



Sprint 1

8INF887 - Apprentissage profond

Sujet : Sous-titrage et traduction automatique de films/vidéos

Réalisé par : MADRIGAL Samuel, MANSOURI Salim et MARTIN Olwen

Programme : Maîtrise en Informatique (IA)

Professeur : BOUCHARD Kévin

28 mars 2025

Table des matières

1	Introduction	3
2	Travail Réalisé	4
2.1	Salim	4
2.2	Olwen	8
2.3	Samuel	9
3	Planification pour le prochain Sprint	11
4	Bilan	13

Table des figures

2.1	Extraction de l'audio via ffmpeg	4
2.2	Transcription avec whisper turbo	4
2.3	Segmentation avec whisper turbo	5
2.4	Résultat de la transcription (audio en français)	5
2.5	Résultat de la segmentation (format SRT)	6
2.6	Traduction en anglais avec Helsinki-NLP	7
2.7	Résultat de la traduction	7
2.8	Exemple de script avec Moviepy	8
2.9	Fonction de nettoyage partie 1	9
2.10	Fonction de nettoyage partie 2	9
2.11	Exemple fichier SRT	10
2.12	Transcription nettoyée	10
3.1	Première Architecture (Sam)	11
3.2	Deuxième Architecture (Salim)	11

Chapitre 1

Introduction

Dans le cadre de notre projet en apprentissage profond, nous avons choisi de nous pencher sur le sous-titrage et la traduction automatique de films et vidéos, un projet que nous développerons en Python.

Pour ce premier sprint, chaque membre de l'équipe explorera plusieurs pistes afin que nous puissions retenir celles qui se révèlent les plus prometteuses.

Il est important de noter que, dans l'industrie cinématographique, tous les sous-titres ne sont pas générés instantanément par une IA à partir d'une piste audio. En réalité, une part significative des sous-titres est produite – ou du moins validée – en amont, de manière semi-automatique ou entièrement manuelle, par des professionnels.

Dans notre cas, nous ambitionnons d'automatiser l'intégralité du processus de sous-titrage et de traduction, sans recourir à des corrections manuelles ou semi-automatiques.

Afin de faciliter la collaboration et d'assurer un suivi rigoureux de l'avancement du projet, nous avons mis en place un dépôt Git pour le partage du code et la gestion du versionnement. Par ailleurs, Discord nous a permis de fluidifier les échanges et la coordination au sein de l'équipe, tandis qu'un document Google Docs partagé a facilité la rédaction collective des rapports de sprint, lesquels ont ensuite été formalisés en LaTeX.

Chapitre 2

Travail Réalisé

2.1 Salim

1. Téléchargement de ffmpeg, un outil permettant d'extraire la bande sonore des vidéos, notamment en entrant la ligne suivante dans notre terminal :

```
ffmpeg -i Test_video.mp4 -vn -acodec pcm_s16le -ar 16000 -ac 1 output_audio.wav
```

FIGURE 2.1 – Extraction de l'audio via ffmpeg

- **i Test_video.mp4** : Spécifier le fichier vidéo source.
- **vn** : Désactiver le traitement vidéo.
- **acodec pcm_s16le** : Encoder l'audio en PCM 16 bits, format standard pour le WAV.
- **ar 16000** : Définir le taux d'échantillonnage à 16 kHz, ce qui est recommandé pour les modèles ASR (Automatic Speech Recognition)
- **ac 1** : Convertir l'audio en mono.
- **output_audio.wav** : Nom du fichier de sortie au format WAV

2. Réalisation d'un test de transcription et de segmentation audio avec le modèle ASR Whisper Turbo d'OpenAI.

```
class Transcribe:
    def __init__(self, model, language: str, output_file: str = "output_transcription.txt"):
        self.model = model
        self.language = language
        self.output_file = output_file

    def transcribe_audio(self, audio_file: str) -> dict:
        # Transcription de l'audio en spécifiant la langue
        result = self.model.transcribe(audio_file, language=self.language)

        # Ajouter un saut de ligne après chaque phrase pour avoir une meilleur lisibilité
        transcription_text = result["text"].replace(".", ".\n")

        # Diviser le texte en lignes
        lines = transcription_text.split("\n")

        # Enlever les lignes avec "Sous-titrage"
        filtered_lines = [line for line in lines if not line.strip().startswith("Sous-titrage")]

        # Réassembler le texte filtré
        filtered_text = "\n".join(filtered_lines).strip()

        # Écriture du texte filtré dans le fichier de sortie
        with open(self.output_file, "w", encoding="utf-8") as txt_file:
            txt_file.write(filtered_text)

        return result
```

