UQAC

Université du Québec
à Chicoutimi

Sprint 2

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

11 avril 2025

Table des matières

Table des figures

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 rappel, lors du premier sprint, nous avions posé les bases d'un système de sous-titrage et de traduction automatique de vidéos. Concrètement, nous avons débuté par l'extraction de la piste audio à l'aide de ffmpeg, puis testé la transcription et la segmentation audio en utilisant le modèle Whisper Turbo d'OpenAI. Ce processus nous a permis d'obtenir un fichier texte découpé en segments synchronisés.

Nous avons ensuite exploré la traduction automatique en plusieurs langues, en nous appuyant sur le modèle Helsinki-NLP, disponible sur la plateforme Hugging Face, afin de valider la faisabilité de notre approche multilingue.

Parallèlement, nous avons commencé à étudier l'intégration des sous-titres dans la vidéo à l'aide de la bibliothèque MoviePy, tout en développant des fonctions de prétraitement et de nettoyage automatique pour garantir une meilleure qualité des sous-titres avant leur insertion.

En fin de sprint, deux architectures possibles ont été proposées.

Pour ce deuxième sprint, nous visons l'achèvement d'une première version pleinement fonctionnelle du système. Celle-ci devra intégrer toutes les fonctionnalités de base : détection et extraction de la piste audio, transcription du contenu parlé, traduction vers une ou plusieurs langues, génération des sous-titres avec horodatage, et export final dans un format standard. Des tests sur plusieurs vidéos de référence permettront également d'évaluer la qualité des résultats, en termes de précision de la transcription, fidélité de la traduction et synchronisation des sous-titres.

Chapitre 2

Travail Réalisé

2.1 Salim

L'un des principaux défis de ce sprint a été de mettre en place un cycle complet de sous-titrage et de traduction automatique d'une vidéo, depuis l'extraction audio jusqu'à la génération finale d'un contenu enrichi.

Dans ma partie, je me suis concentré sur l'ajout des sous-titres à une vidéo. Pour cela, j'ai intégré le fichier SRT généré par le modèle Whisper dans un algorithme utilisant la bibliothèque MoviePy, afin de recréer la vidéo originale avec les sous-titres incrustés.

Étant donné que l'utilisation de MoviePy n'avait pas été implémentée lors du premier sprint, j'ai dû adapter et corriger le code en y intégrant la variable d'environnement 'IMAGEMAGICK_BINARY'. La bibliothèque MoviePy repose sur ImageMagick pour le rendu des textes et la création d'animations, notamment lors de l'incrustation de sous-titres.

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

os.environ["IMAGEMAGICK_BINARY"] = "/opt/homebrew/bin/convert"


# Définir une fonction qui génère un clip de texte pour chaque sous-titre
def subtitle_generator(txt):
    return TextClip(txt,
                    font="Arial",
                    fontsize=24,
                    color="white")

# Charger la vidéo d'origine
video = VideoFileClip("ffmpeg-7.1.1/Test_video.mp4")

# Charger le fichier de sous-titres au format SRT et créer le clip de sous-titres
subtitles = SubtitlesClip("output_subtitles.srt", subtitle_generator)

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

# Sauvegarder la vidéo résultante
video_with_subs.write_videofile("video_with_subs.mp4", fps=video.fps)
```

FIGURE 2.1 – Création de la vidéo finale avec MoviePy

L'utilisation des Timestamps générés automatiquement par Whisper a révélé plusieurs problèmes de segmentation dans la vidéo finale :

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

FIGURE 2.2 – Timestamp avec whisper

FIGURE 2.3 – Vidéo avec les sous-titres Whisper

Dans cette scène, aucun dialogue n'est prononcé et un long silence s'installe. Pourtant, le sous-titre reste affiché à l'écran pendant 6 secondes, à partir de la 30ème seconde de la vidéo, alors que la phrase correspondante ne débute réellement qu'autour de la 36ème seconde. Ce décalage pose un problème majeur de synchronisation.

Pour résoudre cette difficulté, j'ai décidé d'adopter une approche différente en utilisant un autre modèle de segmentation : Vosk.

Vosk est un outil de reconnaissance vocale open source capable de convertir la parole en texte. Il fonctionne hors ligne, ce qui le rend particulièrement adapté aux environnements dépourvus de connexion Internet.

Le principal inconvénient de Vosk est que chaque modèle est spécifique à une langue unique. Dans notre cas, la vidéo test étant en français, nous avons utilisé le modèle "vosk-model-fr-0.22", qui est à ce jour le plus complet pour la langue française.

La segmentation obtenue à l'aide de ce modèle est illustrée dans l'image ci-dessous :

output_subtitles_vosk.srt

```
1
00:00:01,560 --> 00:00:13,299
rolando c'est mon père

2
00:00:04,379 --> 00:00:15,309
bonjour monsieur le vous

3
00:00:06,719 --> 00:00:17,559
ravi de vous rencontrer

4
00:00:10,349 --> 00:00:21,939
laissez-moi seul avec lui une seconde

5
00:00:14,910 --> 00:00:25,779
j'ai à te parler petit

6
00:00:24,989 --> 00:00:37,000
t'inquiète pas ma chérie il veut juste lui parler

7
00:00:35,880 --> 00:00:46,689
je vais t'expliquer

8
00:00:37,200 --> 00:00:48,460
ces seize dernières années

9
00:00:39,210 --> 00:00:53,590
la protection de ma fille a été sous ma responsabilité et sous ma responsabilité seulement

10
00:00:44,189 --> 00:00:58,329
mais là pour la première fois de ma vie ça va devenir ta responsabilité

11
00:00:49,560 --> 00:00:60,280
te plante pas

12
00:00:51,119 --> 00:00:65,259
ou ta maman va pleurer chaque jour de sa vie en poussant son fils chéri dans un fauteuil roulant
```

FIGURE 2.4 – Segmentation avec Vosk

Cette fois-ci, les Timestamps semblent bien alignés avec l'audio de la vidéo originale, ce qui marque une nette amélioration. Cependant, plusieurs problèmes subsistent : aucune ponctuation n'est présente dans le texte, et certains sous-titres risquent de ne pas s'afficher entièrement à l'écran.

Pour remédier à ces défauts, un nouvel algorithme a été développé (voir 4.2).

corrected.srt

```
1
00:00:01,560 --> 00:00:04,299
Rolando c'est mon père.

