Notes

LibreTTS is used here, multispeaker eSpeak-ng is doing phonemisation

Install packages and download models

```
%%shell
git clone https://github.com/yl4579/StyleTTS2.git
cd StyleTTS2
pip install SoundFile torchaudio munch torch pydub pyyaml librosa nltk matplotlib accelerate transformers phonemizer einops e
sudo apt-get install espeak-ng
git-lfs clone https://huggingface.co/yl4579/StyleTTS2-LibriTTS
mv StyleTTS2-LibriTTS/Models .
mv StyleTTS2-LibriTTS/reference_audio.zip .
unzip reference_audio.zip
mv reference_audio Demo/reference_audio
```

```
fatal: destination path 'StyleTTS2' already exists and is not an empty directory.
Collecting git+https://github.com/resemble-ai/monotonic_align.git
  Cloning https://github.com/resemble-ai/monotonic_align.git to /tmp/pip-req-build-1s9tkxc5
  Running command git clone --filter=blob:none --quiet https://github.com/resemble-ai/monotonic_align.git /tmp/pip-req-bu
  Resolved https://github.com/resemble-ai/monotonic_align.git to commit 78b985be210a03d08bc3acc01c4df0442105366f
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: SoundFile in /usr/local/lib/python3.11/dist-packages (0.13.1)
Requirement already satisfied: torchaudio in /usr/local/lib/python3.11/dist-packages (2.6.0+cu124)
Requirement already satisfied: munch in /usr/local/lib/python3.11/dist-packages (4.0.0)
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages (2.6.0+cu124)
Requirement already satisfied: pydub in /usr/local/lib/python3.11/dist-packages (0.25.1)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.11/dist-packages (6.0.2)
Requirement already satisfied: librosa in /usr/local/lib/python3.11/dist-packages (0.10.2.post1)
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (3.9.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: accelerate in /usr/local/lib/python3.11/dist-packages (1.3.0)
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.48.3)
Requirement already satisfied: phonemizer in /usr/local/lib/python3.11/dist-packages (3.3.0)
Requirement already satisfied: einops in /usr/local/lib/python3.11/dist-packages (0.8.1)
Requirement already satisfied: einops-exts in /usr/local/lib/python3.11/dist-packages (0.0.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (4.67.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (4.12.2)
Requirement already satisfied: cffi>=1.0 in /usr/local/lib/python3.11/dist-packages (from SoundFile) (1.17.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from SoundFile) (2.0.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch) (3.17.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch) (2024.10.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch) (
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch) (
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch) (9.1.0
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from torch) (12.4
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch) (11.2.
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from torch) (10
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from torch) (11
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (from torch) (
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch) (0.6
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch) (12.4.1
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch) (1
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch)
Requirement already satisfied: audioread>=2.1.9 in /usr/local/lib/python3.11/dist-packages (from librosa) (3.0.1)
Requirement already satisfied: scipy>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from librosa) (1.14.1)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.11/dist-packages (from librosa) (1.6.1)
Requirement already satisfied: joblib>=0.14 in /usr/local/lib/python3.11/dist-packages (from librosa) (1.4.2)
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.11/dist-packages (from librosa) (4.4.2)
Requirement already satisfied: numba>=0.51.0 in /usr/local/lib/python3.11/dist-packages (from librosa) (0.60.0)
Requirement already satisfied: pooch>=1.1 in /usr/local/lib/python3.11/dist-packages (from librosa) (1.8.2)
Requirement already satisfied: soxr>=0.3.2 in /usr/local/lib/python3.11/dist-packages (from librosa) (0.5.0.post1)
Requirement already satisfied: lazy-loader>=0.1 in /usr/local/lib/python3.11/dist-packages (from librosa) (0.4)
Requirement already satisfied: msgpack>=1.0 in /usr/local/lib/python3.11/dist-packages (from librosa) (1.1.0)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk) (8.1.8)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
```

