# 高雄科技大學

# 智慧商務系

## AZURE 雲端服務認證與證照

## 第二次期中專題

# 影片字幕產生器

姓名：許智程
學號：C109152304

指導老師：謝文川 老師

# 目錄

**目錄** i

1 **研究動機與目的** 1

2 **專案規劃與設計** 1
2.1 專案架構簡介 . . . . . 1
2.2 相關組件與服務 . . . . . 1
2.2.1 FFmpeg . . . . . 1
2.2.2 Azure Cognitive Services . . . . . 2

3 **程式實作** 2
3.1 抽取音檔 . . . . . 2
3.2 從音檔生成字幕 . . . . . 3
3.3 翻譯得到的字幕檔 . . . . . 6
3.4 把字幕跟影片結合 . . . . . 7
3.5 Log 與 Decorator . . . . . 8

4 **後記** 9

**參考資料** 10

# 1 研究動機與目的

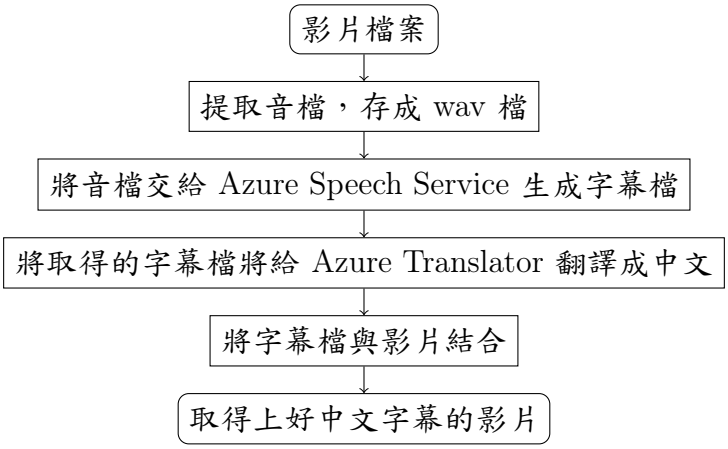
學習資訊技術時，常常得面臨查到的資料只有英文的部分，如果是文章的話當看不懂時還可以用 Chrome 的翻譯功能將文章翻譯成中文幫助理解，不過文章看久了總想看看別人實際下來實作，而在網路上多數的英文實作影片幾乎沒有中文字幕，還得面臨外國佬的口音很重的問題。

除了資訊技術的影片外，很多中文以外的內容通常也是蠻有料的，難道要因為語言的隔閡去放棄這些內容嗎？

為此，本次專題的目標就是利用在課堂上所學的 Azure AI 技巧來達成自動為為影片生成字幕檔，並翻譯成中文，最後直接將字幕嵌到影片中。

# 2 專案規劃與設計

## 2.1 專案架構簡介




如上面的流程圖所示，首先會將影片的音檔提取出來，並存成 wav 檔，方便後續處理，接下來藉由 Azure Captioning 和 Azure Translation 來取得字幕並翻譯成中文，不過得到的結果會將字幕格式檔中紀錄時間的戳記中的標點符號全部由半形轉成全形，需要再經過處理，處理過後再將影片檔與字幕檔結合在一起就能取得上好中文字幕的影片了。

除此之外，還引入了 Log 的機制，紀錄了函數開始的時間與結束的時間，並且計算了耗費的秒數，下面將會簡單介紹所運用到的技術。

## 2.2 相關組件與服務

### 2.2.1 FFmpeg

FFmpeg 是一個開放原始碼的自由軟體，它包含了音訊和視訊多種格式的錄影、轉檔、串流功能，同時也是一個音訊與視訊格式轉換函式庫 (Library)，許多開源的工具都是基於 FFmpeg 打造的。 FFmpeg 主要由以下幾個元件組成：

- CLI Application
  - ffmpeg：用來轉換不同格式的多媒體檔案，為 FFmpeg 最主要的工具
  - ffplay：基於 SDL(Simple DirectMedia Layer) 和 FFmpeg 函式庫的簡易播放器 [1]
  - ffprobe：簡易的多媒體串流分析器