FIGURE 2.2 – Transcription avec whisper turbo

```

import whisper
import datetime

from Transcription_Test import Transcribe

# Convertit un nombre de secondes en format timestamp SRT : HH:MM:SS,ms.
def seconds_to_timestamp(seconds):
    td = datetime.timedelta(seconds=seconds)
    total_seconds = int(td.total_seconds())
    hours, remainder = divmod(total_seconds, 3600)
    minutes, seconds = divmod(remainder, 60)
    milliseconds = int((td.total_seconds() - total_seconds) * 1000)
    return f"{hours:02}:{minutes:02}:{seconds:02},{milliseconds:03}"

# Charger le modèle Whisper
model = whisper.load_model("turbo")

transcriber = Transcribe(model=model, language="fr")

# Transcrire l'audio avec Whisper
result = transcriber.transcribe_audio(audio_file="ffmpeg-7.1.1/output_audio.wav")

# Afficher le dictionnaire pour voir ce que peut donner comme information le modèle
print(result)

# Créer un fichier SRT à partir des segments
with open("output_subtitles.srt", "w", encoding="utf-8") as srt_file:
    for i, segment in enumerate(result["segments"], start=1):
        if not segment["text"].startswith(" Sous-titrage"):
            start_timestamp = seconds_to_timestamp(segment["start"])
            end_timestamp = seconds_to_timestamp(segment["end"])
            text = segment["text"].strip()
            srt_file.write(f"{i}\n")
            srt_file.write(f"{start_timestamp} --> {end_timestamp}\n")
            srt_file.write(f"{text}\n\n")

print("Fichier SRT généré : output_subtitles.srt")

```

FIGURE 2.3 – Segmentation avec whisper turbo

```

Rolando, c'est mon père.
Bonjour, monsieur Levoux.
Ravi de vous rencontrer.
Laissez-moi seul avec lui une seconde.
J'ai hâte de parler, petit.
T'inquiète pas, ma chérie, il veut juste lui parler.
Je vais t'expliquer.
Ces 16 dernières années, la protection de ma fille a été sous ma responsabilité et sous ma responsabilité seulement.
Mais là, pour la première fois de ma vie, ça va devenir ta responsabilité.
Te plante pas.
Ou ta maman va pleurer chaque jour de sa vie en poussant son fils chéri dans un fauteuil roulant.

```

FIGURE 2.4 – Résultat de la transcription (audio en français)

```
1
00:00:00,000 --> 00:00:03,220
Rolando, c'est mon père.

2
00:00:04,419 --> 00:00:05,240
Bonjour, monsieur Levoux.

3
00:00:06,700 --> 00:00:07,480
Ravi de vous rencontrer.

4
00:00:10,300 --> 00:00:11,919
Laissez-moi seul avec lui une seconde.

5
00:00:14,640 --> 00:00:15,679
J'ai hâte de parler, petit.

6
00:00:24,679 --> 00:00:26,899
T'inquiète pas, ma chérie, il veut juste lui parler.

7
00:00:30,000 --> 00:00:36,600
Je vais t'expliquer.

8
00:00:37,219 --> 00:00:38,340
Ces 16 dernières années,

9
00:00:39,259 --> 00:00:41,679
la protection de ma fille a été sous ma responsabilité

10
00:00:41,679 --> 00:00:43,280
et sous ma responsabilité seulement.

11
00:00:44,219 --> 00:00:45,979
Mais là, pour la première fois de ma vie,

12
00:00:46,600 --> 00:00:48,200
ça va devenir ta responsabilité.

13
00:00:49,560 --> 00:00:50,200
Te plante pas.

14
00:00:51,100 --> 00:00:52,920
Ou ta maman va pleurer chaque jour de sa vie

15
00:00:52,920 --> 00:00:55,179
en poussant son fils chéri dans un fauteuil roulant.
```