Load models

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

Imports

```
%cd StyleTTS2
```

```
/content/StyleTTS2
```

```
import torch
torch.manual_seed(0)
torch.backends.cudnn.benchmark = False
torch.backends.cudnn.deterministic = True

import random
random.seed(0)

import numpy as np
np.random.seed(0)

# load packages
import time
import random
import yaml
from munch import Munch
import numpy as np
import torch
from torch import nn
import torch.nn.functional as F
import torchaudio
import librosa
from nltk.tokenize import word_tokenize

from models import *
from utils import *
from text_utils import TextCleaner
textclenaer = TextCleaner()
```

```
177
```

Utlitity Functions

```
%matplotlib inline

to_mel = torchaudio.transforms.MelSpectrogram(
    n_mels=80, n_fft=2048, win_length=1200, hop_length=300)
mean, std = -4, 4

def length_to_mask(lengths):
    mask = torch.arange(lengths.max()).unsqueeze(0).expand(lengths.shape[0], -1).type_as(lengths)
    mask = torch.gt(mask+1, lengths.unsqueeze(1))
    return mask

def preprocess(wave):
    wave_tensor = torch.from_numpy(wave).float()
    mel_tensor = to_mel(wave_tensor)
    mel_tensor = (torch.log(1e-5 + mel_tensor.unsqueeze(0)) - mean) / std
    return mel_tensor

# Remove silence < 30dB, Style TTS needs Sampling rate 24kHz
def compute_style(path):
    wave, sr = librosa.load(path, sr=24000)
    audio, index = librosa.effects.trim(wave, top_db=30)
    if sr != 24000:
        audio = librosa.resample(audio, sr, 24000)
    mel_tensor = preprocess(audio).to(device)

    with torch.no_grad():
        ref_s = model.style_encoder(mel_tensor.unsqueeze(1))
        ref_p = model.predictor_encoder(mel_tensor.unsqueeze(1))

    return torch.cat([ref_s, ref_p], dim=1)
```

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

Phonemisation

Given a text, get sequence of phonemes

```
# load phonemizer
import phonemizer
global_phonemizer = phonemizer.backend.EspeakBackend(language='en-us', preserve_punctuation=True,  with_stress=True)
```

Loding F0, BERT model, Diffusion models

```
config = yaml.safe_load(open("Models/LibriTTS/config.yml"))

# load pretrained ASR model
ASR_config = config.get('ASR_config', False)
ASR_path = config.get('ASR_path', False)
text_aligner = load_ASR_models(ASR_path, ASR_config)

# load pretrained F0 model
F0_path = config.get('F0_path', False)
pitch_extractor = load_F0_models(F0_path)

# load BERT model
from Utils.PLBERT.util import load_plbert
BERT_path = config.get('PLBERT_dir', False)
plbert = load_plbert(BERT_path)

model_params = recursive_munch(config['model_params'])
model = build_model(model_params, text_aligner, pitch_extractor, plbert)
_ = [model[key].eval() for key in model]
_ = [model[key].to(device) for key in model]

params_whole = torch.load("Models/LibriTTS/epochs_2nd_00020.pth", map_location='cpu')
params = params_whole['net']

for key in model:
    if key in params:
        print('%s loaded' % key)
        try:
            model[key].load_state_dict(params[key])
        except:
            from collections import OrderedDict
            state_dict = params[key]
            new_state_dict = OrderedDict()
            for k, v in state_dict.items():
                name = k[7:] # remove `module.`
                new_state_dict[name] = v
            # load params
            model[key].load_state_dict(new_state_dict, strict=False)
#             except:
#                 _load(params[key], model[key])
_ = [model[key].eval() for key in model]

from Modules.diffusion.sampler import DiffusionSampler, ADPM2Sampler, KarrasSchedule

sampler = DiffusionSampler(
    model.diffusion.diffusion,
    sampler=ADPM2Sampler(),
    sigma_schedule=KarrasSchedule(sigma_min=0.0001, sigma_max=3.0, rho=9.0), # empirical parameters
    clamp=False
)
```

```
bert loaded
bert_encoder loaded
predictor loaded
decoder loaded
text_encoder loaded
predictor_encoder loaded
style_encoder loaded
diffusion loaded
text_aligner loaded
pitch_extractor loaded
mpd loaded
msd loaded
wd loaded
```