- Libraries
  - libavcodec：用來編碼與解碼不同的影音格式
  - libavformat：用來處理承載影音串流的多媒體容器
  - libavutil：一些程式上會用到的函式與資料結構
  - libpostproc：對於視訊做前處理的函式庫
  - libswscale：對於影像作縮放的函式庫
  - libavfilter：對於影片與音訊做進一步處理，如倍速、水平翻轉、裁剪、加方框、疊加文字等功能
  - libswresample：對於很多播放器，在輸出時會固定爲一種格式（如 44100hz，雙聲道，16bit signed），因爲多數設備能夠支持這些格式。這種情況下對於不同的多種輸入源，即需要進行音頻重採樣。而相對於舊式的 libavcodec，libswresample 能更輕鬆的處理。

運用這些程式與函數庫，我們可以方便地進行影音處理，例如說：從影片中提取音檔、將字幕嵌到影片中、影音轉檔..... 等都可以藉由 FFmpeg 完成，甚至可以用 FFmpeg 來剪片！

FFmpeg 如此的強大，可想而知的其複雜度也是不容小覷，於是不論是免費的還是付費的，許多人都對 FFmpeg 進行一定的封裝產生許多影音處理工具，連 python 都有輔助使用的 package 了呢！

在這次專題中，會藉由 ffmpeg-python[2] 這個在 python 中的輔助工具來進行取得影片的聲音檔與將字幕與影片檔壓在一起的動作。

#### 2.2.2 Azure Cognitive Services

Azure Cognitive Services 是雲端式的 AI 服務，可使開發人員無須具備 AI 或 Data Science 的相關知識就可以將相關技術整合道專案中。這些服務可透過熱門開發語言 (比如說：C#、Java、Python、JavaScript、golang..... 等) 的 REST API 和 SDK 取得。Azure Cognitive Services 可讓開發人員使用認知解決方案，輕鬆地將認知功能新增至其應用程式，以進行查看、聆聽、說話和分析。

在這次專題中，會使用到 Azure Cognitive Services 中的 Speech Service 與 Translator 來達成生成影片的中文字幕的目的。

# 3 程式實作

## 3.1 抽取音檔

在 ffmpeg 中要拿到影片的音檔可說是非常容易，可以視作將影片轉成聲音檔，也就是說，輸入填上影片檔的路徑，輸出檔名只要是聲音檔的附檔名就可以了。為了方便進行下一步，我選擇轉成 wav 檔。如下面的程式所示：

```
from utils import Logger
import ffmpeg


@Logger
def splitwav(orginFilePath, targetFilePath):
    (
        ffmpeg.input(orginFilePath)
        .output(targetFilePath)
        .overwrite_output()
```

```
        .run()
    )
```

咦 @Logger 是甚麼？簡單來說，這是用一種叫做 Decorator 的技巧來讓紀錄程式執行的 Code 能抽成一個函數獨立出來，而不用黏在每一個函數之中。後面會再詳細說明。

## 3.2 從音檔生成字幕

很幸運的，在微軟的文檔 [3] 中有相關的範例，不過在看過範例程式碼後，發現他們是做成 CLI Tool 的形式，為了更好的與其他部分結合，需要動點手腳，範例 Code 節錄如下：

```
#
# Copyright (c) Microsoft. All rights reserved.
# Licensed under the MIT license. See LICENSE.md file in the project root for full license information.
#

# Notes:
# - Install the Speech SDK. Run:
# pip install azure-cognitiveservices-speech
# - The Python Speech SDK on Windows requires the Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017, 2019, or 2022 on the system. See:
# https://docs.microsoft.com/azure/cognitive-services/speech-service/quickstarts/setup-platform
# - Install gstreamer:
# https://docs.microsoft.com/azure/cognitive-services/speech-service/how-to-use-codec-compressed-audio-input-streams

# 顧及篇幅，只從範例程式全部4個檔案中節錄進入點的主要Class和實際執行的程式
# 如要查看全部的Code
# 請至: https://github.com/Azure-Samples/cognitive-services-speech-sdk/tree/master/scenarios/python/console/captioning
# file：captioning.py

from datetime import datetime, time, timezone
from itertools import groupby, pairwise
from os import linesep, remove
from os.path import exists
from pathlib import Path
from sys import argv
from time import sleep
from typing import Any, List, Optional
import wave
import azure.cognitiveservices.speech as speechsdk # type: ignore
import caption_helper
import helper
import user_config_helper

class Captioning(object) :
    def __init__(self) :
        self._user_config = user_config_helper.user_config_from_args(USAGE)
        self._srt_sequence_number = 1
        self._previous_caption : Optional[caption_helper.Caption] = None
        self._previous_end_time : Optional[time] = None
        self._previous_result_is_recognized = False
        self._recognized_lines : List[str] = []
```

```
        self._offline_results : List[speechsdk.SpeechRecognitionResult] =
            []
        ...
        ...
        ...

if user_config_helper.cmd_option_exists("--help") :
    print(USAGE)
else :
    captioning = Captioning()
    captioning.initialize()
    speech_recognizer_data = captioning.speech_recognizer_from_user_config
        ()
    captioning.recognize_continuous(
        speech_recognizer=speech_recognizer_data["speech_recognizer"],
            format=speech_recognizer_data["audio_stream_format"], callback=
            speech_recognizer_data["pull_input_audio_stream_callback"],
            stream=speech_recognizer_data["pull_input_audio_stream"]
    )
    captioning.finish()
```

