Report: Mini Translator

Rujun Yao
Pandalv9999@gmail.com

Abstract

Language diversity, although contributes to prosperity of different cultures, is a barrier for people of different cultural backgrounds to communicate. Mini Translator is one of the solutions to make inter-human communications convenient. With four different Python scripts, Mini Translator implements three different functionalities—Translate Text, Translate Speech, and Record Audio. Python's scripting nature will make Mini Translator a portable application, and hence can be further applied to different platform. Although Mini Translator is still in development, it will find its role under current multi-cultural environment.

Introduction

Cultures are different among people in the world. One of the most representative things that can differentiate different cultures is language. In modern society, there are about five thousands to seven thousands different languages are spoken by people. More languages disappeared in the long human history. Although the diversity of languages reflects the prosperity of human culture, it increases the difficulties of communication between people spoken different languages. As told in Genesis 11:1-9, a united humanity in the generations following the Great Flood agree to build a tower tall enough to reach heaven. God was astounded by the power of unity and decided to confound their speech so that they can no longer understand each other and scattered them around the world. Hereby, the importance of understanding different languages is obvious. Fortunately, in modern society, people fluent in multiple languages are essential in inter-cultural communication. However, for ordinary people, it is still hard for them to understand people speaking different languages.

Figure 1: People spoken different languages are communication with a translator

The different between human beings and other animals is that human beings know to utilize existing tools and to create new tools. There are many tools that help people understand other languages, for example, paper-made dictionary, electronic dictionary, smartphone application, electronic translator… Those tools make inter-human communication more convenient. As shown in *Figure 1*, the man speaking English, with the assistant of instant translator, could understand the words of the women speaking Japanese. With those translation tools, people traveling around will meet fewer difficulties when they go to other countries whose people speaking different languages.

Mini Translator is such a tool. Utilizing cognitive service of Microsoft, Mini Translator implemented a more efficient and more accurate translation algorithm, Neural Network (NN) Translations. The translation, instead of consider three to five word phases as a entity to translate, as the traditional algorithm Statistical Machine Translation (SMT) does, takes the whole sentences as a entity, with the assistance of Machine Learning techniques, to generate a more human-comprehensive translated sentence. For more detailed information, please go to the

"Principle" part for references. Mini Translator has two functionalities of translation—Translate Speech and Translate Text. In Translate Speech, user can select a prepared wave file of specific format, and Mini Translator will generate a new wave file of languages being translated to. User can also record down the wave file at place and Mini Translator will also generate the translated file. In Translate Text, User can use Mini Translator to translate any prepared text file and the Mini Translator will generated another text file of translated words. The functionality of Mini Translator is sufficient, considering the multi-cultural environment of society.

Noteworthy, Mini Translator is written in Python, one of the most popular programming languages in the world. The scripting language nature of python makes computer only needs an interpreter to run the code. Unlike complier, which considered more elements of computer base layer, interpreter is more portable. Therefore, Mini-Translator can be run in various environments. Such a characteristic is important for translation application. With Mini Translator, the user can quickly take audio translations among ten different languages and acquire text translation among sixty-three languages supported by Microsoft.

Implementation

As mentioned before, Mini Translator use python to take advantages of its scripting language nature. More specifically, Mini Translator is developed under IDE PyCharm 2018. 2 (Professional Edition), and the environment inside the IDE is Python 3.7. This version of Python is the newest version of Python interpreter. However, this version also increases the difficulties of the development process. The most noticeable difficulty is that some python libraries, which were well fit in the previous version (Python 2.7 or other), are incompatible with this version, and the developers of those libraries had not update the libraries. Therefore some alternative solutions was purposed to get around of the incompatibility.

Mini Translator consisting of four different Python scripts. They are classTextTranslate.py, classAudioTranslate.py, classAudioRecord.py and GUI.py. Each script implements one of these important aspects of Mini Translator—Translate Text, Translate speech, record and the front-end interface.

The classTextTranslate.py is the script implementing Translate Text functionality. The script uses the given example code of Microsoft Text Translate API. The example code only supports text translation of a string. The sample code only establishes http connection to the server once to acquire translated text, and the translation is returned in JSON form. After modification, the file input and output API is added to the code. After the file is opened, the script will read one line of the file each time as input, establish http connection to the server with that line of input, and receive the translated text back from the server, until the file input reaches the end. The translated text is encoded in JSON form, and other information in returned message from the server is irrelevant. Therefore, an extra step is required to extract the translated text out from a bunch of irrelevant information. Note that when a new line character ('\n') is sent to the server, the server will return messages as shown in *Figure 2*. Such error messages add complexities to our message extraction process, so an if-statement is added to decide whether the line input is a new line character.

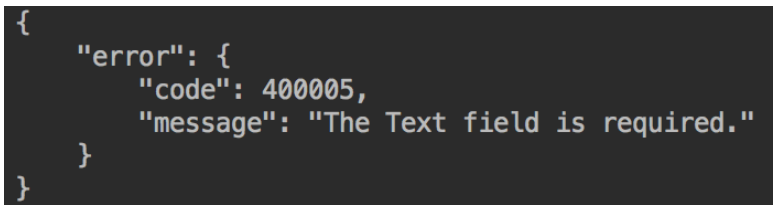

Figure 2. Messages returned from the server after sending new line character '\n'

The whole translation process can be described by the following pseudo-code:

Open input and output file
In each loop (each line of the file input):
Transform the text into dictionary form;
Transform the dictionary form into JSON format;
Add header to the format;
Establish an http connection to the Microsoft host;
Request a translation service and get the response;
Extract the translated text from the returned JSON formatted message;
Write the output in the target text file.
Close input and output file.

The classAudioTranslate.py script takes charge of the Translate Audio Functionality. The script uses the given example code of Microsoft Translate Speech API. The example code takes the input of a standard PCM, 16bit, 16kHz, and mono format wave file and establish Web-socket connection to the server. The server receives the request and analyzes the audio file, and returns the translated audio file and text of the content spoken. Since the example code is powerful enough to be part of Mini Translator, only a little modification is required. First of all, the input file must be specified by the user, and the output file is simply generated with the name of input file adding postfix '_translated.wav'. Second, the user must be able to select languages. The selection is among ten languages that support audio translation.