Inferencing

- text, Speaker reference, weightage for styles of models (alpha, beta),
- diffusion step - balancing between noise and time
- Scaling the embedding parameters

```
def inference(text, ref_s, alpha = 0.3, beta = 0.7, diffusion_steps=5, embedding_scale=1):
    text = text.strip()
    ps = global_phonemizer.phonemize([text])
    #ps = word_tokenize(ps[0])
    ps = ' '.join(ps)
    tokens = textclenaer(ps)
    tokens.insert(0, 0)
    tokens = torch.LongTensor(tokens).to(device).unsqueeze(0)

    with torch.no_grad():
        input_lengths = torch.LongTensor([tokens.shape[-1]]).to(device)
        text_mask = length_to_mask(input_lengths).to(device)

        t_en = model.text_encoder(tokens, input_lengths, text_mask)
        bert_dur = model.bert(tokens, attention_mask=(~text_mask).int())
        d_en = model.bert_encoder(bert_dur).transpose(-1, -2)

        s_pred = sampler(noise = torch.randn((1, 256)).unsqueeze(1).to(device),
                                          embedding=bert_dur,
                                          embedding_scale=embedding_scale,
                                            features=ref_s, # reference from the same speaker as the embedding
                                             num_steps=diffusion_steps).squeeze(1)


        s = s_pred[:, 128:]
        ref = s_pred[:, :128]

        ref = alpha * ref + (1 - alpha)  * ref_s[:, :128]
        s = beta * s + (1 - beta)  * ref_s[:, 128:]

        d = model.predictor.text_encoder(d_en,
                                         s, input_lengths, text_mask)

        x, _ = model.predictor.lstm(d)
        duration = model.predictor.duration_proj(x)

        duration = torch.sigmoid(duration).sum(axis=-1)
        pred_dur = torch.round(duration.squeeze()).clamp(min=1)


        pred_aln_trg = torch.zeros(input_lengths, int(pred_dur.sum().data))
        c_frame = 0
        for i in range(pred_aln_trg.size(0)):
            pred_aln_trg[i, c_frame:c_frame + int(pred_dur[i].data)] = 1
            c_frame += int(pred_dur[i].data)

        # encode prosody
        en = (d.transpose(-1, -2) @ pred_aln_trg.unsqueeze(0).to(device))
        if model_params.decoder.type == "hifigan":
            asr_new = torch.zeros_like(en)
            asr_new[:, :, 0] = en[:, :, 0]
            asr_new[:, :, 1:] = en[:, :, 0:-1]
            en = asr_new

        F0_pred, N_pred = model.predictor.F0Ntrain(en, s)

        asr = (t_en @ pred_aln_trg.unsqueeze(0).to(device))
        if model_params.decoder.type == "hifigan":
            asr_new = torch.zeros_like(asr)
            asr_new[:, :, 0] = asr[:, :, 0]
            asr_new[:, :, 1:] = asr[:, :, 0:-1]
            asr = asr_new

        out = model.decoder(asr,
                                F0_pred, N_pred, ref.squeeze().unsqueeze(0))


    return out.squeeze().cpu().numpy()[..., :-50] # weird pulse at the end of the model, need to be fixed later
```