FIGURE 2.5 – Résultat de la segmentation (format SRT)

3. Test de traduction en plusieurs langues à l'aide du modèle Helsinki-NLP disponible sur Hugging Face.

```
from transformers import pipeline
import nltk

# Télécharger le tokenizer NLTK
nltk.download('punkt_tab')

# Créer le pipeline de traduction du français vers l'anglais
translator = pipeline("translation_fr_to_en", model="Helsinki-NLP/opus-mt-fr-en")

# Lire la transcription
with open("output_transcription.txt", "r", encoding="utf-8") as f:
    transcription_text = f.read()

# Segmenter le texte en phrases car les timestamps ne sont pas forcément représentative d'une phrase entière
sentences = nltk.tokenize.sent_tokenize(transcription_text, language="french")

# Traduire chaque phrase
translated_sentences = []
for sentence in sentences:
    translated = translator(sentence, max_length=512)
    translated_sentences.append(translated[0]['translation_text'])

# Combiner les phrases traduites en un texte complet
translated_text = "\n".join(translated_sentences)

# Ecrire le résultat dans un fichier
with open("output_translated.txt", "w", encoding="utf-8") as f:
    f.write(translated_text)

print("Fichier Traduction généré : output_translated.txt")
```

FIGURE 2.6 – Traduction en anglais avec Helsinki-NLP

```
Rolando, he's my father.
Good morning, Mr. Levoux.
Nice to meet you.
Leave me alone with him for a second.
I can't wait to talk, kid.
Don't worry, honey, he just wants to talk to her.
Let me explain.
For the past 16 years, the protection of my daughter has been my responsibility and only my responsibility.
But now, for the first time in my life, it's gonna become your responsibility.
Don't screw up.
Or your mom's gonna cry every day of her life by pushing her darling son into a wheelchair.
```

FIGURE 2.7 – Résultat de la traduction

2.2 Olwen

Une piste pour ajouter les sous-titres à la vidéo serait d'utiliser moviepy, une bibliothèque de Python qui, à partir d'un fichier SRT, permettrait de superposer les sous-titres sur la vidéo.

Cette étape n'a pas encore été testée.

```
from moviepy.editor import VideoFileClip, CompositeVideoClip, TextClip
from moviepy.video.tools.subtitles import SubtitlesClip

# Charger la vidéo d'origine
video = VideoFileClip("video.mp4")

# Fonction qui génère un clip de texte pour chaque sous-titre
generator = lambda txt: TextClip(txt, font="Arial", fontsize=24, color="white")

# Charger le fichier de sous-titres au format SRT
subtitles = SubtitlesClip("subtitles.srt", generator)

# Superposer les sous-titres sur la vidéo
video_with_subs = CompositeVideoClip([video, subtitles.set_pos(('center', 'bottom'))])

# Sauvegarder la vidéo avec les sous-titres
video_with_subs.write_videofile("video_with_subs.mp4", fps=video.fps)
```

FIGURE 2.8 – Exemple de script avec Moviepy

On crée les parties de textes pour chaque sous-titre avec le format de notre choix via un generator et un fichier SRT comme montré dans la partie de Salim.

Ensuite on crée une nouvelle vidéo avec la vidéo originale combinée avec les sous-titres créés précédemment.

2.3 Samuel

Dans le cadre de notre projet de traduction et de sous-titrage automatique, le prétraitement vise à établir une base solide et fiable pour la génération de sous-titres d'une ressource majeure, telle que le sous-titrage d'une version originale (V.O.) d'un film. En assurant une préparation minutieuse des sous-titres dans la langue source, nous garantissons leur qualité, ce qui est essentiel pour obtenir une traduction fidèle et pertinente.

