華中科技大學

数字语音处理

课程设计报告

成　　员　李顺祺（U202214132）

郑　哲（U202213953）

陈　彬（U202213920）

专　　业　电子信息工程

任 课 教 师　闫增强

院 （系、所）　电子信息与通信

基于 Tacotron2 的孤立词英文数字 TTS 语音合成

成员分工

李顺祺：主要负责数据集收集与预处理，模型的后续参数优化，实验结果的分析与优化。

郑哲：主要负责远程服务器端的搭建、服务器环境配置，以及调试和运行服务器端的程序

陈彬：主要负责学习 TTS 模型原理并编写和调试模型训练程序、语音合成程序搭建， 实验报告撰写

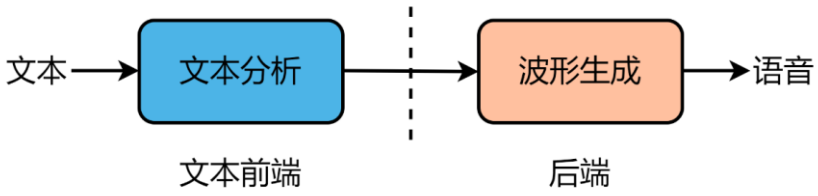
实验背景及目的

1.实验背景

语音合成是语音处理的一个重要组成部分，语音合成技术赋予机器说话的功能，解决了让机器像人一样说话的问题。

按照人类言语的不同层次，语音合成可以分为三类层次：（1）按规则从文字到语音的合成（TTS）；（2）按规则从概念到语音的合成（CTS）；（3）按规则从意向到语音的合成（ITS）。

受限于各种因素，目前阶段主要发展的是 TTS 技术，其结构如图所示：



TTS 系统中的波形合成模块一般采用波形拼接合成的方法，如基音同步叠加法 PSOLA。本实验进行只进行孤立词的合成，采用端到端的语音合成方法，不需要进行语音拼接。

2.实验目的

1). 基于孤立词数据集进行语音合成模型的训练，了解语音合成的基本流程和

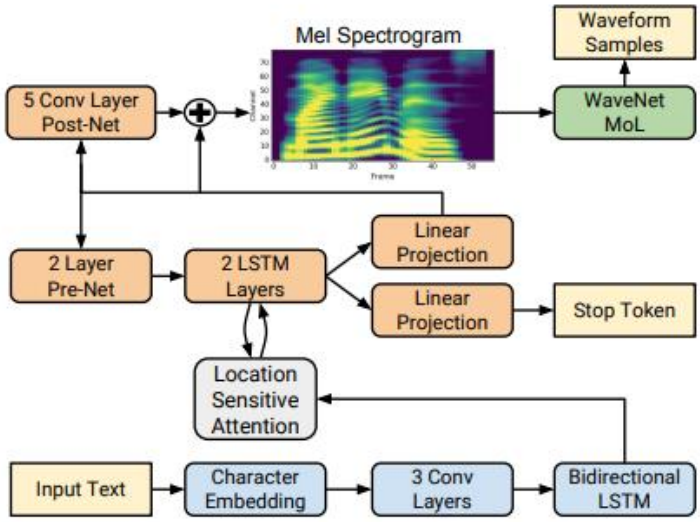
原理；

2). 探究 Mel 频谱等特征参量在语音合成中的重要作用，如停止符预测、帧分析等；

3). 探究不同发音类型对语音合成质量的影响。

实验原理

一.Tacotron2 模型



1. 文本编码（Encoder）

1). 输入层

- **Input Text**：原始文本（如"Hello"）输入。
- **Character Embedding**：字符级嵌入层，将文本转为稠密向量表示。

2). 特征提取

- **3 Conv Layers**：3 层卷积网络，提取字符的局部上下文特征（如字母组合规律）。
- **Bidirectional LSTM**：双向 LSTM 捕获前后文依赖，输出**文本编码序列**（Encoder Hidden States）。