Long Inference

- Cut the audio (>512) and make the inference

```
def LFinference(text, s_prev, ref_s, alpha = 0.3, beta = 0.7, t = 0.7, diffusion_steps=5, embedding_scale=1):
  text = text.strip()
  ps = global_phonemizer.phonemize([text])
  #ps = word_tokenize(ps[0])
  ps = ' '.join(ps)
  ps = ps.replace('``', '"')
  ps = ps.replace("''", '"')

  tokens = textclenaer(ps)
  tokens.insert(0, 0)
  tokens = torch.LongTensor(tokens).to(device).unsqueeze(0)

  with torch.no_grad():
      input_lengths = torch.LongTensor([tokens.shape[-1]]).to(device)
      text_mask = length_to_mask(input_lengths).to(device)

      t_en = model.text_encoder(tokens, input_lengths, text_mask)
      bert_dur = model.bert(tokens, attention_mask=(~text_mask).int())
      d_en = model.bert_encoder(bert_dur).transpose(-1, -2)

      s_pred = sampler(noise = torch.randn((1, 256)).unsqueeze(1).to(device),
                                        embedding=bert_dur,
                                        embedding_scale=embedding_scale,
                                          features=ref_s, # reference from the same speaker as the embedding
                                            num_steps=diffusion_steps).squeeze(1)

      if s_prev is not None:
          # convex combination of previous and current style
          s_pred = t * s_prev + (1 - t) * s_pred

      s = s_pred[:, 128:]
      ref = s_pred[:, :128]

      ref = alpha * ref + (1 - alpha)  * ref_s[:, :128]
      s = beta * s + (1 - beta)  * ref_s[:, 128:]

      s_pred = torch.cat([ref, s], dim=-1)

      d = model.predictor.text_encoder(d_en,
                                        s, input_lengths, text_mask)

      x, _ = model.predictor.lstm(d)
      duration = model.predictor.duration_proj(x)

      duration = torch.sigmoid(duration).sum(axis=-1)
      pred_dur = torch.round(duration.squeeze()).clamp(min=1)


      pred_aln_trg = torch.zeros(input_lengths, int(pred_dur.sum().data))
      c_frame = 0
      for i in range(pred_aln_trg.size(0)):
          pred_aln_trg[i, c_frame:c_frame + int(pred_dur[i].data)] = 1
          c_frame += int(pred_dur[i].data)

      # encode prosody
      en = (d.transpose(-1, -2) @ pred_aln_trg.unsqueeze(0).to(device))
      if model_params.decoder.type == "hifigan":
          asr_new = torch.zeros_like(en)
          asr_new[:, :, 0] = en[:, :, 0]
          asr_new[:, :, 1:] = en[:, :, 0:-1]
          en = asr_new

      F0_pred, N_pred = model.predictor.F0Ntrain(en, s)

      asr = (t_en @ pred_aln_trg.unsqueeze(0).to(device))
      if model_params.decoder.type == "hifigan":
          asr_new = torch.zeros_like(asr)
          asr_new[:, :, 0] = asr[:, :, 0]
          asr_new[:, :, 1:] = asr[:, :, 0:-1]
          asr = asr_new

      out = model.decoder(asr,
                                F0_pred, N_pred, ref.squeeze().unsqueeze(0))
```

```
    return out.squeeze().cpu().numpy()[..., :-100], s_pred # weird pulse at the end of the model, need to be fixed later
```