2
00:00:04,379 --> 00:00:06,309
Bonjour monsieur le vous.

3
00:00:06,719 --> 00:00:08,559
Ravi de vous rencontrer.

4
00:00:10,349 --> 00:00:12,939
Laissez-moi seul avec lui une seconde.

5
00:00:14,910 --> 00:00:16,779
J'ai à te parler petit.

6
00:00:24,989 --> 00:00:28,000
T'inquiète pas ma chérie il veut juste lui parler.

7
00:00:35,880 --> 00:00:37,689
Je vais t'expliquer.

8
00:00:37,200 --> 00:00:39,429
Ces seize dernières années.

9
00:00:39,210 --> 00:00:41,900
La protection de ma fille a été sous ma responsabilité

10
00:00:41,900 --> 00:00:44,590
et sous ma responsabilité seulement.

11
00:00:44,189 --> 00:00:46,759
Mais là pour la première fois de ma vie ça va devenir

12
00:00:46,759 --> 00:00:49,329
ta responsabilité.

13
00:00:49,560 --> 00:00:51,280
Te plante pas.

14
00:00:51,119 --> 00:00:53,689
Ou ta maman va pleurer chaque jour de sa vie en

15
00:00:53,689 --> 00:00:56,259
poussant son fils chéri dans un fauteuil roulant.
```

FIGURE 2.5 – Segmentation corrigé

Ce dernier permet non seulement de corriger les erreurs de segmentation, mais aussi de comparer les transcriptions issues d'autres modèles. Il devient alors possible, si nécessaire, de fusionner les Timestamps générés par Vosk avec une transcription obtenue via un autre modèle, comme Whisper, afin d'obtenir un résultat plus fiable et lisible.

2.2 Olwen et Sam

Dans le cadre de l'ajout d'une fonctionnalité de traduction et de nettoyage du texte, nous avons réutilisé le code développé lors du sprint précédent. Ce dernier a été intégré au module de post-traitement des sous-titres, notamment via l'ajout d'un paramètre lang dans la classe "SRTProcessor". Ce paramètre permet de sélectionner dynamiquement le modèle de traduction Helsinki-NLP à utiliser pour passer du français vers la langue cible.

L'utilisateur peut ainsi spécifier la langue de sortie des sous-titres ; à défaut, ceux-ci resteront en français.

Il est également important de noter que certains lecteurs vidéo rencontrent des difficultés à afficher correctement les caractères accentués, ce qui peut impacter la lisibilité du résultat final.

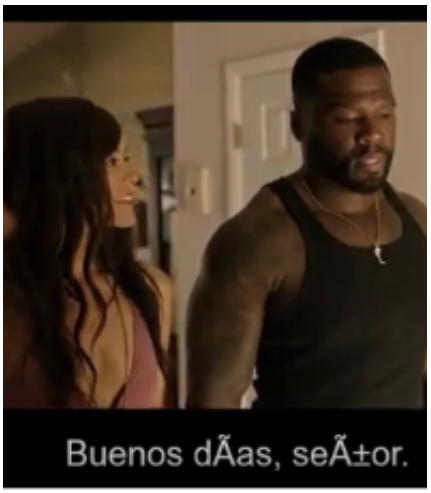


FIGURE 2.6 – Problème avec les accents

Pour y remédier, nous avons ajouté une nouvelle fonction chargée de supprimer les caractères accentués. Celle-ci est désormais appelée automatiquement dans la classe "SRTProcessor", afin de garantir une meilleure compatibilité avec les lecteurs vidéo.

```
1 usage  new *
def remove_accents(text: str) -> str:
    text = re.sub( pattern: r'[àáâã]', repl: 'a', text)
    text = re.sub( pattern: r'[ÀÁÂÃ]', repl: 'A', text)

    text = re.sub( pattern: r'[éèêë]', repl: 'e', text)
    text = re.sub( pattern: r'[ÉÈÊË]', repl: 'E', text)

    text = re.sub( pattern: r'[íìîï]', repl: 'i', text)
    text = re.sub( pattern: r'[ÍÌÎÏ]', repl: 'I', text)

    text = re.sub( pattern: r'[óòõö]', repl: 'o', text)
    text = re.sub( pattern: r'[ÓÒÕÖ]', repl: 'O', text)

    text = re.sub( pattern: r'[úùûü]', repl: 'u', text)
    text = re.sub( pattern: r'[ÚÙÛÜ]', repl: 'U', text)

    text = re.sub( pattern: r'[ç]', repl: 'c', text)
    text = re.sub( pattern: r'[Ç]', repl: 'C', text)
    text = re.sub( pattern: r'[ñ]', repl: 'n', text)
    text = re.sub( pattern: r'[Ñ]', repl: 'N', text)