Nous avons pour ambition d'automatiser l'ensemble du processus, y compris cette phase de prétraitement. Celle-ci implique le nettoyage et l'optimisation des sous-titres générés automatiquement. En nous appuyant sur des outils tels que la bibliothèque MoviePy, nous intégrons ces sous-titres dans la vidéo de manière précise, assurant ainsi une synchronisation parfaite et une qualité optimale, le tout de façon entièrement automatisée.

```
import re

def preprocess_subtitles(input_file, output_file):
    # Depuis un fichier .srt, ne conserve que le "text" et le nettoye tres basiquement.
    processed_lines = []

    # Regex pour les infos du .srt (timestamps)
    time_pattern = re.compile(r'^\d{2}:\d{2}:\d{2},\d{3}\s-->\s\d{2}:\d{2}:\d{2},\d{3}$')

    with open(input_file, 'r', encoding='utf-8') as f:
        for line in f:
            line = line.strip()

            # Skip les lignes vides, les lignes index et le Regex (les timestamps)
            if (not line) or (line.isdigit()) or (time_pattern.match(line)):
                continue

            # (line == le texte "brute") a partir d'ici
            # On peut ensuite traiter le texte
            line = line.lower()          # tout en minuscule
            line = re.sub(r'^\w\s', '', line)  # enlever la ponctuation
            line = line.strip()          # enlever les espaces en trop

            if line:
                processed_lines.append(line)
```

FIGURE 2.9 – Fonction de nettoyage partie 1


```
                if line:
                    processed_lines.append(line)

    # Ecrit dans le fichier que l'on precise
    with open(output_file, 'w', encoding='utf-8') as out_f:
        for processed_line in processed_lines:
            out_f.write(processed_line + '\n')

if __name__ == "__main__":
    input_srt_path = "exemple.srt"
    output_txt_path = "exemple_clean.txt"

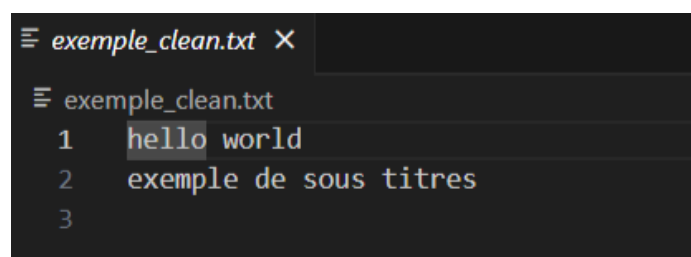
    preprocess_subtitles(input_srt_path, output_txt_path)
    print(f"\nSaved to: {output_txt_path}")
```

FIGURE 2.10 – Fonction de nettoyage partie 2



```
≡ exemple.srt ×
≡ exemple.srt
1 1
2 00:00:00,000 --> 00:00:02,000
3 Hello world !
4
5 2
6 00:00:02,500 --> 00:00:04,000
7 Exemple de sous titres.
```

FIGURE 2.11 – Exemple fichier SRT



```
≡ exemple_clean.txt ×
≡ exemple_clean.txt
1 hello world
2 exemple de sous titres
3
```

FIGURE 2.12 – Transcription nettoyée

Chapitre 3

Planification pour le prochain Sprint

Au bout de ce Sprint, nous avons réfléchi à deux architectures pour mettre en place notre projet :