---

2. 注意力机制（Location-Sensitive Attention）

- **作用**：动态对齐文本序列与声学特征序列的时间步。
- **输入**：

a.编码器的隐藏状态（文本特征）。

b.解码器上一时间步的状态（声学特征上下文）。

- **输出**：注意力权重（决定当前帧应关注哪些文本部分）。

---

3. 频谱解码（Decoder）

- **输入预处理（Pre-Net）**

  2 Layer Pre-Net：两层全连接网络（+Dropout），对上一帧梅尔频谱做非线性变换，作为解码器输入。
- **循环解码**

  2 LSTM Layers：两层 LSTM 逐步生成当前帧的隐藏状态。

  Location-Sensitive Attention：结合注意力上下文，更新当前帧状态。
- **输出预测**

  Linear Projection（×2）：预测当前帧的梅尔频谱（Mel-Spectrogram）；预测停止符（Stop Token，判断是否结束合成）。

---

4. 后处理与波形生成

- **频谱优化（Post-Net）**

5 Conv Layer Post-Net：5 层卷积网络，对预测的梅尔频谱进行残差修正，提升细节质量。

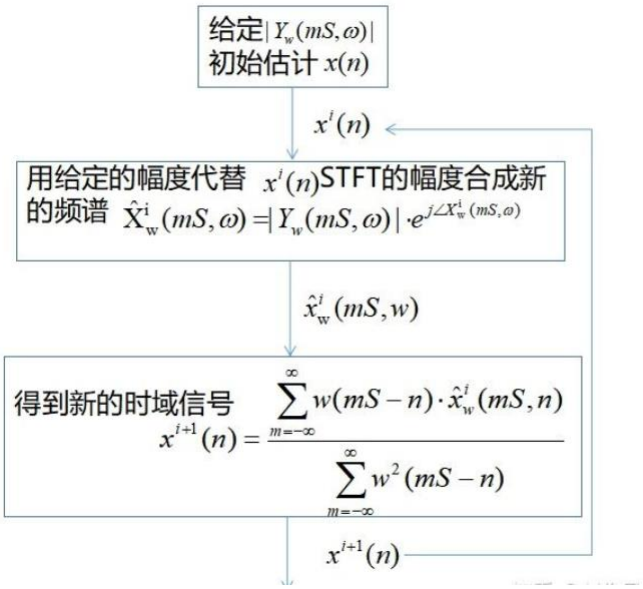
- **波形合成（Vocoder）**

WaveNet Vocoder：将优化后的梅尔频谱转换为波形样本（Waveform Samples）。

二.Vocoder(以 Griffin-lim 为例)

tacotron2 模型输出得到是语音信号的特征频谱，需要使用声码器将这些频谱合成为对应的语音波形。

griffin-lim 算法



1). 算法思想：

griffin-lim 重建语音信号需要使用到幅度谱和相位谱。而 tacotron2 模型输出的 MEL 谱当中是不含相位信息的，griffin-lim 在重建语音博形的时候只有 MEL 谱可以利用。但是通过一些运算，我们可以利用帧与帧之间的关系估计出相位信息，从而重建语音波形。

这里的 MEL 谱可以看做是实部，而相位信息可以看做是虚部，通过对实部和虚部的运算，得到最终的结果。

2). 算法步骤：

它是一个迭代算法，迭代过程如下：

1. 先随机初始化一个相位谱；
2. 用相位谱和已知的幅度谱经过 ISTFT 合成新语音；
3. 对合成的语音做 STFT，得到新的幅度谱和相位谱；
4. 丢弃新的幅度谱，用相位谱和已知的幅度谱合成语音，如此重复。

3). 算法解释：

创建一个复数矩阵，将已知的幅度谱作为实层，用噪声随机初始化虚层。（至此，已有振幅信息，但是没有相位信息）。对复数矩阵进行 ISTFT。（至此，仅依靠幅度谱得到一个初步的时域信号）。

对得到的时域信号进行 STFT，得到一个复数矩阵。（获取小部分的，不准确的相位信息）。复数矩阵是从一不准确的时域信号，得到了一个振幅与相位。用已知的振幅替换该矩阵的实部。（至此，有了已知振幅和初步准确的相位）

重复直至迭代至一个满意的效果或者迭代次数到达指定的上限。

三.LJSpeech DataSet

LJSpeech 是一个免费的数据集，由 13,100 个单一发声者朗读非小说书籍片段的短音频剪辑组成。每个音频剪辑都附带了相应的转录文本，音频长度从 1 秒到 10 秒不等，总时长约为 24 小时。