    return text
```

FIGURE 2.7 – Fonction pour enlever les accents

Les fonctions existantes ont dû être légèrement ajustées en raison de l'évolution de l'architecture de l'application. À ce stade du développement, l'ensemble des composantes du système n'était pas encore intégré, ce qui ne nous avait pas permis d'anticiper pleinement les contraintes liées à l'architecture définie à la fin du rapport du sprint 1.

Initialement, nous avions envisagé une approche reposant sur deux fichiers distincts : un fichier .srt pour les sous-titres et un fichier .txt destiné à faciliter un formatage plus fin du texte. Cette méthode s'est toutefois révélée inapplicable, les sous-titres étant exclusivement extraits du fichier .srt. En conséquence, nous avons adapté nos fonctions pour qu'elles agissent directement sur ce fichier unique.

Par ailleurs, une piste d'amélioration a été explorée afin d'augmenter légèrement la durée d'affichage des sous-titres dans le but d'en améliorer la lisibilité. Cette problématique, qui avait déjà motivé notre adoption de Vosk, méritait selon nous un traitement plus fin.

Dans un premier temps, nous avons envisagé une solution simple : prolonger la fin de chaque sous-titre d'une demi-seconde à une seconde. Toutefois, cette méthode soulevait une incertitude importante : nous ignorions comment les lecteurs vidéo allaient réagir face à des sous-titres aux intervalles temporels partiellement superposés.

Pour explorer cette question, nous avons scindé notre fonction d'analyse des Timestamps en deux sous-fonctions distinctes : l'une responsable de déterminer les points de départ, l'autre de gérer les points de fin. Cette dernière introduisait volontairement un chevauchement en prolongeant artificiellement certains sous-titres jusqu'à dix secondes, afin d'observer les comportements réels des lecteurs vidéo.

Les tests menés avec cette version expérimentale ont révélé un fonctionnement séquentiel des lecteurs : chaque bloc du fichier .srt (composé d'un identifiant, d'un intervalle temporel et du texte) est affiché de manière exclusive. Ainsi, un sous-titre X+1 n'apparaît à l'écran qu'une fois le sous-titre X retiré. En cas de chevauchement, les lecteurs semblent empiler les sous-titres dans une file d'attente : un seul est affiché à la fois, les suivants attendent leur tour. Ce comportement implique qu'une extension arbitraire des durées, sans ajustement coordonné de l'ensemble, risquerait de désynchroniser les dialogues rapides et de nuire à la compréhension globale.

Face à ce constat, nous avons adopté une stratégie plus nuancée. Plutôt que d'ajouter systématiquement une seconde à la fin de chaque sous-titre, notre algorithme choisit dynamiquement l'option la plus pertinente : soit prolonger le sous-titre d'une seconde, soit le couper juste avant le début du suivant, en fonction de l'option la plus proche sur la ligne temporelle. Ce compromis permet d'optimiser la lisibilité tout en préservant une synchronisation fidèle avec le rythme de la vidéo.

Cette stratégie, bien que très intéressante, était assez complexe à introduire dans le code. En raison des contraintes temporelles sous lesquelles nous avons opéré lors de ce deuxième sprint, nous avons choisi de laisser l'application telle quelle pour le moment afin de nous concentrer sur d'autres tâches plus critiques.

Nous avons également réalisé un test de sous-titrage sur une vidéo contenant de la musique et des bruitages, éléments susceptibles de perturber l'analyse audio. Pour ce test, nous avons choisi un extrait de type trailer de film, en raison de sa richesse sonore et de la diversité des éléments audio qu'il contient.

Le résultat obtenu avec Vosk était encourageant, mais insuffisamment précis, notamment en raison de la musique de fond et des accents des acteurs.

Pour améliorer la transcription dans ce type de contexte, une étape de prétraitement audio visant à isoler les dialogues serait utile. Une autre option serait d'utiliser la comparaison avec un second modèle (fonction déjà présente dans notre code 4.2), comme Whisper, afin de corriger automatiquement les erreurs en s'appuyant sur les forces de chaque transcription.

Chapitre 3

Bilan

Au cours de ce deuxième sprint, nous avons consolidé et structuré l'ensemble des étapes du pipeline de sous-titrage automatique, en allant de la transcription initiale jusqu'à l'intégration finale des sous-titres dans la vidéo.

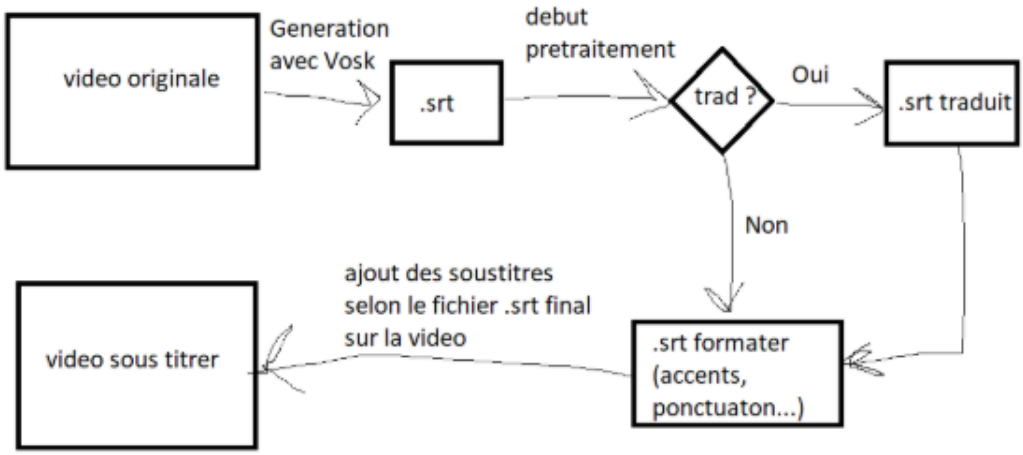


FIGURE 3.1 – Architecture finale

Le schéma mis en place nous a permis de clarifier les enchaînements logiques entre les différentes composantes du système, et de poser les fondations d'un traitement modulaire et adaptable. À ce stade, nous disposons d'un processus fonctionnel capable de :

— Générer automatiquement un fichier `.srt` à partir de la piste audio d'une vidéo, en utilisant le modèle Vosk.

— Traiter ce fichier en vue d'une traduction optionnelle, grâce à l'intégration de modèles Helsinki-NLP.

— Nettoyer et formater le fichier `.srt` pour garantir la lisibilité des sous-titres (ponctuation, accents, durée d'affichage...).

— Ajouter les sous-titres à la vidéo finale à l'aide d'outils de traitement vidéo.

Chapitre 4

Annexe

4.1 Vosk