Additional Reference Text

- Give reference text

```
def STinference(text, ref_s, ref_text, alpha = 0.3, beta = 0.7, diffusion_steps=5, embedding_scale=1):
    text = text.strip()
    ps = global_phonemizer.phonemize([text])
    #ps = word_tokenize(ps[0])
    ps = ' '.join(ps)

    tokens = textclenaer(ps)
    tokens.insert(0, 0)
    tokens = torch.LongTensor(tokens).to(device).unsqueeze(0)

    ref_text = ref_text.strip()
    ps = global_phonemizer.phonemize([ref_text])
    #ps = word_tokenize(ps[0])
    ps = ' '.join(ps)

    ref_tokens = textclenaer(ps)
    ref_tokens.insert(0, 0)
    ref_tokens = torch.LongTensor(ref_tokens).to(device).unsqueeze(0)


    with torch.no_grad():
        input_lengths = torch.LongTensor([tokens.shape[-1]]).to(device)
        text_mask = length_to_mask(input_lengths).to(device)

        t_en = model.text_encoder(tokens, input_lengths, text_mask)
        bert_dur = model.bert(tokens, attention_mask=(~text_mask).int())
        d_en = model.bert_encoder(bert_dur).transpose(-1, -2)

        ref_input_lengths = torch.LongTensor([ref_tokens.shape[-1]]).to(device)
        ref_text_mask = length_to_mask(ref_input_lengths).to(device)
        ref_bert_dur = model.bert(ref_tokens, attention_mask=(~ref_text_mask).int())
        s_pred = sampler(noise = torch.randn((1, 256)).unsqueeze(1).to(device),
                                          embedding=bert_dur,
                                          embedding_scale=embedding_scale,
                                            features=ref_s, # reference from the same speaker as the embedding
                                             num_steps=diffusion_steps).squeeze(1)


        s = s_pred[:, 128:]
        ref = s_pred[:, :128]

        ref = alpha * ref + (1 - alpha)  * ref_s[:, :128]
        s = beta * s + (1 - beta)  * ref_s[:, 128:]

        d = model.predictor.text_encoder(d_en,
                                         s, input_lengths, text_mask)

        x, _ = model.predictor.lstm(d)
        duration = model.predictor.duration_proj(x)

        duration = torch.sigmoid(duration).sum(axis=-1)
        pred_dur = torch.round(duration.squeeze()).clamp(min=1)


        pred_aln_trg = torch.zeros(input_lengths, int(pred_dur.sum().data))
        c_frame = 0
        for i in range(pred_aln_trg.size(0)):
            pred_aln_trg[i, c_frame:c_frame + int(pred_dur[i].data)] = 1
            c_frame += int(pred_dur[i].data)

        # encode prosody
        en = (d.transpose(-1, -2) @ pred_aln_trg.unsqueeze(0).to(device))
        if model_params.decoder.type == "hifigan":
            asr_new = torch.zeros_like(en)
            asr_new[:, :, 0] = en[:, :, 0]
            asr_new[:, :, 1:] = en[:, :, 0:-1]
            en = asr_new

        F0_pred, N_pred = model.predictor.F0Ntrain(en, s)

        asr = (t_en @ pred_aln_trg.unsqueeze(0).to(device))
        if model_params.decoder.type == "hifigan":
```

```
            asr_new = torch.zeros_like(asr)
            asr_new[:, :, 0] = asr[:, :, 0]
            asr_new[:, :, 1:] = asr[:, :, 0:-1]
            asr = asr_new

        out = model.decoder(asr,
                                F0_pred, N_pred, ref.squeeze().unsqueeze(0))


    return out.squeeze().cpu().numpy()[..., :-50] # weird pulse at the end of the model, need to be fixed later
```

Synthesize speech

Basic synthesis (5 diffusion steps, seen speakers)

```
text = "'' Kumbh Mela is an important Hindu pilgrimage, celeb
```

text: " " Kumbh Mela is an important Hindu pilgrimage, c "

```
reference_dicts = {}
reference_dicts['696_92939'] = "Demo/reference_audio/696_92939_000016_000006.wav"
#reference_dicts['1789_142896'] = "Demo/reference_audio/1789_142896_000022_000005.wav"
reference_dicts['1789_142896'] = "Demo/reference_audio/reference_audio/audio_16000Hz.wav"


noise = torch.randn(1,1,256).to(device)
for k, path in reference_dicts.items():
    ref_s = compute_style(path)
    start = time.time()
    wav = inference(text, ref_s, alpha=0.3, beta=0.7, diffusion_steps=5, embedding_scale=1)
    rtf = (time.time() - start) / (len(wav) / 24000)
    print(f"RTF = {rtf:5f}")
    import IPython.display as ipd
    print(k + ' Synthesized:')
    display(ipd.Audio(wav, rate=24000, normalize=False))
    print('Reference:')
    display(ipd.Audio(path, rate=24000, normalize=False))
```

```
WARNING:phonemizer:words count mismatch on 200.0% of the lines (2/1)
RTF = 0.069948
696_92939 Synthesized:
```