数据集结构如下：

```
LJSpeech/
├── metadata.csv          # 核心文本-音频对齐文件
├── wavs/                 # 存放所有音频文件（WAV 格式）
│   ├── LJ001-0001.wav    # 音频文件命名格式：LJXXX-XXXX.wav
│   ├── LJ001-0002.wav    # 每个文件对应一段语音（约 1-10 秒）
│   └── ...               # 共 13,100 个音频文件（默认）
```

metadata.csv 结构如下：

```
LJ001-0001|Printing, in the only sense with which we are at present
concerned,...|Printing in the only sense with which we are at present
concerned ...
LJ001-0002|disagrees with the pronunciation of these words in the
English of the Elizabethan era.|disagrees with the pronunciation of
these words in the English of the Elizabethan era.
```

其中 metadata.csv 是进行模型训练的核心文件，用于因素和发音单元的对齐。

本实验仅进行孤立词语音合成，不必要使用这样完备的数据集，仅参照其结构和使用发放处理自己的小数据集。

实验方法及步骤

1.数据集的初步处理

（1）数据增强

实验课上提供的 Numbers 数据集包含 zero-nine10 个数字的各 20 各音频文件，对于 tacotron2 模型，这样的数据集过小，容易导致模型不拟合与学习不充分。因此需要对数据进行增强处理。

```
y_stretch = librosa.effects.time_stretch(y, rate=1.1)# 时间拉伸
augmented.append(y_stretch)


y_pitch = librosa.effects.pitch_shift(y, sr=sr, n_steps=2) # 音调平移
augmented.append(y_pitch)

y_vol = y * np.random.uniform(0.8, 1.2)# 音量扰动
augmented.append(y_vol)
```

通过添加扰动、时间拉伸等处理，将数据集扩充到原来的三倍，在一定程度上解决了数据集过小的问题。

（2）生成 metadata.csv 文件

仿照 LJSPeech 文件的结构对 Numbers 数据集进行变构后，就可以生成 csv 文件用于模型训练了。由于音频文件较为规律，因此可以编程生成，程序和代码如下：

```
for filename in os.listdir(wavs_dir):
    if filename.endswith(".wav"):
        # 提取数字部分
        digit_part = filename.split('_')[0][5]  # 提取文件名中的数字部分，文件名格式为 "digitX_"
        digit = int(digit_part)  # 获取数字

        # 根据数字获取对应的英文单词
        text = digit_to_word[digit]

        # 移除 .wav 后缀
        filename_without_ext = os.path.splitext(filename)[0]

        # 创建一行数据，格式为：文件名(无后缀) | 对应的英文数字
        metadata.append([filename_without_ext, text])
```

程序遍历了 Numbers 数据集中的每一个文件，通过文件名提取音频对应的文本文件，最后生成的 metadata.csv 文件如图：

```
digit0_10_0|zero
digit0_11_0|zero
digit0_12_0|zero
digit0_13_0|zero
digit0_14_0|zero
digit0_15_0|zero
digit0_16_0|zero
......
```

（3）其它数据集处理

由于模型训练时使用的随机种子不同，在训练得到的模型对不同数字的泛化程度不同，有时甚至会出现某一两个合成音频全为噪声的情况（此时一般过拟合）。发生这种情况时，需要再引入额外数据集对现有模型进行增强训练。

为保证前后一致性，新引入的模型也应为孤立数字英文发音。这里使用 Google Speech Commands Dataset。编程提取其中的数字音频文件，并进行上述处理。

2.模型训练与调试

Coqui TTS 是一个包含了多种 TTS 的 git 库，为训练 tacotron2 模型提供了完善 的框架与函数调用。因此本次实验以 Coqui TTS 为基础编写程序进行模型训练。

（1）数据初始化

使用 BaseDatasetConfig 规定数据集格式（采用 LJSpeech 格式）：

```
dataset_config = BaseDatasetConfig(
    formatter="ljspeech",
    meta_file_train="metadata.csv",
    path=data_path,
)
```

之后程序将按照 LJSpeech 格式处理读取的数据集及 csv 文件。

使用 dataset.py 中的 load_tts_samples()读取数据，并根据需要划分训练集和测试集：

```
train_samples, eval_samples = load_tts_samples(
    dataset_config, #数据集的基本配置，包含取样率、频谱维数等
    eval_split=True,
    eval_split_max_size=config.eval_split_max_size,
    eval_split_size=config.eval_split_size,
)
```