The classAudioRecord.py script records down user's speech and save it as a wave file locally. The wave file is then treated as input of Translate Speech to take further operations. The code is transplanted from an open-source Github site. The code import a library called Pyaudio and use it to create a recorder. The recorder has two different record modes. The first mode is blocking mode, which records for a predetermined duration of time; the second mode is callback mode, which starts recording when a function is called, and stops when another function is called. The second mode is appropriate in Mini Translator, since the GUI will allows the script to start recording when the user press a button, and stop recording when the user press another button.

Besides the features mentioned above, all of these scripts are modified following Object-Oriented Programming principles. Each script is a class, and all functions and variables are members of the class. The classes provide several functions to be its "private" member functions to do operations, although in Python the concept of "private" is vague. Each class provides several "public" functions as Interface for users to utilize different functionalities, as well as functions to modify some variables in the class. Such modifications help the program to use GUI to control the functionality of each script. Even without GUI, Object-Oriented Programming

style also helps to make the code neat and clear. Everyone, without the knowledge how exactly the code works, can use the code to do stuffs, as long as he or she strictly follows the instruction of using interface given by the programmer.

The GUI.py is the core of Mini Translator. It provides a graphic interface for users to interact with Mini Translator. GUI.py, importing use Labels, Buttons, File Selectors, Drop-down Option Menus, Text Entries and other elements to help users to understand those interactions. With the graphic interface, the user can select input file and select languages by clicking mouse, and the output is generated within seconds, under the same directory of the input file. The main menu of Mini Translator is shown by *Figure 3*.

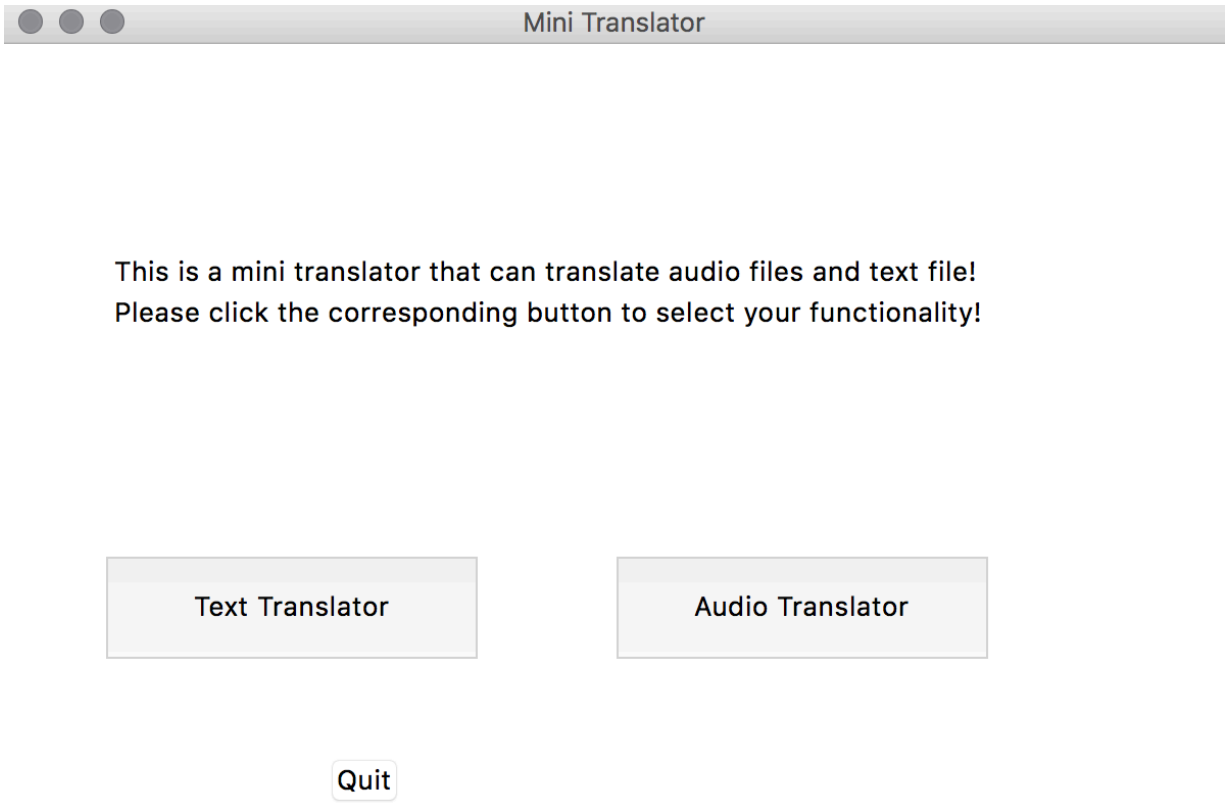

Figure 3: Main Menu of Mini Translator

The title of the main menu is Mini Translator. The text labels in the middle introduce the basic functionalities of Mini Translator, and briefly tell users how to use different functionalities—by simply clicking corresponding buttons. There are three buttons in the main menu. The "Text Translator" buttons will open another window called "Text Translator", which will allow user to select target languages and select the file to be translated. The window will then further run the code in classTextTranslate.py to do the stuffs. The "Audio Translator" buttons will open another window called "Audio Translator", which will allow user to select input languages and output languages. This window will run the code in classAudioTranslate.py and classAudioRecord.py and do the stuffs. The "Quit" button will close Mini Translator.

Figure 4: Text Translator Window of Mini Translator

When "Text Translator" button is clicked, a new "Text Translator" window will be opened. Text Translator Window is shown as *Figure 4*. The window, a TopLevel Window of the main menu, contains more elements compared to the main menu. The text labels on the upper half part give descriptions about the window and some instructions. The Drop-down Menu Bar is for users to select their target languages. *Table 1* gives a full list of languages supported by Mini Translator currently and their language codes. Below the Drop-down Menu Bar, a Text Entry is for the user to put in new subscription keys. Once the "Renew Key" button is pressed, the new keys will replace the default subscription keys before the translation work begins. The "Select Input File" button will open a File Selector Window, allowing user to select the input file (file user want to translate) from any directory of the Computer. A StringVar() from Tkinter is used to record down the input file name. When the script is creating a TranslateClass() instance, input file name, the automatically-generated output file name, and the target languages will passed to the constructor of the instance, and the rest of work will be done by classTextTranslate.py script. The "Return Button" will close current window and return to main menu.