0:09 / 0:09

```
Reference:
```

0:03 / 0:03

```
WARNING:phonemizer:words count mismatch on 200.0% of the lines (2/1)
RTF = 0.059479
1789_142896 Synthesized:
```

0:11 / 0:11

```
Reference:
```

0:10 / 0:21

Basic synthesis (5 diffusion steps, unseen speakers)

The following samples are to reproduce samples in [Section 4](#) of the demo page. All spsakers are unseen during training. You can compare the generated samples to popular zero-shot TTS models like Vall-E and NaturalSpeech 2.

```
reference_dicts = {}
# format: (path, text)
reference_dicts['1221-135767'] = ("Demo/reference_audio/1221-135767-0014.wav", "Yea, his honourable worship is within, but h
reference_dicts['5639-40744'] = ("Demo/reference_audio/5639-40744-0020.wav", "Thus did this humane and right minded father c
reference_dicts['908-157963'] = ("Demo/reference_audio/908-157963-0027.wav", "And lay me down in my cold bed and leave my sh
reference_dicts['4077-13754'] = ("Demo/reference_audio/4077-13754-0000.wav", "The army found the people in poverty and left
reference_dicts['1789_142896'] = ("Demo/reference_audio/reference_audio/audio_16000Hz.wav", "Kumbh Mela is an important Hinc


noise = torch.randn(1,1,256).to(device)
for k, v in reference_dicts.items():
    path, text = v
    ref_s = compute_style(path)
```

```
    start = time.time()
    wav = inference(text, ref_s, alpha=0.3, beta=0.7, diffusion_steps=5, embedding_scale=1)
    rtf = (time.time() - start) / (len(wav) / 24000)
    print(f"RTF = {rtf:5f}")
    import IPython.display as ipd
    print(k + ' Synthesized: ' + text)
    display(ipd.Audio(wav, rate=24000, normalize=False))
    print(k + ' Reference:')
    display(ipd.Audio(path, rate=24000, normalize=False))
```

```
RTF = 0.075998
1221-135767 Synthesized: Yea, his honourable worship is within, but he hath a godly minister or two with him, and likewi
```

0:00 / 0:07

```
1221-135767 Reference:
```

0:00 / 0:03

```
RTF = 0.066617
5639-40744 Synthesized: Thus did this humane and right minded father comfort his unhappy daughter, and her mother embrac
```

0:00 / 0:08

```
5639-40744 Reference:
```

0:00 / 0:03

```
RTF = 0.086395
908-157963 Synthesized: And lay me down in my cold bed and leave my shining lot.
```

0:00 / 0:04

```
908-157963 Reference:
```

0:00 / 0:03

```
RTF = 0.077657
4077-13754 Synthesized: The army found the people in poverty and left them in comparative wealth.
```

0:00 / 0:04

```
4077-13754 Reference:
```

0:00 / 0:03

```
WARNING:phonemizer:words count mismatch on 100.0% of the lines (1/1)
RTF = 0.061675
1789_142896 Synthesized: Kumbh Mela is an important Hindu pilgrimage, celebrated approximately every 6 or 12 years, corr
```

0:11 / 0:11

```
1789_142896 Reference:
```

0:19 / 0:21

Speech expressiveness

The following section recreates the samples shown in [Section 6](#) of the demo page. The speaker reference used is `1221-135767-0014.wav`, which is unseen during training.