（2）文本处理

读取 csv 文件后，需要对其中的进行处理，方便模型在后续的 TTS 中理解输入文字。使用 tokenizer.py 函数处理输入文本：将文本按照分隔符分割为不同部分。由于本实验仅进行孤立词合成，故不需要将文本划分为音素进行处理。

```
def intersperse_blank_char(self, char_sequence: List[str],
                                            use_blank_char: bool = False):
    """Intersperses the blank character between characters in a
sequence.

    Use the ```blank``` character if defined else use the ```pad```
character.
     """
    char_to_use = self.characters.blank_id if use_blank_char else
self.characters.pad
    result = [char_to_use] * (len(char_sequence) * 2 + 1)
    result[1::2] = char_sequence
    return result
```

（3）音频处理

音频预处理

读入数据集的音频文件，并提取 mel 频谱

```
def melspectrogram(self, y: np.ndarray) -> np.ndarray:
    if self.preemphasis != 0:
        y = self.apply_preemphasis(y)  # 应用预加重，self.preemphasis 通常
取值 0.97
    D = stft(
        y=y,
        fft_size=self.fft_size,       # FFT 窗口大小（如 2048）
        hop_length=self.hop_length,   # 帧移（如 512，决定时间轴分辨率）
        win_length=self.win_length,   # 窗口长度（如 1024）
        pad_mode=self.stft_pad_mode,  # 边缘填充方式（如'reflect'）
    )
    S = spec_to_mel(
        spec=np.abs(D),              # 取 STFT 幅度谱（丢弃相位信息）
        mel_basis=self.mel_basis     # 预计算的梅尔滤波器组矩阵
    )
    if self.do_amp_to_db_mel:
        S = amp_to_db(
            x=S,
            gain=self.spec_gain,      # 增益系数（如 1.0）
            base=self.base           # 对数底数（如 10 或 np.e）
        )

    return self.normalize(S).astype(np.float32)
```

- **预加重 (Pre-emphasis)**
  目的：补偿语音信号高频分量的衰减（发声时高频能量通常较弱）；
  典型值：self.preemphasis = 0.97；
- **STFT 参数**
  fft_size：决定频率分辨率（值越大频率划分越细）；
  hop_length：决定时间分辨率（值越小时间轴越密集）；
  win_length：通常为 fft_size 的 1/2~1/4，控制频谱平滑度；
- **梅尔滤波器组 (Mel Basis)**
  通过 self.mel_basis 将线性 Hz 频率转换为梅尔刻度，更接近人耳听觉特性；
- **归一化 (Normalization)**
  方法包括：减均值除方差、缩放到[-1,1]等。

音频后处理（与预处理对称）

在语音合成阶段，使用 Griffin-lim 算法，将 mel 频谱反变换为波形谱

```
def inv_melspectrogram(self, mel_spectrogram: np.ndarray) ->
                                                        np.ndarray:
```

```
    D = self.denormalize(mel_spectrogram)  # 例如从[-1,1]映射回原始分贝值
    S = db_to_amp(
        x=D,
        gain=self.spec_gain,  # 需与 amp_to_db 时的增益一致（如 1.0）
        base=self.base        # 对数底数（如 10 或 np.e）
    )
    S = mel_to_spec(
        mel=S,
        mel_basis=self.mel_basis  # 必须与正向变换使用相同的梅尔滤波器组
    )  # 输出 shape=(fft_size//2 + 1, n_frames)
    W = self._griffin_lim(
        S**self.power  # 功率谱补偿（self.power 通常为 1.2~1.5，增强高频）
    )  # 通过迭代估计相位信息，输出时域波形

    return self.apply_inv_preemphasis(W) if self.preemphasis != 0 else W
```

观察可知，预处理和后处理使用的 mel 滤波器组需完全对称，但是对于过完备的 mel 滤波器组，只能使用近似方法还原，会损失一部分的高频信息。

- **Griffin-Lim 算法核心**

```
def _griffin_lim(self, S: np.ndarray):
    for _ in range(self.griffin_lim_iters):  # 典型迭代 50~100 次
        waveform = istft(S * np.exp(1j * phase), ...) # 用当前相位重建波形
        _, phase = stft(waveform, ...)  # 从波形中提取新相位
    return waveform
```