Language	Code	Code and Region
Arabic	ar	ar-AR
Chinese	zh	zh-CN
English	en	en-US
French	fr	fr-FR
German	de	de-DE
Italian	it	it-IT
Japanese	ja	ja-JA
Portuguese	pt	pt-PT
Russian	ru	ru-RU
Spanish	es	es-ES

Table 1. Languages supported by Mini Translator

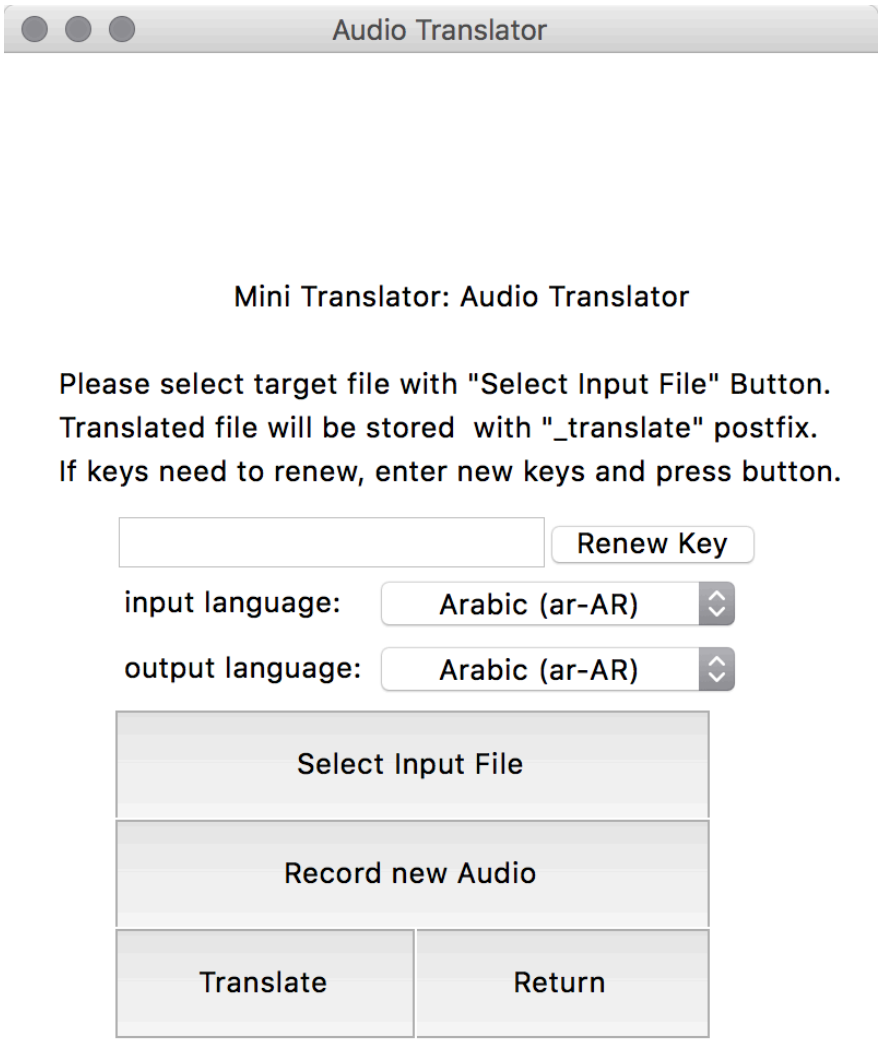

Figure 5. Audio Translator Window of Mini Translator

The “Audio Translator” window, as shown in *Figure 5,* contains even more elements. Similar to previous windows, the upper half is mainly label descriptions, Text Entry to renew subscription keys, Drop-down Option Bars to select languages. Note that in Translate Audio, both the input and the output language must be selected for the API to work properly. “Select Input File”, “Translate” and “Return” buttons are all doing similar works. The only difference here is the new button, “Record new Audio”. When the button is pressed, a new Window is opened. The window uses the functions in classAudioRecord.py to create a new wave file and record down users’ speeches. The new wave file then is fed as input to classAudioTranslate.py, and the returned file from the server is also created, containing the translated speech.

In conclusion, Mini Translator four Python scripts. Three of them are scripts that implement the functionalities, and the remaining script provides an easy mean for user to interact wit the program. The classes communicate with each other by different function call. The communication UML of Mini Translator is shown as *Figure 6.*