```python
import datetime
import json
import os
from vosk import Model, KaldiRecognizer, SetLogLevel
import wave
import sys
import subprocess

class VoskTranscribe:
    def __init__(self, model_path: str, language: str, output_file: str = "output_transcription.txt"):
        self.language = language
        self.output_file = output_file

        # Désactiver les logs non nécessaires
        SetLogLevel(-1)

        # Charger le modèle Vosk
        try:
            self.model = Model(model_path)
        except Exception as e:
            print(f"Erreur lors du chargement du modèle: {e}")
            print(f"Assurez-vous d'avoir téléchargé le modèle pour la langue {language} depuis https://alphacephei.com/vosk/models")
            sys.exit(1)

    def ensure_wav_format(self, audio_file: str) -> str:
        # Vérifier si le fichier est déjà un WAV
        if audio_file.lower().endswith('.wav'):
            return audio_file

        # Sinon, convertir en WAV temporaire
        output_wav = os.path.splitext(audio_file)[0] + "_temp.wav"
        try:
            subprocess.run(
                ["ffmpeg", "-i", audio_file, "-ar", "16000", "-ac", "1", output_wav],
                check=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE
            )
            return output_wav
        except subprocess.CalledProcessError as e:
            print(f"Erreur lors de la conversion audio: {e}")
            sys.exit(1)
        except FileNotFoundError:
            print("FFmpeg n'est pas installé. Veuillez l'installer pour la conversion audio.")
            sys.exit(1)
```

FIGURE 4.1 – Implémentation de Vosk-Partie 1

```python
def transcribe_audio(self, audio_file: str) -> dict:
    # S'assurer que le fichier est au format WAV
    wav_file = self.ensure_wav_format(audio_file)

    # Ouvrir le fichier audio
    wf = wave.open(wav_file, "rb")

    # Préparer le reconnaisseur avec le modèle
    rec = KaldiRecognizer(self.model, wf.getframerate())
    rec.SetWords(True)  # Pour obtenir les timestamps par mot

    # Préparer les structures pour stocker les résultats
    results = []
    segments = []
    full_text = ""

    # Traiter l'audio par morceaux
    while True:
        data = wf.readframes(4000)  # Lire par blocs
        if len(data) == 0:
            break

        if rec.AcceptWaveform(data):
            result = json.loads(rec.Result())
            results.append(result)

            if "result" in result:
                segment_text = result.get("text", "")
                if segment_text:
                    words = result.get("result", [])
                    if words:
                        start_time = words[0].get("start", 0)
                        end_time = words[-1].get("end", start_time)

                        segment = {
                            "start": start_time,
                            "end": end_time,
                            "text": segment_text
                        }
                        segments.append(segment)
                        full_text += segment_text + " "

    # Récupérer le dernier résultat
    final_result = json.loads(rec.FinalResult())
    results.append(final_result)
```

FIGURE 4.2 – Implémentation de Vosk-Partie 2

```
if "result" in final_result:
    segment_text = final_result.get("text", "")
    if segment_text:
        words = final_result.get("result", [])
        if words:
            start_time = words[0].get("start", 0)
            end_time = words[-1].get("end", start_time)

            segment = {
                "start": start_time,
                "end": end_time,
                "text": segment_text
            }
            segments.append(segment)
            full_text += segment_text + " "

