VALL-E-X Voice Cloning

Überblick

In diesem Projekt habe ich die Open-Source-Implementierung VALL-E-X genutzt, um Voice Cloning mit eigenen Sprachsamples zu testen.
Die Basis bildet das GitHub-Repository von Plachtaa, welches Microsofts VALL-E-X Zero-Shot-TTS-Modell nachbildet.

https://github.com/Plachtaa/VALL-E-X

https://www.microsoft.com/en-us/research/project/vall-e-x

Ziel war es eigene Stimmen-Samples zu nutzen um meine Stimme zu klonen und mit verschiedenen Ansätzen herauszufinden, wo die Grenzen des Modells liegen und um stabilere Ergebnisse zu erhalten

Umsetzung

Ich habe das bestehende VALL-E-X-Projekt lokal eingerichtet und nicht das Modell selbst verändert, sondern nur zusätzliche ergänzt, angepasste Bereiche sind:

- Eigene Sprachsamples
- Textprompts
- Kombinieren von Samples zu einem Voice-Embedding
- Erzwingen von einem CPU-kompatiblen Modus
- Laden von benötigten Modellen
- Kleinere Code-Anpassungen

Kombinierte Embeddings

Für die Arbeit mit mehreren Samples habe ich eine Hilfsfunktion erstellt, die:

- aus einem neuen .wav-Sample temporär ein Prompt-Embedding erzeugt
- Audio-Tokens und Text-Tokens extrahiert
- diese Tokens an ein bestehendes Embedding anhängt
- das kombinierte Embedding wieder speichert
- jedes neue Sample erweitert vorhandene Embedding

Sprachsamples

- Samples 1–4: unterschiedliche gesprochene Inhalte, nicht direkt auf den Textprompt abgestimmt
- Sample 5: Kein Sprechtext, nur “Geräusche” verschlechterten Ergebnisse
- Sample 6: Deutsch – für diese Tests unbrauchbar
- Samples 07–09: enthalten explizit Wörter aus „The quick brown fox jumps over the lazy dog.“ auf verschiedene Weisen

Text-Prompts

Test:

> „Sample ‘x’. Hello this is my cloned voice speaking! Do I sound at all close to the real thing? The quick brown fox jumps over the lazy dog.“

Humanized:

> Zusätze wie: „uh“, „um“, „ha ha ha“, *sigh*, *breathes*, *laughs*

Fox:

> „The quick brown fox jumps over the lazy dog.“

Hello World:

> „This is my cloned voice, hello world!“

Other:

> „She sells seashells by the seashore.“
>
> „Ha ha ha, um. Uhhh, hahah.“

Ergebnisse

Struktur der Ordner

- Single Samples
  Audio aus einem einzelnen Sprachsample
  - Ordner für *Test, Humaized* und *Fox* Prompt Ergebnisse
- Embedding
  Audio aus kombinierten Embeddings (mehrere Samples)
  - Ordner für *Fox, Hello World* and *Other* Prompt Ergebnisse

Single-Sample-Ergebnisse

Test:

> sample1_cloned.wav / sample2_cloned.wav / sample3_cloned.wav / sample4_cloned.wav **(LAUT)** / sample5_cloned.wav

Fox:

> sample1.humanized_cloned.wav / sample1.2_cloned.wav / sample2.2_cloned.wav / sample3.2_cloned.wav / sample4.2_cloned.wav / sample5.2_cloned.wav

Embedding-Ergebnisse (kombinierte Samples)

Fox:

> sample07_e.wav (STUMM) / sample07_08_e.wav / sample07-09_e.wav **(STUMM)** / sample07-09_1_e.wav / sample07-09_1_2_e.wav / sample07-09_1-3_e.wav / sample07-09_1-4_e.wav / sample07-09_1-5_e.wav / sample07-09_1-5_07_e.wav **(STUMM)** / sample07-09_1-5_07_08_e.wav / sample07-09_1-5_07-09_e.wav

Hello World:

> all_samples+07.wav / all_samples+07,08.wav / all_samples+07-09.wav / all_samples+07-09+1.wav / all_samples+07-09+1,2.wav / all_samples+07-09+1-3.wav / all_samples+07-09+1-4.wav **(SEHR LAUT)** / all_samples+07-09+1-5.wav

Other:

> all_samples_humanized.wav / all_samples_tonguetwister.wav

Mit zunehmender Anzahl kombinierter Samples wurden Stimme und Intonation konsistenter, jedoch hat erneutes Nutzen von Samples die Ergebnisse stark beeinflusst, somit sind einzelne Ergebnisse im Sprechen der Prompts gescheitert.