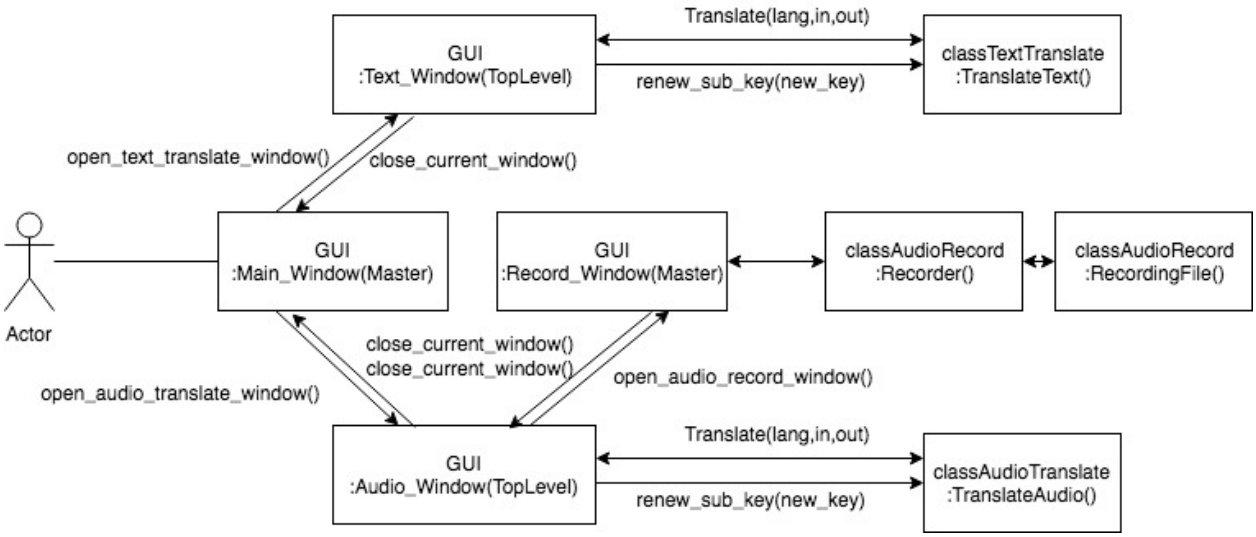

Figure 6: Communication UML Diagram of Mini Translator

Principles

The core technology of Mini Translator is Machine Translation. Machine translation systems are applications or online services that use machine-learning technologies to translate large amounts of text from and to any of their supported languages. Although the concepts behind machine translation technology and the interfaces to use it are relatively simple, the science and technologies behind it are extremely complex and bring together several leading-edge technologies, in particular, deep learning (artificial intelligence), big data, linguistics, cloud computing, and web APIs. Mini Translator utilizes Microsoft web APIs to do the translation work, as shown in *Figure 7*.

Historically, the primary machine learning technique used in the industry was Statistical Machine Translation (SMT). SMT uses advanced statistical analysis to estimate the best possible

translations for a word given the context of a few words. SMT has been used since the mid-2000s by all major translation service providers, including Microsoft. The advent of Neural Machine Translation (NMT) caused a radical shift in translation technology, resulting in much higher quality translations. This translation technology started deploying for users and developers in the latter part of 2016.

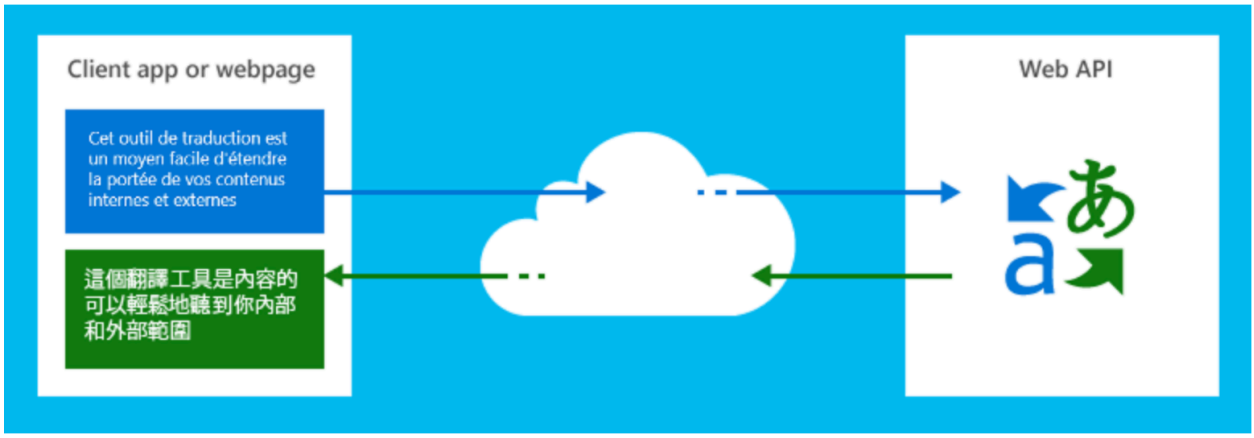

Figure 7. Mini Translator utilizes web APIs to do the translation work.

Both SMT and NMT translation technologies have two elements in common: Both require large amounts of pre-human translated content (up to millions of translated sentences) to train the systems. Neither act as bilingual dictionaries, translating words based on a list of potential translations, but translate based on the context of the word that is used in a sentence. However, there differences are more noteworthy.

In SMT, a so-called "parallel corpora" plays an important role in word, phase and idiomatic translations for many pairs of languages. SMT techniques and some other algorithms help the computer address the problems of decipherment, in other words, finding the correspondence to target languages and source languages, in order to generate better and more comprehensible translations. The translation does not merely rely on dictionaries and grammatical rules.

In NN Translation, however, things work differently. The approach takes into context the full sentence and produces more fluid translations. Each world is coded by a 500-dimension and representing its unique characteristic with a language pair. *Figure 8* and the following steps give a detailed explanation on how NN Translation works.

1. Each word, or more specifically the 500-dimension vector representing it, goes through a first layer of “neurons” that will encode it in a 1000-dimension vector (b) representing the word within the context of the other words in the sentence.

2. Once all words have been encoded one time into these 1000-dimension vectors, the process is repeated several times, each layer allowing better fine-tuning of this 1000-dimension representation of the word within the context of the full sentence (contrary to SMT technology that can only take into consideration a 3 to 5 words window)

3. The final output matrix is then used by the attention layer (i.e. a software algorithm) that will use both this final output matrix and the output of previously translated words to define which word, from the source sentence, should be translated next. It will also use these calculations to potentially drop unnecessary words in the target language.

4. The decoder (translation) layer translates the selected word (or more specifically the 1000-dimension vector representing this word within the context of the full sentence) in its most appropriate target language equivalent. The output of this last layer (c) is then fed back into the attention layer to calculate which next word from the source sentence should be translated.