從上面的 Code 可以看出，實際的工作都由 Captioning 這個 Class 執行，然後分析這個 Class 的 Constructor 得知，程式執行縮需所有的參數都是由 user_config_helper.user-_config_from_args(USEAGE) 這一個函數來的，一樣從範例給出這個函數的原始碼：

```
#
# Copyright (c) Microsoft. All rights reserved.
# Licensed under the MIT license. See LICENSE.md file in the project root
    for full license information.
#

# 顧及篇幅，這裡把焦點放在傳回的東西上面
# 詳細的使用者輸入參數處理流程請參閱: https://github.com/Azure-Samples/
    cognitive-services-speech-sdk/blob/master/scenarios/python/console/
    captioning/user_config_helper.py
# file: user_config_helper.py

from datetime import timedelta
from enum import Enum
from os import linesep, environ
from sys import argv
from typing import List, Optional
import azure.cognitiveservices.speech as speechsdk # type: ignore
import helper

def user_config_from_args(usage : str) -> helper.Read_Only_Dict :
    # start process
    ...
    ...
    ...
    # end process

    return helper.Read_Only_Dict({
        "use_compressed_audio" : cmd_option_exists("--format"),
        "compressed_audio_format" : get_compressed_audio_format(),
        "profanity_option" : get_profanity_option(),
        "language" : get_language(),
        "input_file" : get_cmd_option("--input"),
        "output_file" : get_cmd_option("--output"),
        "phrases" : get_phrases(),
        "suppress_console_output" : cmd_option_exists("--quiet"),
```

```
        "captioning_mode" : captioning_mode,
        "remain_time" : td_remain_time,
        "delay" : td_delay,
        "use_sub_rip_text_caption_format" : cmd_option_exists("--srt"),
        "max_line_length" : int_max_line_length,
        "lines" : int_lines,
        "stable_partial_result_threshold" : get_cmd_option("--threshold"),
        "subscription_key" : key,
        "region" : region,
    })
```

