

Ce code sert à générer de la synthèse vocale à partir de texte en utilisant un modèle de Text-to-Speech (TTS). Voici une explication détaillée des étapes :

1. Chargement du modèle et du vocodeur :

- `load_speech_model()`: Cette fonction charge un modèle de TTS et un vocodeur pour transformer le texte en audio.
- Elle utilise deux points de contrôle (bilalaye/speecht5_tts-wolof pour le modèle de TTS et microsoft/speecht5_hifigan pour le vocodeur).
- Elle vérifie également si le GPU est disponible pour accélérer les calculs.

2. Extraction de l'embedding vocal :

- Le code charge un échantillon vocal spécifique depuis un ensemble de données (cmu-arctic-xvectors) pour générer une empreinte vocale ou un style spécifique.

3. Fonction `generate_speech_from_text()` :

- Cette fonction prend en entrée un texte et utilise le modèle pour générer l'audio correspondant.
- Elle ajuste différents paramètres de génération pour contrôler la qualité et la durée de l'audio :
 - `max_length`, `min_length`, `num_beams` (paramètres de contrôle de la qualité et de la longueur).
 - Elle utilise un mécanisme de décodage par faisceau (beam search) pour améliorer la précision.

4. Exemple d'utilisation :

- Un exemple de texte wolof est défini et utilisé pour générer une synthèse vocale.
- Le résultat audio est ensuite lu directement dans Jupyter Notebook avec `IPython.display.Audio`.