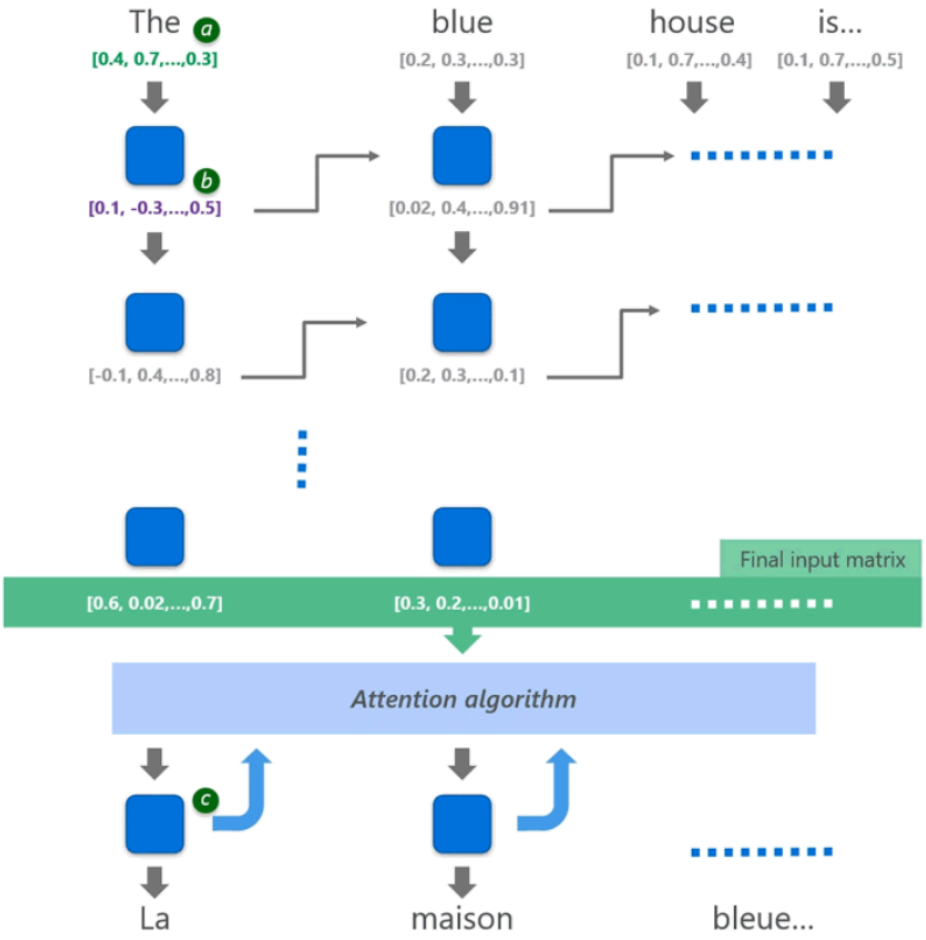

Figure 8. NN Translation graphic explanation.

In the example above, since the words are taken consideration in the context of the whole sentence, "maison", the feminine world in french will follows a feminine article "la", and considering the different adjective order in French, the final output will be "La maison bleue".

The Translator Speech API is an API that can translate a speech of a certain language to other languages. The raw text output is further improved by removes dis-fluencies and repeated words. Meanwhile, masking or removing profanities is also included. The API can handle conversational speech. The services detect silence to detect the end of a speech. To properly translate the "source" speech from one language to a different "target" language, the system goes through a four-step process. *Figure 9* gives a graphic explanation.

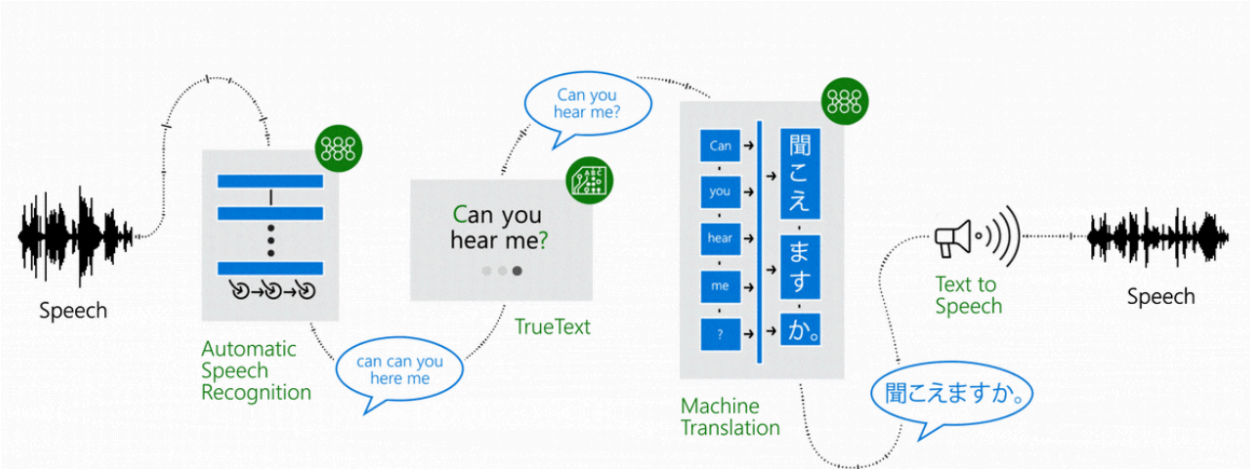

Figure 9: Graphic Explanation of Audio Translate Process.

1. Speech recognition, to convert audio in into text. Automatic Speech Recognition (ASR) is performed using a neural network D(NN) system trained on analyzing thousands of hours of incoming audio speech. This model is trained on human-to-human interactions rather than human-to-machine commands, producing speech recognition that is optimized for normal conversations. To achieve this, much more data is needed as well as a larger DNN than traditional human-to-machine ASRs.

2. TrueText: A Microsoft technology that normalizes the text to make it more appropriate for translation. As humans conversing with other humans, we don't speak as perfectly, clearly or neatly as we often think we do. With the TrueText technology, the literal text is transformed to more closely reflect user intent by removing speech disfluencies (filler words), such as "um"s, "ah"s, "and"s, "like"s, stutters, and repetitions. The text is also made more readable and translatable by adding sentence breaks, proper punctuation, and capitalization. To achieve these results, we used the decades of work on language technologies; we developed from Translator to create TrueText. The following diagram depicts, through a real-life example, the various transformations TrueText operates to normalize this literal text.

