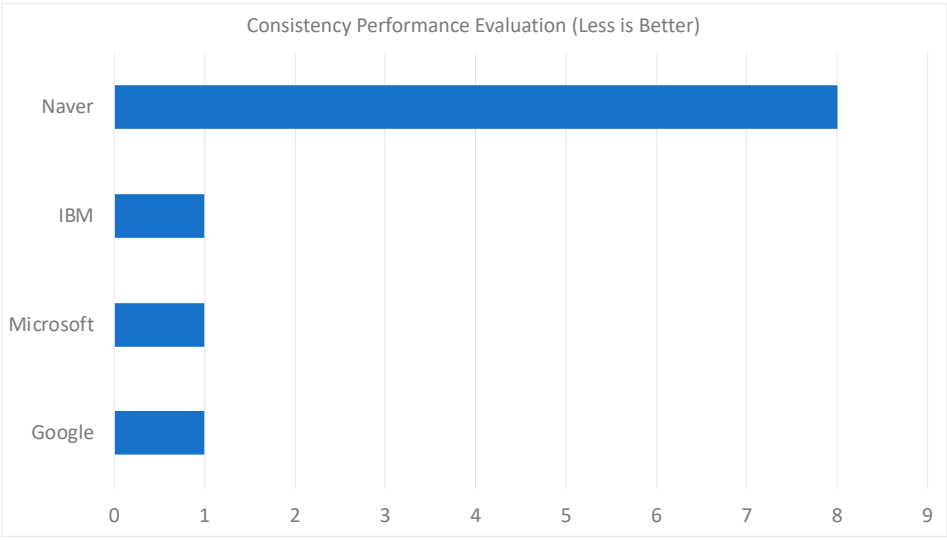

Figure 12. The Consistency Evaluation Graph.

6. Conclusions and Future Studies

This paper developed a mobile application that automatically generates a score of how well foreigners speak Korean sentences using STT technology and the Sentence Transformer framework for Python to check the similarity to help them increase their Korean pronunciation. The application shows a pre-defined Korean sentence on the screen for the user to read and record themselves using the voice recording function inside the application. Once the user records their voice, they can upload the audio file to the server that will transcribe it into text using the STT engine and then check the similarity between the original pre-defined Korean sentence and the transcribed text. The similarity score result is in percent value. Thus the generated score is based on a combination of the user's ability to pronounce the sentence correctly and the good performance of the STT engine by being able to transcribe the text regardless of the user's accent. The STT engine's performance is essential because this paper not only implements the auto-scoring feature but also gathers the user's voice, score, and transcribed text from various open-source STT engines as the initial dataset for the ideal system.

The application can communicate with the server using a Rest API developed in this paper. The Rest API has two components to handle the system's different functions. The PHP Rest API handles the audio file, i.e., uploading and managing the audio file. In contrast, the Python Rest API is responsible for converting the raw audio file into the correct extension and specification, performing the STT procedure, and executing the similarity check function. The results from the Python Rest API are returned to the PHP Rest API to be formatted in JSON before sending the result to the mobile application. The Rest API receives the audio file, pre-defined Korea sentence, and selected STT option as the parameters. After processing, the Rest API will return the transcribed text from the selected STT engine along with the similarity score, which is then displayed in the mobile application.

The performance evaluation is conducted to measure the response time and consistency. The response time measures how long the user taps the 'upload' button until the result is displayed on the screen, measured in 3.7 s as the overall average. The consistency measures how consistent the transcribed text and similarity score results are. After testing, it is shown that Google STT, Microsoft, and IBM show a consistent figure showing the exact transcribed text and score. In contrast, Naver Clova is the least consistent, showing eight different transcribed texts and scores.