Ce code est utile pour tester la synthèse vocale en wolof en utilisant un modèle pré-entraîné

```
[3]: import torch
from transformers import SpeechT5ForTextToSpeech, SpeechT5Processor
from transformers import SpeechT5HiFiGan

def load_speech_model(checkpoint="bilalaye/speecht5_tts-wolof", vocoder_checkpoint="microsoft/speecht5_hifigan"):

    # Check for GPU availability and set device accordingly
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # Load the SpeechT5 processor and model
    processor = SpeechT5Processor.from_pretrained(checkpoint)
    model = SpeechT5ForTextToSpeech.from_pretrained(checkpoint).to(device) # Move model to the correct device

    # Load the HiFi-GAN vocoder
    vocoder = SpeechT5HiFiGan.from_pretrained(vocoder_checkpoint).to(device) # Move vocoder to the correct device

    return processor, model, vocoder, device

# Example usage
processor, model, vocoder, device = load_speech_model()

# Verify the device being used
print(f"Model and vocoder loaded on device: {device}")

from datasets import load_dataset
# Load speaker embeddings (this dataset contains speaker-specific embeddings)
embeddings_dataset = load_dataset("Matthijs/cmu-arctic-xvectors", split="validation")
speaker_embedding = torch.tensor(embeddings_dataset[7306]["xvector"]).unsqueeze(0)

import torch
from transformers import SpeechT5ForTextToSpeech, SpeechT5Processor, SpeechT5HiFiGan
from IPython.display import Audio, display

def generate_speech_from_text(text,
                              speaker_embedding=speaker_embedding,
                              processor=processor,
                              model=model,
                              vocoder=vocoder):
    # Parameters for text-to-speech generation
    max_text_positions = model.config.max_text_positions # Token limit
    max_length = model.config.max_length * 1.2 # Slightly extended max_length
    min_length = max_length // 3 # Adjust based on max_length
    num_beams = 7 # Use beam search for better quality
    temperature = 0.6 # Reduce temperature for stability

    # Tokenize the input text and move input tensor to the correct device
    inputs = processor(text=text, return_tensors="pt", padding=True, truncation=True, max_length=max_text_positions)
    inputs = {key: value.to(model.device) for key, value in inputs.items()} # Move inputs to device

    # Generate speech
    speech = model.generate(
        inputs["input_ids"],
        speaker_embeddings=speaker_embedding.to(model.device), # Ensure speaker_embedding is also on the correct device
        vocoder=vocoder,
        max_length=int(max_length),
        min_length=int(min_length),
        num_beams=num_beams,
        temperature=temperature,
        no_repeat_ngram_size=3,
        repetition_penalty=1.5,
        eos_token_id=None,
        use_cache=True
    )

    # Detach the speech from the computation graph and move it to CPU
    speech = speech.detach().cpu().numpy()

    # Play the generated speech using IPython Audio
    display(Audio(speech, rate=16000))

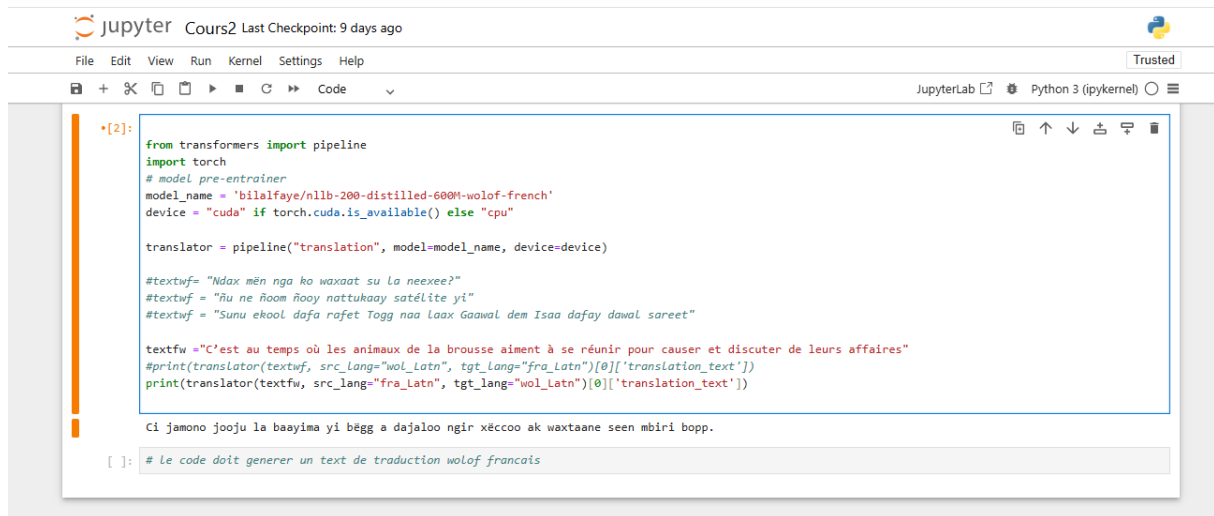
# Example usage
text = "Ci jamono jooju la baayima yi bëgg a dajaloo ngir xëccoo ak waxtaane seen mbiri bopp."
#text = "Waañ wi warul am benn mbót Mbubbu Seex Ka mi rafet na Nee nañu tombe moo waral aay gaaf Mbonaat bari na ci àll bi tey"
#text = "Am na xale bu dem sàmm baayima yi. Dafa ñàkk aw nag bu tudd Boori. Pàppaam ne ko: Soo ma indiiwaatoul Boori"
#text = "Nën nga ko baamtu, soo ko bëggee?"
generate_speech_from_text(text)
```

Model and vocoder loaded on device: cpu

[]: # apres traitement le code genere une petite audio

[]:

Ce code utilise la bibliothèque **Hugging Face Transformers** pour effectuer une traduction entre le français et le wolof en utilisant le modèle pré-entraîné bilalface/nllb-200-distilled-600M-wolof-french. Voici une explication détaillée :



```
•[2]:  
from transformers import pipeline  
import torch  
# model pre-entrainer  
model_name = 'bilalface/nllb-200-distilled-600M-wolof-french'  
device = "cuda" if torch.cuda.is_available() else "cpu"  
  
translator = pipeline("translation", model=model_name, device=device)  
  
#textwf= "Ndax mën nga ho waxaat su la neexee?"  
#textwf = "ñu ne ñoom ñooy nattukaay satelite yi"  
#textwf = "Sunu ekool dafa rafet Togg naa Laax Gaawal dem Isaa dafay dawal sareet"  
  
textfw = "C'est au temps où les animaux de la brousse aiment à se réunir pour causer et discuter de leurs affaires"  
#print(translator(textwf, src_lang="wol_Latn", tgt_lang="fra_Latn")[0]['translation_text'])  
print(translator(textfw, src_lang="fra_Latn", tgt_lang="wol_Latn")[0]['translation_text'])  
  
Ci jamono jooju la baayima yi bëgg a dajaloo ngir xëccoo ak waxtaane seen mbiri bopp.  
  
[ ]: # Le code doit generer un text de traduction wolof francais
```