3. Translation through the text translation engine described above but on translation models specially developed for real life spoken conversations. Translations using the speech translation API (as a developer) or in a speech translation app or service, is powered with the newest neural-network based translations for all the speech-input supported languages (see here for the full list). These models were also built by expanding the current, mostly written-text trained translation models, with more spoken-text corpora to build a better model for spoken conversation types of translations. These

models are also available through the “speech” standard category of the traditional text translation API.

4. Text-to-speech, when necessary, to produce the translated audio. If the target language is one of the 18 supported text-to-speech languages, and the use case requires an audio output, the text is then converted into speech output using speech synthesis. This stage is omitted in speech-to-text translation scenarios.

Conclusion

In conclusion, Machine translation systems are applications or online services that use machine-learning technologies to translate large amounts of text from and to any of their supported languages. The service translates a “source” text from one language to a different “target” language. The Text Translation API allows user to take text messages (either a simple sentence, or more complicate text files) of a certain languages as inputs and translate them into one or more other languages as outputs. The Translator Speech API can translate the speech of a language to another languages. I think the API can be used with other things, such as voice change system, the text-to-audio API and other things to create a variety number of new services. However, there are also some drawbacks. For example, when a language is not spoken by a native speaker, there will be error on the convert between text and speech.

Mini Translator, with the combinations of APIs mentioned above, is a preliminary attempt on the corroboration of different APIs within Microsoft. Since those APIs are relatively new, some problems still exist and need to be improved or corrected. The Audio Translate part only accepts a specific format of wave input. If audios in format such as MP3 or other unsupported formats need to be translated, users have to first find ways to do format-conversion. That is inconvenient in users perspective. The whole bunch of service is not free. Users have to purchase subscription keys to enjoy the services and the key need to be replaced frequently… All of these are current disadvantages and places that need to be improved

In the future, as the development of modern society, there are more chances for ordinary people to meet foreigners from other cultures, speaking different languages. Translator should be smaller in size, faster in translation, more accurate in translated languages, and cheaper in price. However, the best way to due with the situation is to learn other languages. Under any situations, it is only better for a person to be fluent in more languages.

Reference

https://docs.microsoft.com/zh-cn/azure/cognitive-services/translator-speech/
https://docs.microsoft.com/zh-cn/azure/cognitive-services/translator/
https://www.microsoft.com/en-us/translator/business/machine-translation/
https://people.csail.mit.edu/hubert/pyaudio/docs/#platform-specific

Appendix

Source code of Mini Translator.

classTextTranslate.py

```
import http.client
import urllib.parse
import uuid
import json

import ssl
ssl._create_default_https_context = ssl._create_unverified_context


class TranslateText:

    _subscription_keys = '0579518fc929440cbcbe711aa5b1add3'
    _host = 'api.cognitive.microsofttranslator.com'
      path = '/translate?api-version=3.0'
    _params = ""
    _input_file = None
    _output_file = None

    def __init__(self, params, input, output):

        self._params = "&to=" + params

        try:
            self._input_file = open(input, 'r')
        except IOError:
            print("Error! Could not open the input file")
            exit(1)

        try:
            self._output_file = open(output, 'w+')
        except IOError:
            print("Error! Could not open the input file")
            exit(1)

        return

    def __del__(self):

        self._output_file.close()
        self._input_file.close()

    def _translate_(self, content):
        headers = {
            'Ocp-Apim-Subscription-Key': self._subscription_keys,
            'Content-type': 'application/json',
            'X-ClientTraceId': str(uuid.uuid4())
        }

        conn = http.client.HTTPSConnection(self._host)
        conn.request("POST", self._path + self._params, content, headers)
        response = conn.getresponse()
        return response.read().decode('utf-8')

    def renew_sub_key(self, new_sub_keys):
        self._subscription_keys = new_sub_keys
```

```
    def translate(self):
        for line in self._input_file:
            if line == '\n':
                print("Receive a new line input")
                self._output_file.write('\n')
                continue

            requestBody = [{'Text': line, }]
            content = json.dumps(requestBody, ensure_ascii=False).encode('utf-8')
            result = self._translate_(content)
            # Note: We convert result, which is JSON, to and from an object so we can pretty-print it.
            # We want to avoid escaping any Unicode characters that result contains. See:
            # https://stackoverflow.com/questions/18337407/saving-utf-8-texts-in-json-dumps-as-utf8-not-as-u-escape-sequence
            output = json.dumps(json.loads(result), indent=4, ensure_ascii=False)
            print(output)

            translated_text = json.loads(result)[0]['translations'][0]['text']
            self._output_file.write(translated_text)
```

classAudioTranslate.py

```
import websocket
import ssl


class AudioTranslate:

    _subscription_keys = '801361cdf47544eeadd940b6923f8e1e'
    _host = 'wss://dev.microsofttranslator.com'
    _path = '/speech/translate'
    _params = ''  # '?api-version=1.0&from=en-US&to=it-IT&features=texttospeech&voice=it-IT-Elsa'
    _uri = ''  # host + path + params

    _input_file = None
    _output_file = None

    _output = None
     client = None

    def     init    (self, input, output, in  l='en-US', out  l='zh-CN'):

        self._params = '?api-version=1.0&from=' + in_l + '&to=' + out_l + '&features=texttospeech'
        self._uri = self._host + self._path + self._params

        self._input_file = input
        self._output_file = output

        self._output = bytearray()

        self._client = websocket.WebSocketApp(
            self.  uri,
            header=[
                'Ocp-Apim-Subscription-Key: ' + self._subscription_keys
            ],
            on_open=self.on_open,
            on_data=self.on_data,
            on  error=self.on  error,
            on_close=self.on_close
```

```
        )

    def __del__(self):
        self._input_file.close()

    def on_open(self, client):
        print("Connected.")

        # r = read. b = binary.
        with open(self._input_file, mode='rb') as file:
            data = file.read()

        print("Sending audio.")
        client.send(data, websocket.ABNF.OPCODE_BINARY)
        # Make sure the audio file is followed by silence.
        # This lets the service know that the audio input is finished.
        print("Sending silence.")
        client.send(bytearray(32000), websocket.ABNF.OPCODE_BINARY)

    def on_data(self, client, message, message_type, is_last):

        # global output
        if websocket.ABNF.OPCODE_TEXT == message_type:
            print("Received text data.")
            print(message)
        # For some reason, we receive the data as type websocket.ABNF.OPCODE  CONT.
        elif websocket.ABNF.OPCODE_BINARY == message_type or websocket.ABNF.OPCODE_CONT == message_type:
            print("Received binary data.")
            print("Is last? " + str(is_last))
            self._output = self._output + message
            if (True == is_last):
                # w = write. b = binary.
                with open(self._output_file, mode='wb') as file:
                    file.write(self._output)
                    print("Wrote data to output file.")
                client.close()
        else:
            print("Received data of type: " + str(message_type))

    def on_error(self, client, error):
        print("Connection error: " + str(error))

    def on_close(self, client):
        print("Connection closed.")

    def run(self):
        print("Connecting...")
        self._client.run_forever(sslopt={"cert_reqs": ssl.CERT_NONE})
```

