LANGAGE
PYTHON

Septembre 2025

Assistant vocal IA multilingue

Projet en Open Source

Nicolas Leca

RÉSUMÉ DU PROJET

Le projet consiste à créer un assistant vocal local, fonctionnant entièrement hors-ligne, capable de :

- Enregistrer la voix de l'utilisateur via le microphone
- Transcrire l'audio en texte (ASR – speech-to-text)
- Envoyer la transcription à un LLM local (Qwen2:7B via Ollama)
- Récupérer et vocaliser la réponse grâce à un moteur TTS
- Gérer automatiquement la détection ou sélection de langue (FR/EN/CN) pour l'ASR et le TTS

Aucun service cloud n'est utilisé : tout tourne en local, pour des raisons de performance, de confidentialité et de coût.

ARCHITECTURE DU CODE

1. **Audio → Microphone**

Enregistrement temporaire d'un fichier audio (WAV) via sounddevice

---

2. **ASR via faster-whisper**

Transcription de l'audio en texte, avec un module "smart detection" pour choisir la bonne langue.

---

3. **Envoi de la transcription à Ollama (Qwen2:7B)**

Génération d'une réponse textuelle par un modèle LLM open source, exécuté localement.

---

4. **TTS via XTTS v2 (coqui-ai)**

Conversion du texte en audio, avec possibilité de clonage de voix via un fichier de référence.

---

5. **Lecture de la réponse vocale**

Lecture du fichier WAV généré avec sounddevice ou playsound.

LEXIQUE ET OUTILS UTILISÉS

ASR (AUTOMATIC SPEECH RECOGNITION)

L'ASR est la technologie qui convertit un signal audio en texte exploitable.
Dans ce projet, il sert à transcrire la voix de l'utilisateur.
Il permet de transformer une question orale en texte avant de l'envoyer au modèle de langage.
L'ASR utilisé est faster-whisper, une version optimisée du modèle Whisper :

- plus rapide
- plus léger
- open source
- multilingue (FR/EN/CN..)

TTS (TEXT-TO-SPEECH)

Le TTS est le processus inverse : convertir un texte en voix synthétique.
Dans ce projet, il permet à l'assistant de répondre oralement, comme un véritable assistant vocal.
Le moteur utilisé est XTTS v2, qui offre :

- synthèse vocale naturelle
- prise en charge native du français, anglais et coréen
- clonage de voix (optionnel) via un échantillon de 6–10 secondes
- fonctionnement totalement hors-ligne

LLM (LARGE LANGUAGE MODEL)

Un LLM est un modèle de langage de grande taille capable de comprendre, analyser et générer du texte.
Dans ce projet, il sert à :

- analyser la transcription de la question
- produire une réponse pertinente
- converser en langage naturel
- Le LLM fonctionne localement via Ollama, ce qui garantit la confidentialité des données et supprime toute dépendance au cloud.

OBJECTIFS DE CE PROJET

LE BESOIN

Dans ma pratique du **chinois**, j'ai vite réalisé que l'expression et la compréhension **orale** sont des éléments clé dans l'apprentissage de la langue.
Ayant de bonnes bases en langage **python**, je me suis dit qu'avec les ressources qui étaient à ma disposition (ordinateur relativement puissant, nouvelles technologies récentes en **Open Source**, IA conversationnelle accessible en **local**), j'étais en mesure de faire **moi-même** un assistant vocal avec l'IA.

Les solutions comme **ChatGPT** utilisent souvent les données personnelles pour s'entraîner et demandent beaucoup de **ressources** serveur. J'ai donc choisi de développer ma propre alternative : une solution **gratuite** qui protège la **vie privée** et qui m'apprend à maîtriser l'utilisation des IA en mode **applicatif**.

Le projet repose entièrement sur des outils open source (Whisper, Coqui, Qwen2, Ollama), ce qui garantit :

- Transparence
- Confidentialité totale
- Aucun coût d'utilisation
- Possibilité de personnaliser et améliorer le système

Performance locale

- faster-whisper offre une transcription très rapide en local
- Ollama permet d'exécuter des LLM sans configuration complexe
- Coqui TTS génère de l'audio haute qualité en quelques millisecondes

Multilingue

Whisper et XTTS sont capables de gérer FR/EN/CN, ce qui était une contrainte importante du projet.