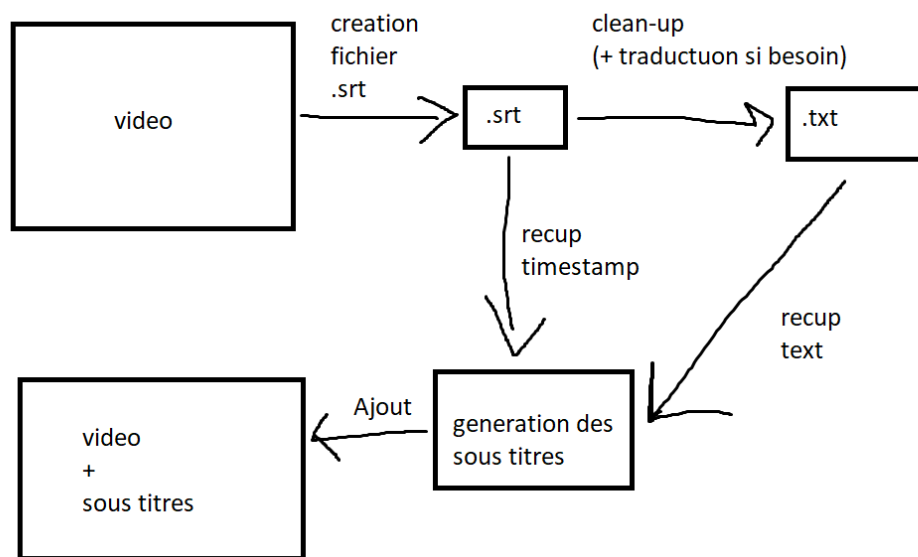


FIGURE 3.1 – Première Architecture (Sam)

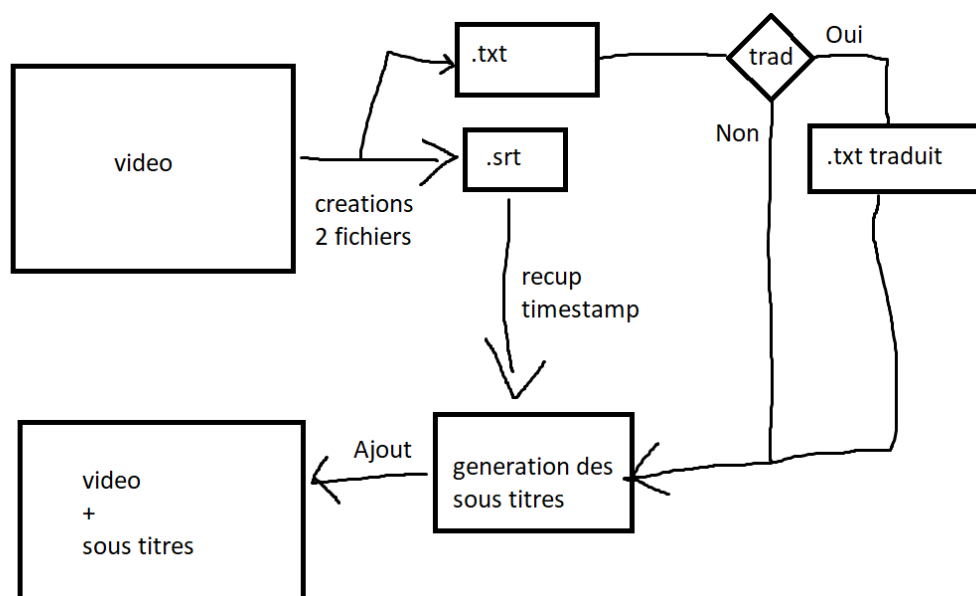


FIGURE 3.2 – Deuxième Architecture (Salim)

Dans cette deuxième architecture, nous pouvons générer simultanément les fichiers SRT et TXT, évitant ainsi l'étape intermédiaire de nettoyage. Nous concentrerons donc nos efforts sur cette approche pour optimiser notre pipeline de traitement.

Pour le prochain sprint, notre objectif est de mettre en place un cycle complet de sous-titrage et de traduction automatique d'une vidéo, tout en renforçant la robustesse de notre système face à des pistes audio fortement perturbées par du bruit parasite.

Nous procéderons à une série de tests sur diverses vidéos présentant des niveaux de complexité audio variés.

Chapitre 4

Bilan

Au cours de ce premier sprint, nous avons réalisé avec succès la transcription, la segmentation et la traduction d'une vidéo dépourvue de contraintes articulatoires et de bruit parasite, garantissant ainsi une transcription fluide et précise.

Nous avons également exploré différentes approches pour préparer la suite du cycle de sous-titrage et de traduction automatique. À ce stade, nous disposons d'une base technique solide, qui nous permet de :

- Extraire l'audio d'une vidéo et le convertir en texte, en identifiant précisément les débuts et fins de chaque segment.
- Traduire ce texte dans une langue cible.
- Intégrer ce texte à la vidéo pour générer une version sous-titrée.

Il reste toutefois à interconnecter ces différents modules pour constituer un système complet et automatisé, ce qui constituera notre principal objectif lors du prochain sprint.