classAudioRecord.py

```
# -*- coding: utf-8 -*-

import pyaudio
import wave
import time


class Recorder(object):
    '''A recorder class for recording audio to a WAV file.
    Records in mono by default.
    '''
```

```
    def __init__(self, channels=1, rate=16000, frames_per_buffer=1024):  # original rate=44100. new rate=16000
        self.channels = channels
        self.rate = rate
        self.frames_per_buffer = frames_per_buffer

    def open(self, fname, mode='wb'):
        return RecordingFile(fname, mode, self.channels, self.rate,
                             self.frames_per_buffer)

class RecordingFile(object):
    def __init__(self, fname, mode, channels,
                 rate, frames_per_buffer):
        self.fname = fname
        self.mode = mode
        self.channels = channels
        self.rate = rate
        self.frames_per_buffer = frames_per_buffer
        self._pa = pyaudio.PyAudio()
        self.wavefile = self._prepare_file(self.fname, self.mode)
        self._stream = None

    def __enter__(self):
        return self

    def __exit__(self, exception, value, traceback):
        self.close()

    def record(self, duration):
        # Use a stream with no callback function in blocking mode
        self._stream = self._pa.open(format=pyaudio.paInt16,
                                     channels=self.channels,
                                     rate=self.rate,
                                     input=True,
                                     frames_per_buffer=self.frames_per_buffer)
        for _ in range(int(self.rate / self.frames_per_buffer * duration)):
            audio = self._stream.read(self.frames_per_buffer)
            self.wavefile.writeframes(audio)
        return None

    def start_recording(self):
        # Use a stream with a callback in non-blocking mode
        self._stream = self._pa.open(format=pyaudio.paInt16,
                                     channels=self.channels,
                                     rate=self.rate,
                                     input=True,
                                     frames_per_buffer=self.frames_per_buffer,
                                     stream_callback=self.get_callback())
        self._stream.start_stream()
        return self

    def stop_recording(self):
        self._ stream.stop  stream()
        return self

    def get_callback(self):
        def callback(in_data, frame_count, time_info, status):
            self.wavefile.writeframes(in_data)
            return in_data, pyaudio.paContinue
        return callback

    def close(self):
```

```
        self._stream.close()
        self._pa.terminate()
        self.wavefile.close()

    def _prepare_file(self, fname, mode='wb'):
        wavefile = wave.open(fname, mode)
        wavefile.setnchannels(self.channels)
        wavefile.setsampwidth(self._pa.get_sample_size(pyaudio.paInt16))
        wavefile.setframerate(self.rate)
        return wavefile
```