Fazit

Das Projekt zeigt, wie eine lokale Methode zum Klonen von Stimmen mit KI umgesetzt werden kann. Dabei wurde deutlich, dass der locale Ansatz im Vergleich zu externen Voice-Cloning-Diensten deutlich schwächer ist und gleichzeitig mehr Aufwand erfordert. Positiv ist jedoch, dass sich dadurch zeigt, dass Voice Cloning bei lokaler Umsetzung nicht einfach Auszunutzen ist und schon ein gewisses technisches Verständnis voraussetzt.

Außerdem wurde der Unterschied zwischen Voice Cloning mit einzelnen Samples und mit kombinierten Embeddings sichtbar. Einzelne Samples liefern stark vom Input abhängige Ergebnisse, während kombinierte Embeddings ein bisschen stabiler waren. Trotzdem ist auch diese Methode nicht perfekt und benötigt mehrere gute Sprachsamples, um wirklich brauchbare Resultate zu liefern.

Einen großen Einfluss hatte die Qualität der Samples. Sprache, Inhalt und Aufnahmequalität haben sehr stark auf das Ergebnis eingewirkt. Samples, die nicht zum gewünschten Text passen oder in einer anderen Sprache aufgenommen wurden, führten zu schlechteren oder sogar unbrauchbaren Resultaten. Daraus lässt sich schließen, dass man die Grenzen des Modells kennen muss, um es sinnvoll zu nutzen.

Zusätzlich habe ich gelernt, mit der Anwendung umzugehen, wobei ein großer Teil der Arbeit in das Einrichten des Projekts und in das Aufnehmen der Samples geflossen ist. Die Audiogenerierung selbst nahm ebenfalls viel Zeit in Anspruch. Insgesamt hat das Projekt gezeigt, dass gute Ergebnisse nur mit passenden Samples, Geduld und einem ausreichendem Verständnis der Modellgrenzen möglich sind.

Leider hatte ich ein Paar Probleme ein GitHub Repository zu erstellen, sowohl mit großen Dateien, als auch ohne. Deshalb hier einmal die Code Teile an denen ich gearbietet, oder etwas verändert habe.

testclone.py

```
import os
from scipy.io.wavfile import write as write_wav
from utils.generation import SAMPLE_RATE, generate_audio, preload_models
from utils.voice_enroll import update_combined_embedding

# CPU-safe
os.environ["CUDA_VISIBLE_DEVICES"] = ""
os.environ["PYTORCH_ENABLE_MPS_FALLBACK"] = "1"

# Paths
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
combined_voice_name = "sample0"
voice_file = os.path.join(BASE_DIR, "data/voices/wavs/sample08.wav") ###
output_dir = os.path.join(BASE_DIR, "results")
output_file = os.path.join(output_dir, f"{combined_voice_name}.19_cloned.wav")

# text prompt
text_to_speak = "ha ha ha, um. uhhh, hahah."

# Add to combined embedding
update_combined_embedding(combined_voice_name, voice_file)

# Generate audio using all enrolled samples
preload_models()
audio_array = generate_audio(text_to_speak, prompt=combined_voice_name)

# Save
```

```
os.makedirs(output_dir, exist_ok=True)
write_wav(output_file, SAMPLE_RATE, audio_array)
print(f"✅ Cloned voice saved to {output_file}")
```

voice_enroll.py

```
import os
import numpy as np
from utils.prompt_making import make_prompt

def update_combined_embedding(voice_name, new_wav_path, combined_dir="./customs", transcript=None):
  os.makedirs(combined_dir, exist_ok=True)

  # Create temp embedding
  make_prompt(
    name=voice_name,
    audio_prompt_path=new_wav_path,
    transcript=transcript
  )

  temp_npz = os.path.join(combined_dir, f"{voice_name}.npz")

  with np.load(temp_npz, allow_pickle=True) as data:
    new_audio_tokens = data["audio_tokens"]
    new_text_tokens = data["text_tokens"]
    lang_code = data["lang_code"]

  # close file before deleting
  os.remove(temp_npz)

  combined_path = os.path.join(combined_dir, f"{voice_name}.npz")

  if os.path.exists(combined_path):
    with np.load(combined_path, allow_pickle=True) as existing:
      audio_tokens = np.concatenate(
        [existing["audio_tokens"], new_audio_tokens], axis=0
      )
      text_tokens = np.concatenate(
        [existing["text_tokens"], new_text_tokens], axis=0
      )
      # keep original language
      lang_code = existing["lang_code"]
  else:
    audio_tokens = new_audio_tokens
    text_tokens = new_text_tokens

  np.savez(
    combined_path,
    audio_tokens=audio_tokens,
    text_tokens=text_tokens,
    lang_code=lang_code
  )

  print(f"✅ Updated combined embedding: {combined_path}")
```