從上面的 Code 可以看出，使用者輸入的參數在經過一連串的處理後，傳回一個自訂類別 helper.Read_Only_Dict，顧名思義，這是一個唯讀 Dict，可以保護參數在執行時不受影響。在了解上面的事情後，就可以著手改造 Code，使其能融合進專案裡面了。最後的結果如下：

```
# 顧及篇幅，只給出實際更動的部分。
# 詳細的Code請直接到captioning.py觀看

from datetime import timedelta
from datetime import time
from more_itertools import pairwise
from os import linesep, remove
from os.path import exists
from time import sleep
from typing import List, Optional
import wave
import azure.cognitiveservices.speech as speechsdk  # type: ignore
import caption_helper
import helper
import user_config_helper
from user_config_helper import CaptioningMode, get_phrases,
    get_compressed_audio_format
from utils import AzureSpeechConfig, Logger
from CAPTIONINGUSAGE import CAPTIONINGUSAGE


class Captioning(object):
    def __init__(self, input, output, language):

        # self._user_config = user_config_helper.user_config_from_args(
            CAPTIONINGUSAGE)

        self._user_config = helper.Read_Only_Dict({
            "use_compressed_audio": False,
            "compressed_audio_format": get_compressed_audio_format(),
            "profanity_option": speechsdk.ProfanityOption.Raw,
            "language": language,
            "input_file": input,
            "output_file": output,
            "phrases": get_phrases(),
            "suppress_console_output": False,
            "captioning_mode": CaptioningMode.OFFLINE,
            "remain_time": timedelta(milliseconds=1000),
            "delay": timedelta(milliseconds=1000),
            "use_sub_rip_text_caption_format": True,
            "max_line_length": helper.DEFAULT_MAX_LINE_LENGTH_SBCS,
            "lines": 1,
            "stable_partial_result_threshold": '3',
            "subscription_key": AzureSpeechConfig.subscription_key.value,
            "region": AzureSpeechConfig.region.value,
```

```
        })
        self._srt_sequence_number = 1
        self._previous_caption: Optional[caption_helper.Caption] = None
        self._previous_end_time: Optional[time] = None
        self._previous_result_is_recognized = False
        self._recognized_lines: List[str] = []
        self._offline_results: List[speechsdk.SpeechRecognitionResult] = []
        ...
        ...
        ...

@Logger
def makeSubtitles(filePath, outputPath, lang):
    if user_config_helper.cmd_option_exists("--help"):
        print(CAPTIONINGUSAGE)
    else:

        captioning = Captioning(
            input=filePath,
            output=outputPath,
            language=lang
        )
        captioning.initialize()
        speech_recognizer_data = captioning.
            speech_recognizer_from_user_config()
        captioning.recognize_continuous(
            speech_recognizer=speech_recognizer_data["speech_recognizer"],
            format=speech_recognizer_data["audio_stream_format"],
            callback=speech_recognizer_data["
                pull_input_audio_stream_callback"],
            stream=speech_recognizer_data["pull_input_audio_stream"]
        )
        captioning.finish()
```

### 3.3 翻譯得到的字幕檔

處理好字幕的部分之後，接下來就是翻譯了，不過翻譯時會連同時間軌的半形符號一起轉成全形符號，如果就這樣拿去給 ffmpeg 合成的話會出問題的！於是在翻譯結束後把音檔中被轉成全形的符號轉回半形就是首要任務了。這裡用 regex expression 去制定格式，當符合格式時才將全形符號轉回半形。完整的 Code 如下：

```
import re
import requests
import uuid
from utils import AzureTranslateConfig
from utils import Logger


@Logger
def translate(text, resource, to, outputFilePath):
    # Add your key and endpoint
    key = AzureTranslateConfig.subscription_key.value.decode('utf-8')
    endpoint = AzureTranslateConfig.endpoint.value.decode('utf-8')

    # location, also known as region.
    # required if you're using a multi-service or regional (not global)
       resource. It can be found in the Azure portal on the Keys and
       Endpoint page.
    location = AzureTranslateConfig.region.value.decode('utf-8')
```

```
    path = '/translate'
    constructed_url = endpoint + path

    params = {
        'api-version': '3.0',
        'from': resource,
        'to': to
    }

    headers = {
        'Ocp-Apim-Subscription-Key': key,
        # location required if you're using a multi-service or regional (not global) resource.
        'Ocp-Apim-Subscription-Region': location,
        'Content-type': 'application/json',
        'X-ClientTraceId': str(uuid.uuid4())
    }

    # You can pass more than one object in body.
    body = [{
        'text': text
    }]

    response = requests.post(
        constructed_url,
        params=params,
        headers=headers,
        json=body
    ).json()

    with open(outputFilePath, 'w', encoding='utf-8') as f:
        lines=response[0]['translations'][0]['text'].split('\n')
        result=[]
        for line in lines:
            if re.search('^\d+：\d+：\d+，\d+ --> \d+：\d+：\d+，\d+$',line):
                line = line.replace('：', ':')
                line = line.replace('，', ',')
            result.append(line)
        f.write('\n'.join(result))
```

## 3.4 把字幕跟影片結合

最後，就是要把字幕跟影片結合在一起了，這一步一樣可以用 ffmpeg 完成，要在 ffmpeg 中將影片跟字幕結合，需要用到 ffmpeg 的一種機制叫做 filter，filter 簡單來說就是 ffmpeg 用來處理圖片與影音編輯的工具，舉例來說處理字幕的 filter 叫做 subtitles，有了它我們就能將套字幕這件事完美融合進專題了！