Le fichier app.py sert de cerveau à votre application web. Il contient la logique qui gère les requêtes des utilisateurs, traite les données et interagit avec les différents composants de votre application. Plus précisément, dans ce cas, il s'occupe de la synthèse vocale à partir de texte.

Fonctionnalités clés :

1. **Framework Flask** : Il utilise le framework Flask pour créer et gérer l'application web. Flask permet de définir des routes (URL) et les fonctions qui seront appelées lorsque ces routes sont accédées.
2. **Chargement de modèles de traitement de la parole** : Il charge des modèles pré-entraînés de deep learning pour la synthèse vocale (text-to-speech). Ces modèles sont capables de convertir du texte en parole de manière réaliste. Il utilise notamment les modèles SpeechT5 de la bibliothèque transformers.
3. **Gestion des requêtes** :
 - Il définit une route racine ("/") qui rend le template index.html. Ce template contient probablement l'interface utilisateur de votre application (un formulaire pour entrer du texte, par exemple).
 - Il définit une route "/generate" qui gère les requêtes POST. Les requêtes POST sont utilisées pour envoyer des données au serveur (ici, le texte à convertir en parole). Cette route appelle la fonction generate().
 - Il définit une route "/audio/<filename>" pour servir les fichiers audio générés.
4. **Fonction generate()** :
 - C'est la fonction centrale de l'application. Elle reçoit le texte à convertir en parole.
 - Elle génère un nom de fichier unique pour la sortie audio.

- Elle utilise les modèles chargés pour effectuer la conversion du texte en parole. Elle utilise également un "embedding" de locuteur pour contrôler la voix de sortie.
- Elle enregistre le fichier audio au format WAV.
- Elle retourne le nom du fichier audio.

5. Fonction `get_audio()` :

- Cette fonction est appelée lorsqu'un utilisateur accède à un fichier audio via la route `"/audio/<filename>"`.
- Elle sert le fichier audio à partir du serveur.

En résumé :

`app.py` est le fichier qui contient toute la logique métier de votre application de synthèse vocale. Il gère les requêtes web, charge les modèles de deep learning, effectue la conversion du texte en parole et sert les fichiers audio générés. Il utilise le framework Flask pour faciliter la création de l'application web.

```
1 # app.py
2 from flask import Flask, render_template, request, send_file
3 import torch
4 from transformers import SpeechT5ForTextToSpeech, SpeechT5Processor, SpeechT5HifiGAN
5 from datasets import load_dataset
6 import soundfile as sf
7 import uuid
8 import os
9
10 app = Flask(__name__)
11
12 # Chargement des modèles une fois au démarrage
13 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
14 processor = SpeechT5Processor.from_pretrained("bilalrhyane/speecht5_tts-wolof")
15 model = SpeechT5ForTextToSpeech.from_pretrained("bilalrhyane/speecht5_tts-wolof").to(device)
16 vocoder = SpeechT5HifiGAN.from_pretrained("microsoft/speecht5_hifigan").to(device)
17 embeddings_dataset = load_dataset("Matthijs/cmu-arctic-xvectors", split="validation")
18 speaker_embedding = torch.tensor(embeddings_dataset[7306]["xvector"]).unsqueeze(0)
19
20 @app.route('/')
21 def home():
22     return render_template('index.html')
23
24 @app.route('/generate', methods=['POST'])
25 def generate():
26     try:
27         text = request.json['text']
28         filename = f"output_{uuid.uuid4().hex}.wav"
29
30         # Génération de la parole
31         inputs = processor(text=text, return_tensors="pt", padding=True, truncation=True)
32         inputs = {key: value.to(device) for key, value in inputs.items()}
33
34         speech = model.generate(
35             inputs["input_ids"],
36             speaker_embeddings=speaker_embedding.to(device),
37             vocoder=vocoder,
38             num_beams=7,
39             temperature=0.6
40         )
41
42         # Sauvegarde du fichier audio
43         sf.write(filename, speech.detach().cpu().numpy().squeeze(), 16000)
44
45         return {'filename': filename}
46     except Exception as e:
47         return {'error': str(e)}, 500
48
49 @app.route('/audio/<filename>')
50 def get_audio(filename):
51     return send_file(filename, mimetype='audio/wav')
52
53 if __name__ == '__main__':
54     app.run(host='0.0.0.0', port=5000, debug=True)
```