- **功率谱补偿**

通过指数放大高频分量（因梅尔谱压缩导致高频信息衰减），典型值 power=1.2。

- **反预加重**

逆转预加重滤波器的效果。

（4）模型处理

完成上述准备工作后，即可使用 tacotron2 进行模型训练。

Tacotron2 模型包含的结构如下：

模块	参数类型	作用
Embedding	嵌入矩阵	将字符转为向量
Encoder	卷积权重、LSTM 参数	提取文本特征并建模语义上下文
Attention	线性层权重、卷积核	计算编码器与解码器之间的对齐
Decoder	LSTM、前馈层、Stop token 参数	逐帧生成 mel 频谱
Postnet	卷积核、偏置、BatchNorm	对 mel 频谱进行增强修复，提升自然度

（5）程序调试

1). 初次运行程序训练模型，出现报错：

```
text = cols[2]
IndexError: list index out of range
```

经检查发现，LJSpeech 原文件中 metadata.csv 每行至多有三项，而使用的小数据集所生成的 csv 文件每行固定为两项。找到报错的库函数

TTS\tts\datasets\formatters.py

将其中的 text = col[2]修改为 text = col[1],即可解决问题。

2). 程序运行一段时间后，经常中断：

```
HTTPSConnectionPool(host='coqui.gateway.scarf.sh', port=443): Max
retries exceeded with url: /trainer/training_run (Caused by
SSLError(SSLEOFError(8, '[SSL: UNEXPECTED_EOF_WHILE_READING] EOF
occurred in violation of protocol (_ssl.c:1007)')))
```

经查阅得知，这是库中的模型函数回定期与服务器通信，当网络发生波动时可能发生通信中断，为保证程序稳定性，将这部分代码注释即可。

完成一系列调试后，程序已经可以在本地顺利运行。

3.远程服务器搭建

实验程序涉及到大量数据和卷积计算，对设备的计算能力有较高的要求。小组成员的设备均没有搭载 GPU，若使用 CPU 进行训练，需要大量的时间且占用设备。经讨论，决定租用远程服务器完成实验。

经过比较，根据价格和服务选择 AutoDL 平台上的 GPU 实例。

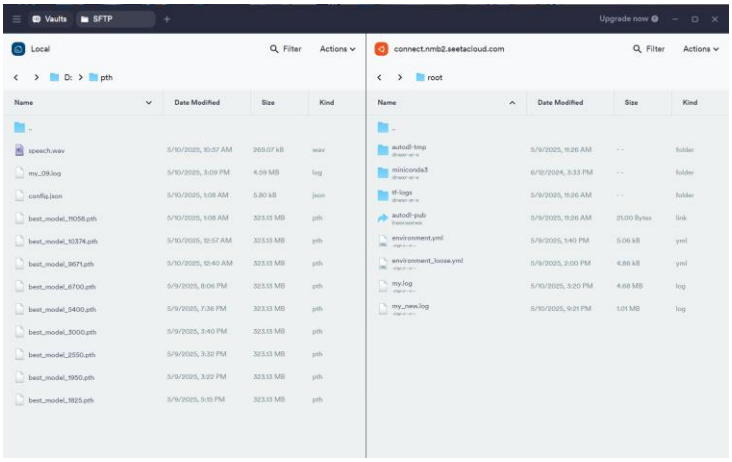
（1）服务器租用

由于训练任务不算复杂，经查阅资料后决定租用如下配置服务器：

租用成功后，得到实例：

（2）SSH 调试工具配置

使用 Termius 进行远程管理，按照教程输入服务器地址及访问密码后，可以进入如下界面：



此时可以将本地的文件上传到服务器端，也可以将服务器上得到的模型下载到本地。

（3）服务器端环境的配置

起初，尝试使用 Termius 将本地的 conda 环境文件夹传输到服务器端，但受限于带宽，传输速度不尽人意。因此，决定使用平台提供的 jupyter 直接在服务器端配置环境。