不過這個 filter 輸出的影片檔是沒有聲音的，因此我們前面抽出來的音檔又能派上用場了，只要分別定義好音檔跟經過 subtitles filter 處理的影片檔，最後融合在一起就可以了。下面是完整的程式：

```
import ffmpeg
from utils import Logger

@Logger
def combine_video(filepath, filename,substitleFilePath):
    video = (
```

```
        ffmpeg.input(filepath)
        .filter('subtitles', substitleFilePath)
    )

    audio = (
        ffmpeg.input(f'./statics/{filename}.wav')
    )

    (
        ffmpeg.concat(video, audio, v=1, a=1)
        .output(f'./result/{filename}_subtitled.mp4')
        .overwrite_output()
        .run()
    )
```

### 3.5 Log 與 Decorator

為了了解程式執行的過程與每個步驟的執行時間，需要在每個函數的開頭和結為加入以下程式：

```
t1 = time.time()
# begin program
...
...
...
# end program
t2 = time.time()
print(f"cost {t2-t1}s")
```

如果要計時的函數多起來的話，那可是一項不小的功夫，於是 Decorator[4] 就派上用場了，只要寫好一個通用的計時 Decorator，將要計時的函數傳入 Decorator 中就可以幫你計時了！就像下面的 Code 所示：

```
import time
import logging


def Logger(func):
    def wrap(*args, **kwargs):
        logger = logging.getLogger()
        logger.setLevel(logging.DEBUG)
        formatter = logging.Formatter(
            '[%(levelname)s %(asctime)s] %(message)s',
            datefmt='%Y%m%d %H:%M:%S'
        )

        logStream = logging.StreamHandler()
        logStream.setLevel(logging.DEBUG)
        logStream.setFormatter(formatter)
        logger.addHandler(logStream)
        logging.info(f'function {func.__name__}() start')
        t1 = time.time()
        result = func(*args, **kwargs)
        t2 = time.time()
        logging.info(f'function {func.__name__}() end')
        logging.info(f'cost {t2-t1} s')
        logger.removeHandler(logStream)
        return result
    return wrap
```

```

@ Logger
def func1():
    print("func1 ! ")
    time.sleep(5)


@ Logger
def func2():
    print("func2 ! ")
    time.sleep(10)


func1()
func2()
```

在上面的 Code 中，除了計時之外，在輸出的時候選擇了 Logging[5] 這個套件，可以更好的顯示所需要的資訊，執行結果如下：

```
[INFO 20221213 14:45:48] function func1() start
func1 !
[INFO 20221213 14:45:53] function func1() end
[INFO 20221213 14:45:53] cost 5.001032590866089 s
[INFO 20221213 14:45:53] function func2() start
func2 !
[INFO 20221213 14:46:03] function func2() end
[INFO 20221213 14:46:03] cost 10.005817413330078 s
```

對了，像 @Decorator 的寫法是 python 提供的語法糖 (Syntactic sugar)，如果不想用這個方法寫的話，可以用下面的方式使用：

```
Logger(func1)()
Logger(func2)()
```

# 4 後記

藉由這次專題，對於 Azure 可以說更加熟悉了，相對於為了考過 AI900 背的題目，果然還是實作更有趣呢。對了，結果放在 result 資料夾中。

# 參考資料


[1] Magiclen. FFmpeg：免費開源、功能強大的影音處理框架. URL: https://magiclen.org/ffmpeg/.

[2] kkroening. Python bindings for FFmpeg - with complex filtering support. URL: https://pypi.org/project/ffmpeg-python/.

[3] Microsoft. 快速入門：使用語音轉換文字建立字幕. URL: https://learn.microsoft.com/zh-tw/azure/cognitive-services/speech-service/captioning-quickstart?tabs=windows%2Cterminal&pivots=programming-language-python.

[4] codemee. Python 裝飾器 (Decorator). URL: https://dev.to/codemee/python-decorator-3bni.

[5] shengyu7697. Python logging 日誌用法與範例. URL: https://shengyu7697.github.io/python-logging/.