Modularité

Chaque étape (ASR → LLM → TTS) est indépendante, ce qui permet d'améliorer ou de remplacer un module facilement.

1. POURQUOI PYTHON ?

Nous avons choisi Python pour plusieurs raisons techniques :
1.1 Écosystème IA dominant
La majorité des bibliothèques de :

- Deep Learning
- ASR (Automatic Speech Recognition)
- TTS (Text-to-Speech)
- NLP

sont développées en Python.
Les frameworks comme OpenAI, Hugging Face ou Meta publient leurs modèles avec des bindings Python natifs.

1.2 Simplicité et rapidité de prototypage
Python permet :

- Un code lisible
- Une intégration rapide des modèles pré-entraînés
- Une grande compatibilité avec CUDA / GPU

1.3 Compatibilité avec PyTorch
La plupart des modèles modernes (Whisper, Qwen, XTTS...) sont entraînés sous PyTorch, donc Python est naturellement le langage standard.

2. COMMANDES UTILISÉES POUR INSTALLER LES MODULES

2.1 CRÉATION D'UN ENVIRONNEMENT VIRTUEL

```
python -m venv venv
venv\Scripts\activate
```

Pourquoi ?

- Isoler les dépendances
- Éviter les conflits de versions

2.2 MISE À JOUR DE PIP

pip install --upgrade pip

2.3 INSTALLATION DE PYTORCH
CONFIGURATION CPU :

pip install torch torchvision torchaudio

2.4 INSTALLATION DE FASTER-WHISPER

pip install faster-whisper

2.5 INSTALLATION DU LLM (QWEN)

pip install transformers accelerate

Pourquoi ?

- transformers permet de charger des LLM depuis Hugging Face
- accelerate optimise le chargement GPU

2..6 INSTALLATION XTTS (TEXT-TO-SPEECH)

pip install TTS

Bibliothèque développée par Coqui.

2.7 AUTRES DÉPENDANCES COURANTES

pip install numpy scipy sounddevice pyaudio

- numpy → calcul numérique
- scipy → traitement du signal
- sounddevice / pyaudio → capture micro

3. LE CODE

Importation des modules

```
import sys
import torch
import requests
from pathlib import Path
from faster_whisper import WhisperModel
from TTS.api import TTS
import sounddevice as sd
from scipy.io.wavfile import write
```

Enregistrement audio

```
def record_audio(filename="temp_input.wav", duration=5, samplerate=16000):
    print(f"🎙️ Parle pendant {duration} secondes...")
    audio = sd.rec(int(duration * samplerate), samplerate=samplerate, channels=1, dtype="int16")
    sd.wait() # attend la fin de l'enregistrement
    write(filename, samplerate, audio)
    print(f"✅ Audio enregistré : {filename}")
    return filename
```

Ce qu'on fait ici :
Enregistre un audio depuis le micro.
filename : nom du fichier .wav de sortie
duration : durée en secondes
samplerate : fréquence d'échantillonnage

Optimisation de la communication

```
ASR_MODEL = "distil-large-v3"
DEVICE_ASR = "cuda" if torch.cuda.is_available() else "cpu"
COMPUTE_ASR = "float16" if DEVICE_ASR == "cuda" else "int8"

TTS_MODEL = "tts_models/multilingual/multi-dataset/xtts_v2"
DEVICE_TTS = "cuda" if torch.cuda.is_available() else "cpu"

OLLAMA_URL = "http://localhost:11434/api/generate" # API Ollama locale
MODEL_NAME = "qwen2:7b" # modèle téléchargé via "ollama pull qwen2:7b"
```

Cette section permet de configurer l'environnement matériel et logiciel. Le programme détecte automatiquement si une carte graphique (GPU) est disponible pour accélérer les calculs, choisit les modèles d'IA à charger et définit les adresses de communication pour le cerveau de l'IA (Ollama).