For future studies, this application will be distributed to a group of foreigners who can speak Korean adequately as a volunteer to collect all information (audio file, transcribed text, and similarity score). Then, the information will be compiled to create a dataset for developing an ML or DL-based model that can automatically generate the score through a more profound analysis, semantically and holistically, instead of being based only on similarity. Furthermore, a digital game-based learning approach that combines learning objectives, methods, and outcomes into an interactive game that offers multiple advantages compared to traditional teaching methods and an automatic lexicon analyzer can be added to the mobile application. With this approach, the mobile application will provide a game with multiple levels for foreigners to train their proficiency in speaking Korean. The game's level will start by ordering foreigners to speak simple Korean daily words and finish with complex sentences along with specific tests such as filling in the blanks or speaking the correct answer based on the question. In addition, to increase the depth of the application, a Content Management System will also be added to provide an administrator to efficiently manage the content or asset of the game, including words/sentences used in every level, the scoring system, the penalty system, etc. The lexicon analyzer can help the user understand the lexicon category of each word to be used properly [18,19,22,23]. Therefore, the mobile application can further gather users' voices as the dataset for the ML or DL-based model. At the same time, it also provides users with a training service to increase their proficiency in Korean.

Author Contributions: Conceptualization, A.B.W. and M.H.; methodology, A.B.W. and M.H.; software, A.B.W.; validation, M.H.; formal analysis, A.B.W. and M.H.; investigation, A.B.W.; resources, A.B.W.; writing—original draft preparation, A.B.W.; writing—review and editing, A.B.W. and M.H.; visualization, A.B.W.; supervision, M.H.; funding acquisition, M.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by NSDevil Company and Gyeongnam SW Convergence Cluster 2.0, under contract.

Institutional Review Board Statement: Ethical review and approval were waived for this study, due to not involving identifiable personal nor sensitive data.

Informed Consent Statement: Students' consent was waived due to not involving identifiable personal nor sensitive data.