GUI.py

```
import classTextTranslate
import classAudioTranslate
import  classAudioRecord

# import everything from tkinter
from tkinter import *
from tkinter.filedialog import askopenfilename

OPTIONS_txt = [
    "Arabic (ar)",
    "Chinese (zh)",
    "English (en)",
    "French (fr)",
    "German (de)",
    "Italian (it)",
    "Japanese (ja)",
    "Portuguese (pt)",
    "Russian (ru)",
    "Spanish (es)"
    ]

OPTIONS_wav = [
    "Arabic (ar-AR)",
    "Chinese (zh-CN)",
    "English (en-US)",
    "French (fr-FR)",
    "German (de-DE)",
    "Italian (it-IT)",
    "Japanese (ja-JA)",
    "Portuguese (pt-PT)",
    "Russian (ru-RU)",
    "Spanish (es-ES)"
    ]

class Main_Window(Frame):

    _text_window = None
    _audio_window = None

    _label_msg_1 = "This is a mini translator that can translate audio files and text file!"
      label  msg  2 = "Please click the corresponding button to select your functionality!"

    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.master = master
        self.init_window()

    # specification of some details of the code
    def init_window(self):
```

```
        # changing the title of our master widget
        self.master.title("Mini Translator")

        # making the size of the window
        self.master.geometry("600x400")

        # allowing the widget to take the full space of the root window
        self.pack(fill=BOTH, expand=1)

        # creating button instance
        label_1 = Label(self, text=self._label_msg_1)
        label_2 = Label(self, text=self._label_msg_2)

        button_1 = Button(self, text="Text Translator", command=self.open_text_translate_window, height=3, width=20)
        button_2 = Button(self, text="Audio Translator", command=self.open_audio_translate_window, height=3, width=20)
        quit_button = Button(self, text="Quit", command=self.self_exit)

        # placing elements on my window
        label_1.place(x=50, y=100)
        label_2.place(x=50, y=120)
        button_1.place(x=50, y=250)
        button_2.place(x=300, y=250)
        quit_button.place(x=160, y=350)

    def open_text_translate_window(self):
        self.  text  window = Text  Window(self)

    def open  audio  translate  window(self):
        self._audio_window = Audio_Window(self)

    def self_exit(self):
        exit(0)


# This is  the definition of the audio translation button

class Audio_Window(Toplevel):

      input  file  name = ''
    _output_file_name = ''

    _in_language_sel = None
    _out_language_sel = None
    _new_sub_keys = None

    _record_window = None

    REPLACE = False

    _label_msg_1 = 'input language:'
    _label_msg_2 = 'output language:'
    _msg1 = 'Please select target file with "Select Input File" Button.'
      msg2 = 'Translated file will be stored  with "  translate" postfix.'
    _msg3 = 'If keys need to renew, enter new keys and press button.'
    _msg4 = 'Mini Translator: Audio Translator'

    def __init__(self, master):
        Toplevel.__init__(self)
        self.master = master
        self.init_window()

    def init_window(self):
```

```
    self.title("Audio Translator")
    self.geometry("400x500")

    button_1 = Button(self, text="Select Input File", command=self.open_file_selection_window, height=3, width=30)
    button_2 = Button(self, text="Record new Audio", command = self.open_new_record_window, height=3, width=30)
    button_3 = Button(self, text="Return", command=self.close_current_window, height=3, width=15)
    button_4 = Button(self, text="Translate", command=self.translate, height=3, width=15)
    button_5 = Button(self, text="Renew Key", command=self.renew_sub_key, height=1, width=10)

    label_1 = Label(self, text=self._label_msg_1)
    label_2 = Label(self, text=self._label_msg_2)
    label_3 = Label(self, text=self._msg1)
    label_4 = Label(self, text=self._msg2)
    label_5 = Label(self, text=self._msg3)
    label_6 = Label(self, text=self._msg4)

    self._in_language_sel = StringVar()
    self._out_language_sel = StringVar()
    self._in_language_sel.set(OPTIONS_wav[0])
    self._out_language_sel.set(OPTIONS_wav[0])

    options_1 = OptionMenu(self, self._in_language_sel, *OPTIONS_wav)
    options_2 = OptionMenu(self, self._out_language_sel, *OPTIONS_wav)
    options_1.config(width=15)
    options_2.config(width=15)

    self._new_sub_keys = StringVar()
    new_keys = Entry(self, textvariable=self._new_sub_keys)

    button_1.place(x=50, y=300)
    button_2.place(x=50, y=350)
    button_3.place(x=185, y=400)
    button_4.place(x=50, y=400)
    button_5.place(x=250, y=215)

    label_1.place(x=50, y=240)
    label_2.place(x=50, y=270)
    label_3.place(x=20, y=140)
    label_4.place(x=20, y=160)
    label_5.place(x=20, y=180)
    label_6.place(x=100, y=100)

    options_1.place(x=170, y=240)
    options_2.place(x=170, y=270)

    new_keys.place(x=50, y=210, width=200)

def close_current_window(self):
    self.destroy()

def open_file_selection_window(self):

    self._input_file_name = askopenfilename(initialdir="/", title="Select inout file",
                                            filetypes=(("wave files", "*.wav"), ("all files", "*.*")))

    self._output_file_name = self._input_file_name[:-4] + '_translate.wav'

def open_new_record_window(self):
    self._record_window = Record_Window(self)

def renew_sub_key(self):
```

```
        self.REPLACE = TRUE

    def translate(self):
        _in = self._in_language_sel.get()[-6:-1]
        _out = self._out_language_sel.get()[-6:-1]

        translate = classAudioTranslate.AudioTranslate(self._input_file_name, self._output_file_name, _in, _out)

        translate.run()

# This is the definition of the text translation button


class Text_Window(Toplevel):

    _input_file_name = ''
    _output_file_name = ''

    _msg1 = 'Please select target file with "Select Input File" Button.'
    _msg2 = 'Translated file will be stored  with "_translate" postfix.'
    _msg3 = 'If keys need to renew, enter new keys and press button.'
    _msg4 = 'Mini Translator: Text Translator'

      language  selection = None
    _new_sub_keys = None

    REPLACE = False

    def __init__(self, master):
        Toplevel.__init__(self)
        self.master = master
        self.init_window()

    def init_window(self):

        self.title("Text Translator")
        self.geometry("400x400")

        button  1 = Button(self, text="Select Input File", command=self.open  file  selection  window, height=3, width=30)
        button_2 = Button(self, text="Translate", command=self.translate, height=3, width=15)
        button_3 = Button(self, text="Return", command=self.close_current_window, height=3, width=15)
        button_4 = Button(self, text="Renew Key", command=self.renew_sub_key, height=1, width=10)

        label_1 = Label(self, text=self._msg1)
        label  2 = Label(self, text=self.  msg2)
        label_3 = Label(self, text=self._msg3)
        label_4 = Label(self, text=self._msg4)

        self._language_selection = StringVar()
        self._language_selection.set(OPTIONS_txt[0])
        options = OptionMenu(self, self._language_selection, *OPTIONS_txt)
        options.config(width=27)

        self._new_sub_keys = StringVar()
        new_keys = Entry(self, textvariable=self._new_sub_keys)

        button_1.place(x=50, y=250)
        button_2.place(x=50, y=300)
        button_3.place(x=185, y=300)
        button_4.place(x=250, y=215)

        label_1.place(x=20, y=100)
```

```
        label_2.place(x=20, y=120)
        label_3.place(x=20, y=140)
        label_4.place(x=100, y=50)

        options.place(x=50, y=180)
        new_keys.place(x=50, y=210, width=200)

    def open_file_selection_window(self):

        self._input_file_name = askopenfilename(initialdir="/", title="Select inout file",
                                    filetypes=(("text files", "*.txt"), ("all files", "*.*")))

        self._output_file_name = self._input_file_name[:-4] + '_translate.txt'

    def translate(self):
        language = self._language_selection.get()[-3:-1]
        translate = classTextTranslate.TranslateText(language, self._input_file_name, self._output_file_name)

        if self.REPLACE:
            translate.renew_sub_key(self._new_sub_keys.get())

        translate.translate()

    def renew_sub_key(self):
        self.REPLACE = TRUE

    def close_current_window(self):
        self.destroy()


class Record_Window(Toplevel):

    def __init__(self, master):
        Toplevel.__init__(self)
        self.master = master
        self.init_window()

    def init_window(self):
        self.title("Recorder")
        self.geometry("400x400")


main_window_frame = Tk()  # Create the root window
main_window = Main_Window(main_window_frame)  # Create the instance of the root window.
main_window.mainloop()
```