# Formater le texte pour une meilleure lisibilité
formatted_text = full_text.replace(". ", ".\n").strip()

# Diviser le texte en lignes
lines = formatted_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)

# Nettoyer le fichier temporaire si nécessaire
if wav_file != audio_file:
    os.remove(wav_file)

# Créer un résultat similaire à celui de Whisper pour la compatibilité
return {
    "text": filtered_text,
    "segments": segments,
    "language": self.language
}
```

FIGURE 4.3 – Implémentation de Vosk-Partie 3

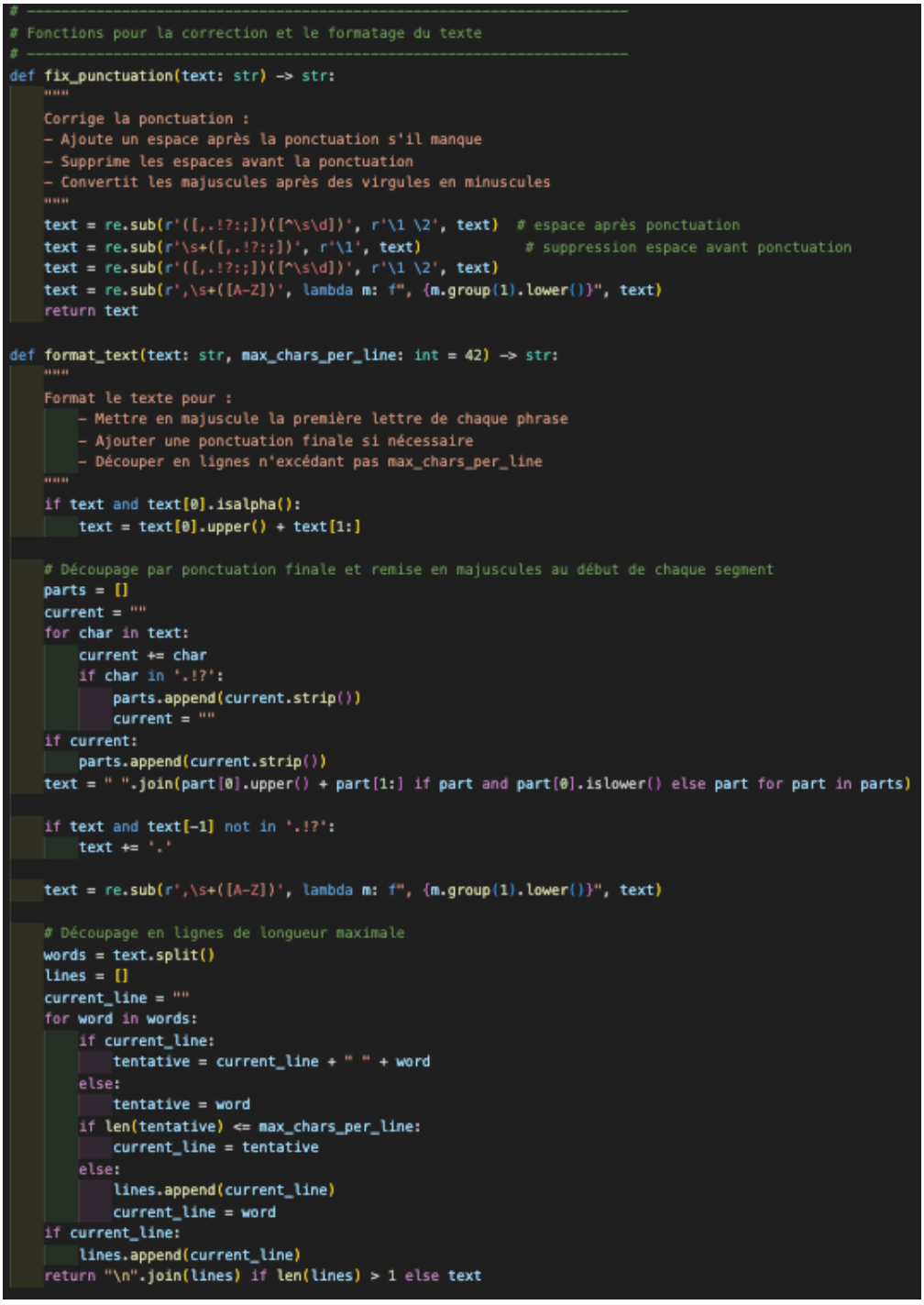
```python
# Convertit un nombre de secondes en format timestamp SRT : HH:MM:SS,ms.
def seconds_to_timestamp(seconds):
    td = datetime.timedelta(seconds=seconds)
    total_seconds = int(td.total_seconds())
    hours, remainder = divmod(total_seconds, 3600)
    minutes, seconds = divmod(remainder, 60)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

def seconds_to_timestamp2(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+1:02},{milliseconds:03}"


def main():
    model_path = "vosk-model-fr-0.22"
    language = "fr"

    if not os.path.exists(model_path):
        print(f"Le modèle Vosk pour {language} n'est pas trouvé.")
        print(f"Veuillez télécharger le modèle depuis https://alphacephei.com/vosk/models")
        print(f"et l'extraire dans un dossier nommé '{model_path}'")
        sys.exit(1)

    # Initialiser le transcripteur Vosk
    transcriber = VoskTranscribe(model_path=model_path, language=language)

    # Chemin du fichier audio à transcrire
    audio_file = "ffmpeg-7.1.1/output_Modop.wav"  # À adapter selon votre fichier

    # Transcrire l'audio avec Vosk
    print(f"Transcription en cours de {audio_file}...")
    result = transcriber.transcribe_audio(audio_file=audio_file)

    print("Transcription terminée!")

    # Créer un fichier SRT à partir des segments
    with open("output_subtitles_vosk.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_timestamp2(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_vosk.srt")