Data Availability Statement: The data presented in this study are available upon request from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. The Korean Culture and Information Service (KOCIS). Summary: Korea.Net: The Official Website of the Republic of Korea. Available online: https://www.korea.net/AboutKorea/Society/South-Korea-Summary (accessed on 12 October 2022).
2. Lee, K.M.; Ramsey, S.R. *A History of the Korean Language*; Cambridge University Press: New York, NY, USA, 2011.
3. Yonhap. Number of Foreigners Staying in S. Korea Decreased 3.9% in 2021 Amid Pandemic. Available online: https://www.koreaherald.com/view.php?ud=20220126000736 (accessed on 12 October 2022).
4. TOPIK Information. Test Outline: TOPIK Korean Proficiency Test. Available online: https://www.topik.go.kr/TWGUID/TWGUID0010.do (accessed on 12 October 2022).
5. A, V.; Jose, D.V. Speech to Text Conversion and Summarization for Effective Understanding and Documentation. *Int. J. Electr. Comput. Eng. (IJECE)* **2019**, *9*, 3642–3648. [CrossRef]
6. Karpagavalli, S.; Chandra, E. A Review on Automatic Speech Recognition Architecture and Approaches. *IJSIP* **2016**, *9*, 393–404. [CrossRef]
7. Ziman, K.; Heusser, A.C.; Fitzpatrick, P.C.; Field, C.E.; Manning, J.R. Is Automatic Speech-to-Text Transcription Ready for Use in Psychological Experiments? *Behav. Res. Methods* **2018**, *50*, 2597–2605. [CrossRef] [PubMed]
8. Iancu, B. Evaluating Google Speech-to-Text API's Performance for Romanian e-Learning Resources. *Inform. Econ.* **2019**, *23*, 17–25. [CrossRef]
9. Wang, X. Research on Open Oral English Scoring System Based on Neural Network. *Comput. Intell. Neurosci.* **2022**, *2022*, e1346543. [CrossRef] [PubMed]
10. Zhan, X. A Convolutional Network-Based Intelligent Evaluation Algorithm for the Quality of Spoken English Pronunciation. *J. Math.* **2022**, *2022*, 7560033. [CrossRef]
11. Mitra, S.; Tooley, J.; Inamdar, P.; Dixon, P. Improving English Pronunciation: An Automated Instructional Approach. *Inf. Technol. Int. Dev.* **2003**, *1*, 75–84. [CrossRef]
12. Oh, Y.R.; Park, K.; Jeon, H.-B.; Park, J.G. Automatic Proficiency Assessment of Korean Speech Read Aloud by Non-Natives Using Bidirectional LSTM-Based Speech Recognition. *ETRI J.* **2020**, *42*, 761–772. [CrossRef]
13. Razak, Z.; Sumali, S.R.; Idris, M.Y.I.; Ahmedy, I.; Yusoff, M.Y.Z.B.M. Review of hardware implementation of Speech-To-Text Engine for Jawi Character. In Proceedings of the 2010 International Conference on Science and Social Research, Kuala Lumpur, Malaysia, 5–7 December 2010.
14. Chen, L.; Zechner, K.; Yoon, S.-Y.; Evanini, K.; Wang, X.; Loukina, A.; Tao, J.; Davis, L.; Lee, C.M.; Ma, M.; et al. Automated Scoring of Nonnative Speech Using the SpeechRaterSM v. 5.0 Engine. *ETS Res. Rep. Ser.* **2018**, *2018*, 1–31. [CrossRef]
15. Wang, C.; Tang, Y.; Ma, X.; Wu, A.; Okhonko, D.; Pino, J. Fairseq S2T: Fast Speech-to-Text Modeling with Fairseq. In Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: System Demonstrations, Suzhou, China, 4–7 December 2020.
16. Liu, Y.; Shriberg, E.; Stolcke, A.; Hillard, D.; Ostendorf, M.; Harper, M. Enriching Speech Recognition with Automatic Detection of Sentence Boundaries and Disfluencies. *IEEE Trans. Audio Speech Lang. Process.* **2006**, *14*, 1526–1540. [CrossRef]
17. Jones, D.; Jones, D.A.; Wolf, F.; Gibson, E.; Williams, E.; Fedorenko, E.; Reynolds, D.A.; Zissman, M.A. Measuring the Readability of Automatic Speech-to-Text Transcripts. In Proceedings of the 8th European Conference on Speech Communication and Technology, EUROSPEECH, Geneva, Switzerland, 1–4 September 2003.
18. Pattnaik, S.; Nayak, A.K.; Patnaik, S. A semi-supervised learning of HMM to build a POS tagger for a low resourced language. *J. Inf. Commun. Converg. Eng.* **2020**, *18*, 207–215. [CrossRef]
19. Jiang, S.; Fu, S.; Lin, N.; Fu, Y. Pretrained models and evaluation data for the Khmer language. *Tsinghua Sci. Technol.* **2022**, *27*, 709–718. [CrossRef]

20. Wahyutama, A.B.; Hwang, M. Performance Comparison of Open Speech-to-Text Engines using Sentence Transformer Similarity Check with the Korean Language by Foreigners. In Proceedings of the IEEE International Conference on Industry 4.0, Artificial Intelligence and Communications Technology, Kuta, Bali, 28–30 July 2022.
21. Mobile Operating System Market Share Republic of Korea. Available online: https://gs.statcounter.com/os-market-share/mobile/south-korea/#monthly-202112-202212 (accessed on 13 December 2022).
22. Wahyutama, A.B.; Hwang, M. Design and Implementation of Digital Game-based Contents Management System for Package Tour Application. *J. Korea Inst. Inf. Commun. Eng.* **2022**, *26*, 872–880. [CrossRef]
23. Wahyutama, A.B.; Hwang, M. Implementation of Digital Game-based Learning Feature for Package Tour Management Application. *J. Korea Inst. Inf. Commun. Eng.* **2022**, *26*, 1004–1012. [CrossRef]