With `embedding_scale=1`

This is the classifier-free guidance scale. The higher the scale, the more conditional the style is to the input text and hence more emotional.

```
ref_s = compute_style("Demo/reference_audio/1221-135767-0014.wav")
```

```
texts = {}
texts['Happy'] = "We are happy to invite you to join us on a journey to the past, where we will visit the most amazing monum
texts['Sad'] = "I am sorry to say that we have suffered a severe setback in our efforts to restore prosperity and confidence
texts['Angry'] = "The field of astronomy is a joke! Its theories are based on flawed observations and biased interpretations
texts['Surprised'] = "I can't believe it! You mean to tell me that you have discovered a new species of bacteria in this pon

for k,v in texts.items():
    wav = inference(v, ref_s, diffusion_steps=10, alpha=0.3, beta=0.7, embedding_scale=1)
    print(k + ": ")
    display(ipd.Audio(wav, rate=24000, normalize=False))
```

```
Happy:
```

0:01 / 0:08

```
WARNING:phonemizer:words count mismatch on 100.0% of the lines (1/1)
Sad:
```

0:06 / 0:06

```
Angry:
```

0:03 / 0:07

```
Surprised:
```

0:06 / 0:06

With embedding_scale=2

```
texts = {}
texts['Happy'] = "We are happy to invite you to join us on a journey to the past, where we will visit the most amazing monum
texts['Sad'] = "I am sorry to say that we have suffered a severe setback in our efforts to restore prosperity and confidence
texts['Angry'] = "The field of astronomy is a joke! Its theories are based on flawed observations and biased interpretations
texts['Surprised'] = "I can't believe it! You mean to tell me that you have discovered a new species of bacteria in this por

for k,v in texts.items():
    noise = torch.randn(1,1,256).to(device)
    wav = inference(v, ref_s, diffusion_steps=10, alpha=0.3, beta=0.7, embedding_scale=2)
    print(k + ": ")
    display(ipd.Audio(wav, rate=24000, normalize=False))
```

```
Happy:
```

0:00 / 0:08

```
WARNING:phonemizer:words count mismatch on 100.0% of the lines (1/1)
Sad:
```

0:00 / 0:06

```
Angry:
```

0:00 / 0:08

```
Surprised:
```

0:00 / 0:06

Longform Narration

This section includes basic implementation of Algorithm 1 in the paper for consistent longform audio generation. The example passage is taken from Section 5 of the demo page.

passage: " "Kumbh Mela is an important Hindu pilgrimage, celebrated approximately every 6 or 12 years, correlated with the partial or fu "

Show code

```
# seen speaker
path = "Demo/reference_audio/696_92939_000016_000006.wav"
s_ref = compute_style(path)
sentences = passage.split('.') # simple split by comma
wavs = []
s_prev = None
for text in sentences:
    if text.strip() == "": continue
    text += '.' # add it back

    wav, s_prev = LFinference(text,
                              s_prev,
                              s_ref,
                              alpha = 0.3,
                              beta = 0.9,  # make it more suitable for the text
                              t = 0.7,
                              diffusion_steps=10, embedding_scale=1.5)
```

```
    wavs.append(wav)
print('Synthesized: ')
display(ipd.Audio(np.concatenate(wavs), rate=24000, normalize=False))
print('Reference: ')
display(ipd.Audio(path, rate=24000, normalize=False))
```

```
WARNING:phonemizer:words count mismatch on 100.0% of the lines (1/1)
WARNING:phonemizer:words count mismatch on 100.0% of the lines (1/1)
ðɪs f'ɛstɪvəl ɪz h'ɛld æt ðə m'æhem,æhæm t'æŋk (n,ɪɹ k'ævɚɹi ɹ'ɪvɚ) 'ɛvɹi tw'ɛlv j'ɪɹz æt k'ʌmbek,ɑːnɑːm, ɐtɹ'ækts m'ɪli
ðɪs f'ɛstɪvəl ɪz h'ɛld æt ðə m'æhem,æhæm t'æŋk (n,ɪɹ k'ævɚɹi ɹ'ɪvɚ) 'ɛvɹi tw'ɛlv j'ɪɹz æt k'ʌmbek,ɑːnɑːm, ɐtɹ'ækts m'ɪli
Synthesized:
```