if __name__ == "__main__":
    main()
```

FIGURE 4.4 – Implémentation de Vosk-Partie 4

4.2 Correction des sous-titres

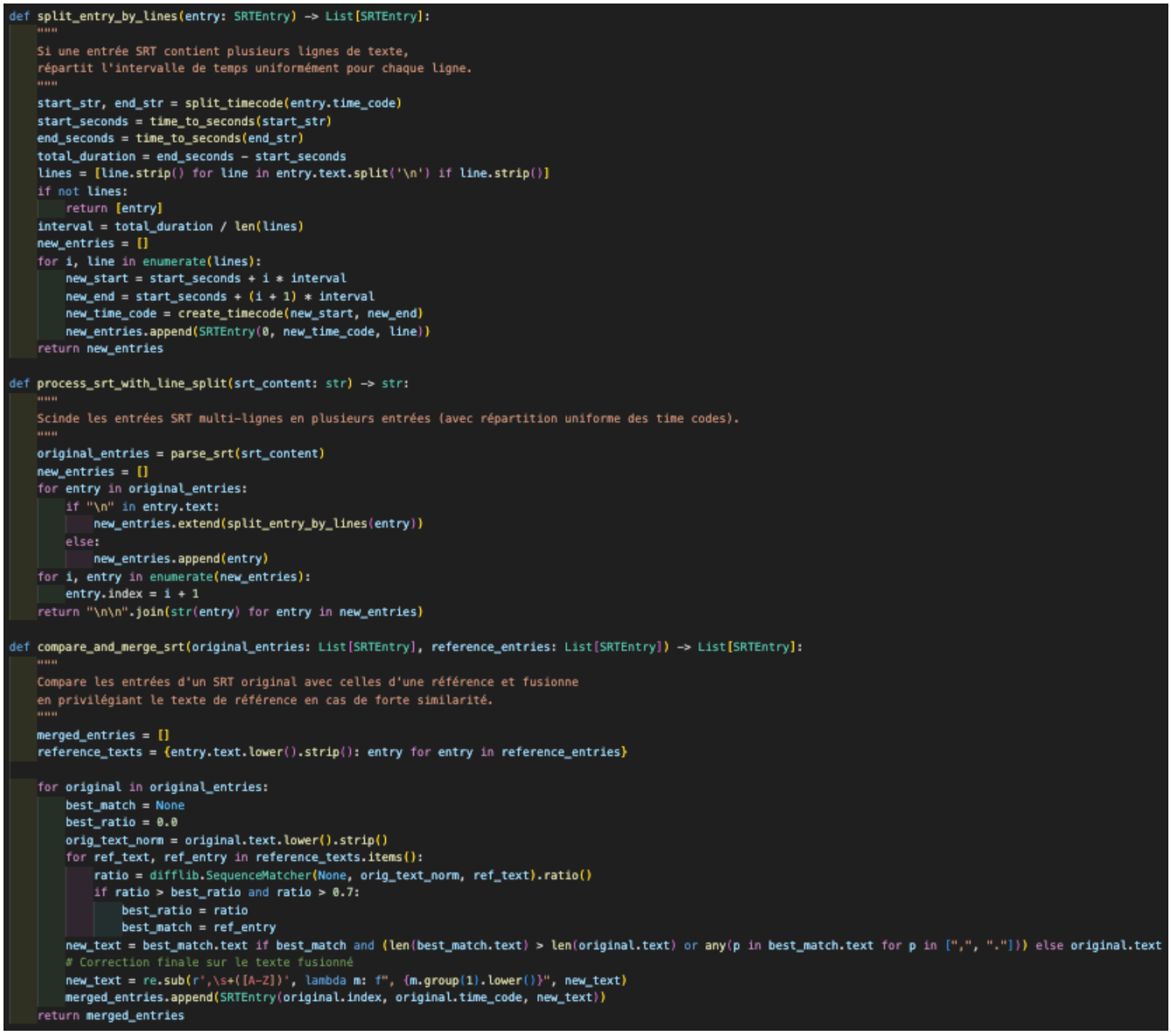
```python
import re
from typing import List, Optional
import difflib
from transformers import pipeline

# ------------------------------------------------------------------------
# Classe représentant une entrée SRT
# ------------------------------------------------------------------------
class SRTEntry:
    def __init__(self, index: int, time_code: str, text: str):
        self.index = index
        self.time_code = time_code
        self.text = text

    def __str__(self):
        # Retourne l'entrée au format SRT (une ligne vide à la fin)
        return f"{self.index}\n{self.time_code}\n{self.text}\n\n"

# ------------------------------------------------------------------------
# Fonctions utilitaires pour le format SRT
# ------------------------------------------------------------------------
def parse_srt(content: str) -> List[SRTEntry]:
    """
    Parse le contenu SRT en entrées individuelles.
    Nettoie le texte, sépare en blocs et crée des SRTEntry.
    """
    content = content.strip().replace('\r\n', '\n')
    blocks = re.split(r'\n\n+', content)
    entries = []
    for block in blocks:
        if not block.strip():
            continue
        lines = block.split('\n')
        if len(lines) < 3:
            continue
        try:
            index = int(lines[0])
            time_code = lines[1]
            text = '\n'.join(lines[2:]).strip()
            if text:
                entries.append(SRTEntry(index, time_code, text))
        except ValueError:
            continue
    return entries

def time_to_seconds(time_str: str) -> float:
    """Convertit un time code 'HH:MM:SS,mmm' en secondes."""
    hours, minutes, rest = time_str.split(':')
    seconds, millis = rest.split(',')
    return int(hours) * 3600 + int(minutes) * 60 + int(seconds) + int(millis) / 1000

def seconds_to_timecode(seconds: float) -> str:
    """Convertit des secondes en time code 'HH:MM:SS,mmm'."""
    hours = int(seconds // 3600)
    minutes = int((seconds % 3600) // 60)
    secs = int(seconds % 60)
    millis = int(round((seconds - int(seconds)) * 1000))
    return f"{hours:02d}:{minutes:02d}:{secs:02d},{millis:03d}"

def split_timecode(time_code: str) -> tuple:
    """Sépare le time code complet en début et fin."""
    return time_code.split(" --> ")