Transcription de l’audio et détection de langue

```
def transcribe_audio(path):
 print("🎤 Transcription...")
 model = WhisperModel(ASR_MODEL, device=DEVICE_ASR, compute_type=COMPUTE_ASR)
 segments, info = model.transcribe(path, beam_size=5, vad_filter=True)
 text = " ".join([seg.text for seg in segments])
 print(f"Langue détectée : {info.language} (p={info.language_probability:.2f})")
 print(f"Texte : {text}")
 return text, info.language
```

Envoi de la transcription à Ollama (notre IA)

```
def ask_ollama(prompt, model=MODEL_NAME):
 print("🧠 Envoi à Ollama...")
 payload = {
  "model": model,
  "prompt": prompt,
  "stream": False # on récupère la réponse en une fois
 }
 resp = requests.post(OLLAMA_URL, json=payload)
 if resp.status_code != 200:
  raise RuntimeError(f"Erreur Ollama {resp.status_code}: {resp.text}")
 data = resp.json()
 reply = data.get("response", "").strip()
 print("🤖 Réponse :", reply)
 return reply
```

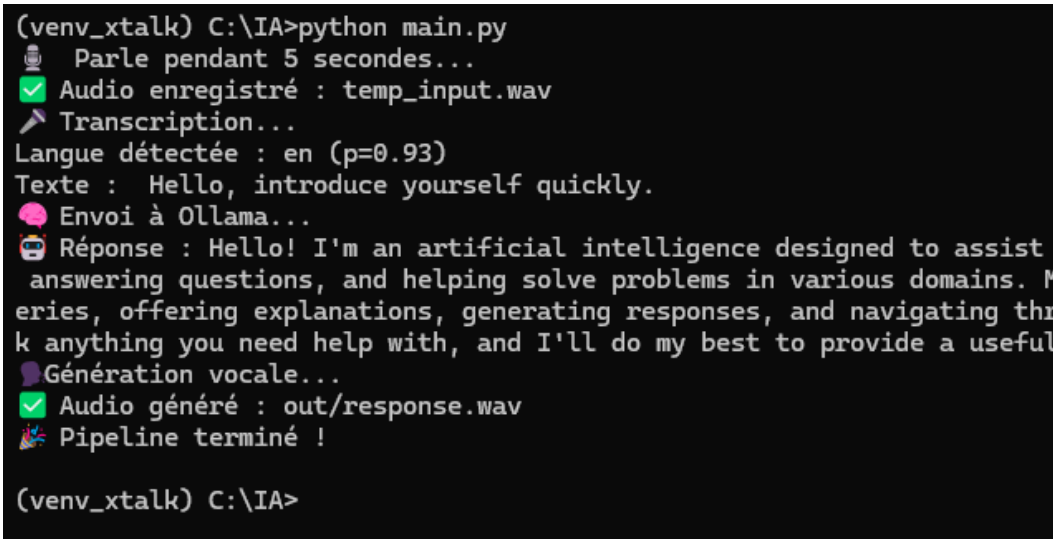
Envoi de la transcription à Ollama (notre IA)

```
def speak_text(text, language, out_path, speaker_wav=None):
 print("Génération vocale...")
 tts = TTS(TTS_MODEL).to(DEVICE_TTS)
 Path(out_path).parent.mkdir(parents=True, exist_ok=True)
 if speaker_wav:
  tts.tts_to_file(text=text, speaker_wav=speaker_wav, language=language,
file_path=out_path)
 else:
  # XTTS v2 → clonage la voix de l’IA à partir d’un extrait audio de référence en .wav
  tts.tts_to_file(text=text, speaker_wav="test.wav", language=language,
file_path=out_path)
 print(f"✅ Audio généré : {out_path}")
```

L'exécution de notre code

```
if __name__ == "__main__":
 # on enregistre directement avec le micro
 audio_file = record_audio(filename="temp_input.wav", duration=5)

 # 1. ASR
 user_text, lang = transcribe_audio(audio_file)

 # 2. Ollama
 ai_reply = ask_ollama(user_text, model=MODEL_NAME)

 # 3. TTS
 out_file = "out/response.wav"
 speak_text(ai_reply, lang, out_file, speaker_wav="test.wav")

 print("🎉 Pipeline terminé !")
```

Résultat final !



Quand je parle à l'IA en anglais pendant 5 secondes, le code détecte la langue avec un taux de probabilité (1 étant le plus haut, 0 le plus bas).
Ici, l'IA a compris que je parlais en anglais et m'a répondu dans la langue, en générant un fichier audio de sa réponse.

Je peux lui demander de me répondre en n'importe quelle langue, si je lui fournit une voix à cloner disons en chinois, l'IA répondra avec un accent authentique et cela me permettrait d'avoir des audios de compréhension orale dans n'importe quelle langue.

La prochaine étape pour moi serait de réaliser une interface graphique et d'avoir une réponse directe au lieu d'aller devoir ouvrir un fichier audio.