0:38 / 0:38

```
Reference:
```

0:01 / 0:03

Style Transfer

The following section demostrates the style transfer capacity for unseen speakers in Section 6 of the demo page. For this, we set `alpha=0.5, beta = 0.9` for the most pronounced effects (mostly using the sampled style).

```
# reference texts to sample styles

ref_texts = {}
ref_texts['Happy'] = "We are happy to invite you to join us on a journey to the past, where we will visit the most amazing m
ref_texts['Sad'] = "I am sorry to say that we have suffered a severe setback in our efforts to restore prosperity and confic
ref_texts['Angry'] = "The field of astronomy is a joke! Its theories are based on flawed observations and biased interpretat
ref_texts['Surprised'] = "I can't believe it! You mean to tell me that you have discovered a new species of bacteria in this


path = "Demo/reference_audio/1221-135767-0014.wav"
s_ref = compute_style(path)

text = "Yea, his honourable worship is within, but he hath a godly minister or two with him, and likewise a leech."
for k,v in ref_texts.items():
    wav = STinference(text, s_ref, v, diffusion_steps=10, alpha=0.5, beta=0.9, embedding_scale=1.5)
    print(k + ": ")
    display(ipd.Audio(wav, rate=24000, normalize=False))
```

```
Happy:
```

0:00 / 0:08

```
WARNING:phonemizer:words count mismatch on 100.0% of the lines (1/1)
Sad:
```

0:00 / 0:08

```
Angry:
```

0:00 / 0:07

```
Surprised:
```

0:00 / 0:07

Extra fun!

You can record your own voice and clone it using pre-trained StyleTTS 2 model here.

Run the following cell to record your voice for 5 seconds. Please keep speaking to have the best effect.

```
# all imports
from IPython.display import Javascript
from google.colab import output
from base64 import b64decode

RECORD = """
const sleep  = time => new Promise(resolve => setTimeout(resolve, time))
const b2text = blob => new Promise(resolve => {
  const reader = new FileReader()
  reader.onloadend = e => resolve(e.srcElement.result)
```

```
  reader.readAsDataURL(blob)
})
var record = time => new Promise(async resolve => {
  stream = await navigator.mediaDevices.getUserMedia({ audio: true })
  recorder = new MediaRecorder(stream)
  chunks = []
  recorder.ondataavailable = e => chunks.push(e.data)
  recorder.start()
  await sleep(time)
  recorder.onstop = async ()=>{
    blob = new Blob(chunks)
    text = await b2text(blob)
    resolve(text)
  }
  recorder.stop()
})
"""

def record(sec=3):
  display(Javascript(RECORD))
  s = output.eval_js('record(%d)' % (sec*1000))
  b = b64decode(s.split(',')[1])
  with open('audio.wav','wb') as f:
    f.write(b)
  return 'audio.wav'  # or webm ?
```

Please run this cell and speak:

```
print('Speak now for 10 seconds.')
audio = record(sec=10)
import IPython.display as ipd
display(ipd.Audio(audio, rate=24000, normalize=False))
```

```
Speak now for 10 seconds.
```

0:09 / 0:09

Synthesize in your own voice

text: " " text to speech model that leverages style diffusion and adversarial training with large speech language models to achieve huma "

Show code

```
reference_dicts = {}
reference_dicts['You'] = audio


start = time.time()
noise = torch.randn(1,1,256).to(device)
for k, path in reference_dicts.items():
    ref_s = compute_style(path)
    wav = inference(text, ref_s, alpha=0.1, beta=0.5, diffusion_steps=10, embedding_scale=2)
    rtf = (time.time() - start) / (len(wav) / 24000)
    print('Speaker: ' + k)
    import IPython.display as ipd
    print('Synthesized:')
    display(ipd.Audio(wav, rate=24000, normalize=False))
    print('Reference:')
    display(ipd.Audio(path, rate=24000, normalize=False))
```

```
<ipython-input-6-e2121fab3e2d>:20: UserWarning: PySoundFile failed. Trying audioread instead.
  wave, sr = librosa.load(path, sr=24000)
WARNING:phonemizer:words count mismatch on 200.0% of the lines (2/1)
Speaker: You
Synthesized:
```

0:11 / 0:11

```
Reference:
```

0:09 / 0:09