def create_timecode(start: float, end: float) -> str:
    """Crée un time code à partir des valeurs de début et de fin en secondes."""
    return f"{seconds_to_timecode(start)} --> {seconds_to_timecode(end)}"
```

FIGURE 4.5 – Implémentation de la correction de sous-titres Vosk-Partie 1



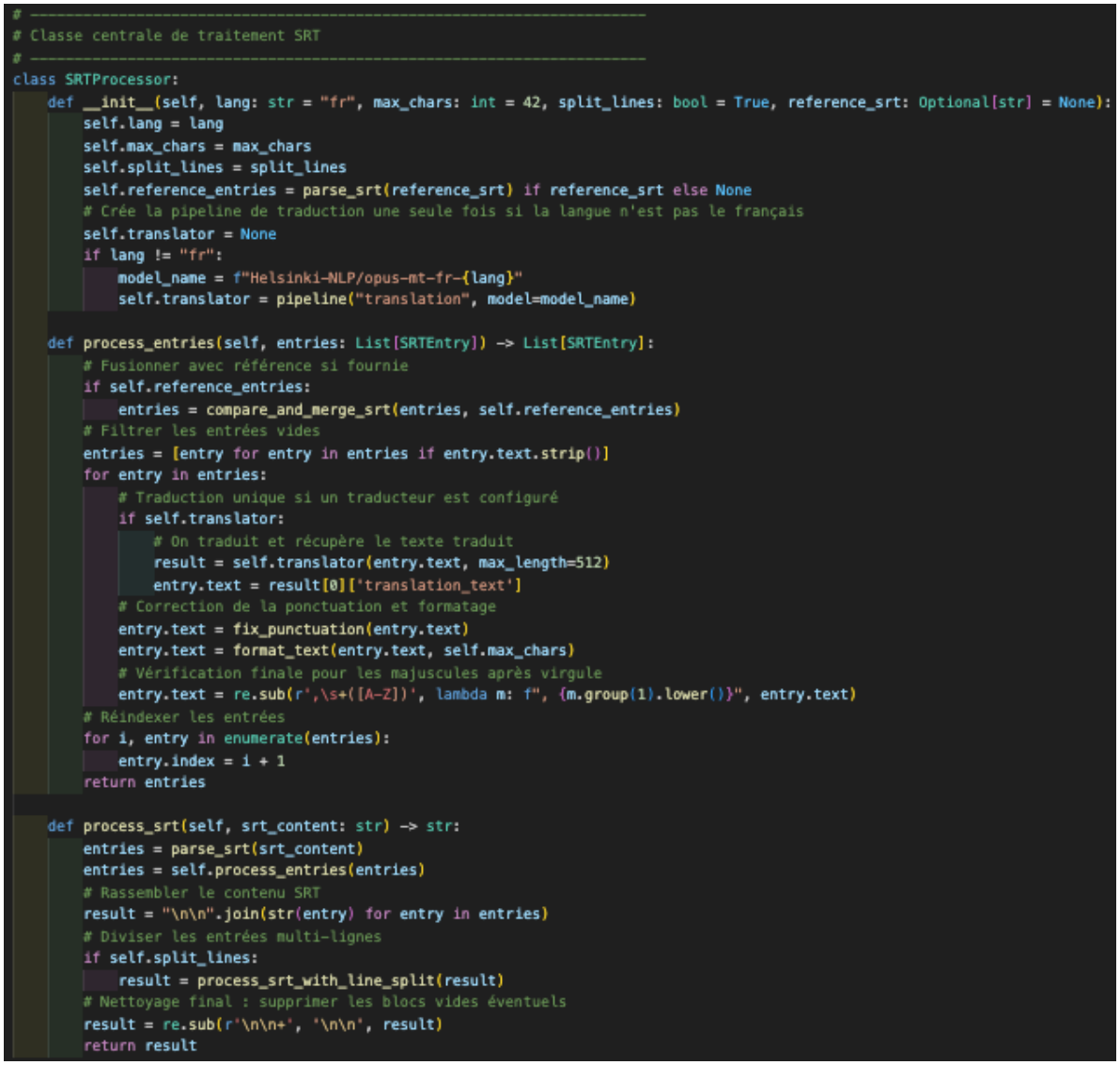
```python
# ------------------------------------------------------------------------
# Fonctions pour la correction et le formatage du texte
# ------------------------------------------------------------------------
def fix_punctuation(text: str) -> str:
    """
    Corrige la ponctuation :
    - Ajoute un espace après la ponctuation s'il manque
    - Supprime les espaces avant la ponctuation
    - Convertit les majuscules après des virgules en minuscules
    """
    text = re.sub(r'([,.!?:;])([^\s\d])', r'\1 \2', text)  # espace après ponctuation
    text = re.sub(r'\s+([,.!?:;])', r'\1', text)            # suppression espace avant ponctuation
    text = re.sub(r'([,.!?:;])([^\s\d])', r'\1 \2', text)
    text = re.sub(r',\s+([A-Z])', lambda m: f", {m.group(1).lower()}", text)
    return text

def format_text(text: str, max_chars_per_line: int = 42) -> str:
    """
    Format le texte pour :
        - Mettre en majuscule la première lettre de chaque phrase
        - Ajouter une ponctuation finale si nécessaire
        - Découper en lignes n'excédant pas max_chars_per_line
    """
    if text and text[0].isalpha():
        text = text[0].upper() + text[1:]

    # Découpage par ponctuation finale et remise en majuscules au début de chaque segment
    parts = []
    current = ""
    for char in text:
        current += char
        if char in '.!?':
            parts.append(current.strip())
            current = ""
    if current:
        parts.append(current.strip())
    text = " ".join(part[0].upper() + part[1:] if part and part[0].islower() else part for part in parts)

    if text and text[-1] not in '.!?':
        text += '.'

    text = re.sub(r',\s+([A-Z])', lambda m: f", {m.group(1).lower()}", text)

    # Découpage en lignes de longueur maximale
    words = text.split()
    lines = []
    current_line = ""
    for word in words:
        if current_line:
            tentative = current_line + " " + word
        else:
            tentative = word
        if len(tentative) <= max_chars_per_line:
            current_line = tentative
        else:
            lines.append(current_line)
            current_line = word
    if current_line:
        lines.append(current_line)
    return "\n".join(lines) if len(lines) > 1 else text
```

FIGURE 4.6 – Implémentation de la correction de sous-titres Vosk-Partie 2