prompt_making.py

```
import os
import torch
import torchaudio
import logging
import langid
import whisper
langid.set_languages(['en', 'zh', 'ja'])

import numpy as np
from data.tokenizer import (
  AudioTokenizer,
  tokenize_audio,
)
from data.collation import get_text_token_collater
from utils.g2p import PhonemeBpeTokenizer
```

```
from macros import *

text_tokenizer = PhonemeBpeTokenizer(tokenizer_path="./utils/g2p/bpe_69.json")
text_collater = get_text_token_collater()

device = torch.device("cpu")
if torch.cuda.is_available():
  device = torch.device("cuda", 0)
if torch.backends.mps.is_available():
  device = torch.device("mps")
codec = AudioTokenizer(device)

if not os.path.exists("./whisper/"): os.mkdir("./whisper/")
whisper_model = None

@torch.no_grad()
def transcribe_one(model, audio_path):
  # load audio and pad/trim it to fit 30 seconds
  audio = whisper.load_audio(audio_path)
  audio = whisper.pad_or_trim(audio)

  # make log-Mel spectrogram and move to the same device as the model
  mel = whisper.log_mel_spectrogram(audio).to(model.device)

  # detect the spoken language
  _, probs = model.detect_language(mel)
  print(f"Detected language: {max(probs, key=probs.get)}")
  lang = max(probs, key=probs.get)
  # decode the audio
  options = whisper.DecodingOptions(temperature=1.0, best_of=5, fp16=False if device == torch.device("cpu") else True, sample_len=150)
  result = whisper.decode(model, mel, options)

  # print the recognized text
  print(result.text)

  text_pr = result.text
  if text_pr.strip(" ")[-1] not in "?!.,。， ？！。、":
    text_pr += "."
  return lang, text_pr

def make_prompt(name, audio_prompt_path, transcript=None):
  global model, text_collater, text_tokenizer, codec
  wav_pr, sr = torchaudio.load(audio_prompt_path)
  # check length
  if wav_pr.size(-1) / sr > 15:
    raise ValueError(f"Prompt too long, expect length below 15 seconds, got {wav_pr / sr} seconds.")
  if wav_pr.size(0) == 2:
    wav_pr = wav_pr.mean(0, keepdim=True)
  text_pr, lang_pr = make_transcript(name, wav_pr, sr, transcript)

  # tokenize audio
  encoded_frames = tokenize_audio(codec, (wav_pr, sr))
  audio_tokens = encoded_frames[0][0].transpose(2, 1).cpu().numpy()

  # tokenize text
  phonemes, langs = text_tokenizer.tokenize(text=f"{text_pr}".strip())
  text_tokens, enroll_x_lens = text_collater(
    [
      phonemes
    ]
  )

  message = f"Detected language: {lang_pr}\n Detected text {text_pr}\n"

  # save as npz file
  save_path = os.path.join("./customs/", f"{name}.npz")
  np.savez(save_path, audio_tokens=audio_tokens, text_tokens=text_tokens, lang_code=lang2code[lang_pr])
  logging.info(f"Successful. Prompt saved to {save_path}")
```

```
def make_transcript(name, wav, sr, transcript=None):

    if not isinstance(wav, torch.FloatTensor):
        wav = torch.tensor(wav)
    if wav.abs().max() > 1:
        wav /= wav.abs().max()
    if wav.size(-1) == 2:
        wav = wav.mean(-1, keepdim=False)
    if wav.ndim == 1:
        wav = wav.unsqueeze(0)
    assert wav.ndim and wav.size(0) == 1
    if transcript is None or transcript == "":
        logging.info("Transcript not given, using Whisper...")
        global whisper_model
        if whisper_model is None:
            whisper_model = whisper.load_model("medium", download_root=os.path.join(os.getcwd(), "whisper"))
        whisper_model.to(device)
        torchaudio.save(f"./prompts/{name}.wav", wav, sr)
        lang, text = transcribe_one(whisper_model, f"./prompts/{name}.wav")
        lang_token = lang2token[lang]
        text = lang_token + text + lang_token
        os.remove(f"./prompts/{name}.wav")
        whisper_model.cpu()
    else:
        text = transcript
        lang, _ = langid.classify(text)
        lang_token = lang2token[lang]
        text = lang_token + text + lang_token

    torch.cuda.empty_cache()
    return text, lang
```