2025-02-14 00:52:18.258722: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.

* Debugger is active!

* Debugger PIN: 392-472-164

127.0.0.1 - - [14/Feb/2025 00:52:29] "GET / HTTP/1.1" 200 -

127.0.0.1 - - [14/Feb/2025 00:52:35] "GET /favicon.ico HTTP/1.1" 404 -

Asking to truncate to max_length but no maximum length is provided and the model has no predefined maximum length. Default to no truncation.

127.0.0.1 - - [14/Feb/2025 00:54:40] "POST /generate HTTP/1.1" 200 -

127.0.0.1 - - [14/Feb/2025 00:56:13] "GET /audio/output_f4a4763f5b8c4cf58e5b9611b4b1d6f2.wav HTTP/1.1" 206 -

Ce code frontальный permet à l'utilisateur de saisir du texte en Wolof, de le soumettre à votre application Flask pour la conversion en parole, et d'écouter le résultat via un lecteur audio intégré à la page web.

Points importants:

- **Communication avec le backend (app.py):** Le code JavaScript communique avec votre application Flask via des requêtes HTTP (POST pour envoyer le texte, GET implicite pour récupérer le fichier audio).
- **Affichage dynamique:** Le lecteur audio est initialement caché et n'est affiché que lorsque la génération de la parole réussit.
- **Gestion des erreurs:** Le code inclut une gestion des erreurs pour informer l'utilisateur en cas de problème.

```

39 </head>
40 <body>
41   <div class="container">
42     <h1>Synthèse Vocale en Wolof</h1>
43     <textarea id="textInput" placeholder="Entrez votre texte en Wolof ici..."></textarea>
44     <button onclick="generateSpeech()">Générer la parole</button>
45     <div id="audioPlayer" style="display: none;">
46       <audio controls id="audioElement"></audio>
47     </div>
48     <div id="error" style="color: red; margin-top: 10px;"></div>
49   </div>
50
51   <script>
52     async function generateSpeech() {
53       const text = document.getElementById('textInput').value;
54       const errorDiv = document.getElementById('error');
55       errorDiv.textContent = '';
56
57       try {
58         const response = await fetch('/generate', {
59           method: 'POST',
60           headers: {'Content-Type': 'application/json'},
61           body: JSON.stringify({text: text})
62         });
63
64         if (!response.ok) {
65           throw new Error(await response.text());
66         }
67
68         const data = await response.json();
69         const audioElement = document.getElementById('audioElement');
70         document.getElementById('audioPlayer').style.display = 'block';
71         audioElement.src = '/audio/${data.filename}';
72
73       } catch (error) {
74         errorDiv.textContent = `Erreur: ${error.message}`;
75       }
76     }
77   </script>
78 </body>
79 </html>

```

Synthèse Vocale en Wolof

Ci jamono jooju la baayima yi bëgg a dajaloo ngir xëccoo ak waxtaane seen mbiri bopp.

Générer la parole

▶ 0:00 / 0:06 🔊 ⋮