```python
def split_entry_by_lines(entry: SRTEntry) -> List[SRTEntry]:
    """
    Si une entrée SRT contient plusieurs lignes de texte,
    répartit l'intervalle de temps uniformément pour chaque ligne.
    """
    start_str, end_str = split_timecode(entry.time_code)
    start_seconds = time_to_seconds(start_str)
    end_seconds = time_to_seconds(end_str)
    total_duration = end_seconds - start_seconds
    lines = [line.strip() for line in entry.text.split('\n') if line.strip()]
    if not lines:
        return [entry]
    interval = total_duration / len(lines)
    new_entries = []
    for i, line in enumerate(lines):
        new_start = start_seconds + i * interval
        new_end = start_seconds + (i + 1) * interval
        new_time_code = create_timecode(new_start, new_end)
        new_entries.append(SRTEntry(0, new_time_code, line))
    return new_entries

def process_srt_with_line_split(srt_content: str) -> str:
    """
    Scinde les entrées SRT multi-lignes en plusieurs entrées (avec répartition uniforme des time codes).
    """
    original_entries = parse_srt(srt_content)
    new_entries = []
    for entry in original_entries:
        if "\n" in entry.text:
            new_entries.extend(split_entry_by_lines(entry))
        else:
            new_entries.append(entry)
    for i, entry in enumerate(new_entries):
        entry.index = i + 1
    return "\n\n".join(str(entry) for entry in new_entries)

def compare_and_merge_srt(original_entries: List[SRTEntry], reference_entries: List[SRTEntry]) -> List[SRTEntry]:
    """
    Compare les entrées d'un SRT original avec celles d'une référence et fusionne
    en privilégiant le texte de référence en cas de forte similarité.
    """
    merged_entries = []
    reference_texts = {entry.text.lower().strip(): entry for entry in reference_entries}

    for original in original_entries:
        best_match = None
        best_ratio = 0.0
        orig_text_norm = original.text.lower().strip()
        for ref_text, ref_entry in reference_texts.items():
            ratio = difflib.SequenceMatcher(None, orig_text_norm, ref_text).ratio()
            if ratio > best_ratio and ratio > 0.7:
                best_ratio = ratio
                best_match = ref_entry
        new_text = best_match.text if best_match and (len(best_match.text) > len(original.text) or any(p in best_match.text for p in [",", "."])) else original.text
        # Correction finale sur le texte fusionné
        new_text = re.sub(r',\s+([A-Z])', lambda m: f", {m.group(1).lower()}", new_text)
        merged_entries.append(SRTEntry(original.index, original.time_code, new_text))
    return merged_entries
```

FIGURE 4.7 – Implémentation de la correction de sous-titres Vosk-Partie 3



```python
# ------------------------------------------------------------------------
# Classe centrale de traitement SRT
# ------------------------------------------------------------------------
class SRTProcessor:
    def __init__(self, lang: str = "fr", max_chars: int = 42, split_lines: bool = True, reference_srt: Optional[str] = None):
        self.lang = lang
        self.max_chars = max_chars
        self.split_lines = split_lines
        self.reference_entries = parse_srt(reference_srt) if reference_srt else None
        # Crée la pipeline de traduction une seule fois si la langue n'est pas le français
        self.translator = None
        if lang != "fr":
            model_name = f"Helsinki-NLP/opus-mt-fr-{lang}"
            self.translator = pipeline("translation", model=model_name)

    def process_entries(self, entries: List[SRTEntry]) -> List[SRTEntry]:
        # Fusionner avec référence si fournie
        if self.reference_entries:
            entries = compare_and_merge_srt(entries, self.reference_entries)
        # Filtrer les entrées vides
        entries = [entry for entry in entries if entry.text.strip()]
        for entry in entries:
            # Traduction unique si un traducteur est configuré
            if self.translator:
                # On traduit et récupère le texte traduit
                result = self.translator(entry.text, max_length=512)
                entry.text = result[0]['translation_text']
            # Correction de la ponctuation et formatage
            entry.text = fix_punctuation(entry.text)
            entry.text = format_text(entry.text, self.max_chars)
            # Vérification finale pour les majuscules après virgule
            entry.text = re.sub(r',\s+([A-Z])', lambda m: f", {m.group(1).lower()}", entry.text)
        # Réindexer les entrées
        for i, entry in enumerate(entries):
            entry.index = i + 1
        return entries

    def process_srt(self, srt_content: str) -> str:
        entries = parse_srt(srt_content)
        entries = self.process_entries(entries)
        # Rassembler le contenu SRT
        result = "\n\n".join(str(entry) for entry in entries)
        # Diviser les entrées multi-lignes
        if self.split_lines:
            result = process_srt_with_line_split(result)
        # Nettoyage final : supprimer les blocs vides éventuels
        result = re.sub(r'\n\n+', '\n\n', result)
        return result
```

FIGURE 4.8 – Implémentation de la correction de sous-titres Vosk-Partie 4

```python
# ----------------------------------------------------------------------
# Fonction principale
# ----------------------------------------------------------------------
def main():
    # Demande de la langue à utiliser
    lang = input("Entrez la langue souhaitée (laisser vide pour 'fr') : ").strip() or "fr"
    # Lecture du fichier SRT d'entrée
    with open('output_subtitles_vosk.srt', 'r', encoding='utf-8') as f:
        original_srt = f.read()

    # Lecture du SRT de référence
    # with open('reference.srt', 'r', encoding='utf-8') as f:
    #     reference_srt = f.read()
    reference_srt = None

    # Création d'une instance du processeur SRT et traitement
    processor = SRTProcessor(lang=lang, max_chars=55, split_lines=True, reference_srt=reference_srt)
    corrected_srt = processor.process_srt(original_srt)

    # Sauvegarde du résultat dans un fichier
    with open('corrected.srt', 'w', encoding='utf-8') as f:
        f.write(corrected_srt)
    print("Traitement terminé, fichier 'corrected.srt' généré.")

if __name__ == "__main__":
    main()
```

FIGURE 4.9 – Implémentation de la correction de sous-titres Vosk-Partie 5