在本地当前激活的 conda 环境下运行如下命令：

```
conda env export > environment.yml
```

得到当前环境的配置文件。在本地打开服务器的 jupyter Lab，进入根 conda 环境，运行如下命令：

```
conda env create -f environment.yml -n new_env_name
```

即可在服务器端创建一个与本地相同的 conda 环境，将程序从本地上传到服务器后，即可正常运行。（**由于在本地调试和过程中对 TTS package 进行了修改，服务器端也要进行相同的修改**）

在初次创建环境时，出现如下错误：

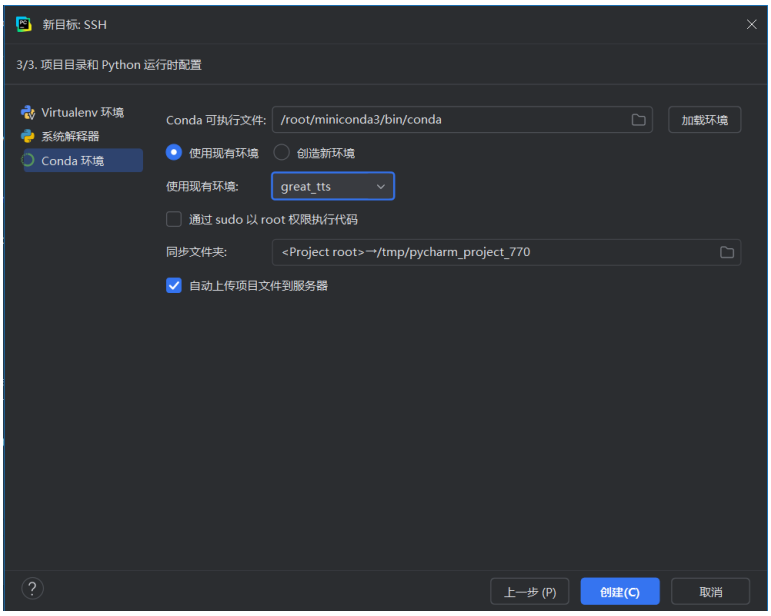
```
ResolvePackageNotFound:
  - vc=14.42
  - vs2015_runtime=14.42.34433
```

根本原因是服务器端使用 Linux 系统，而 Linux 系统中不需要使用错误信息中列出的 C++解释器，删去 yaml 文件中的对应行即可。

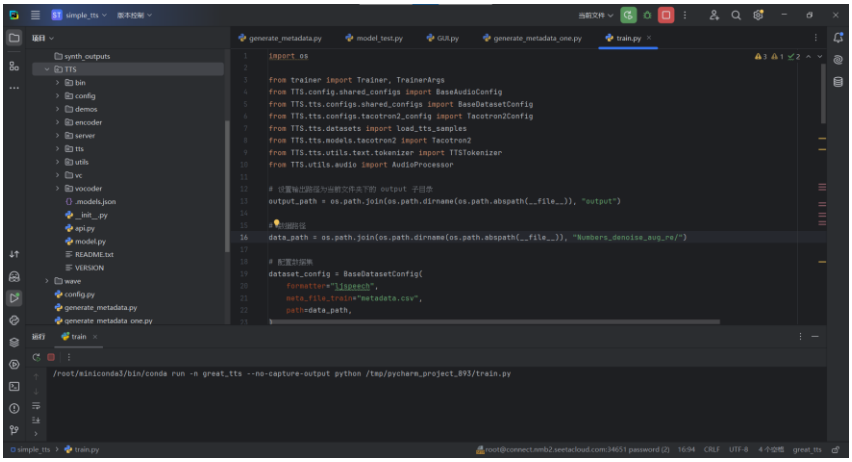
（4）pycharm 连接服务器端环境

由于项目文件中包含数据集，同样不适合用 Termius 传输，查阅资料，可以使用 pycharm 的 SSH 连接远程服务器同步本地项目到服务器。

选择服务器上配置好的 conda 环境，并创建同步项目：



创建成功后，pycharm 将自动将本地项目及更改实时同步到服务器端，实现在本地调试，服务器端运行。此时在 pycharm 内运行文件，实际执行计算等任务的是远程的服务器，pycharm 通过 SSH 连接获取服务器端的日志并输出到窗口：



（5）程序在服务器端的后台运行

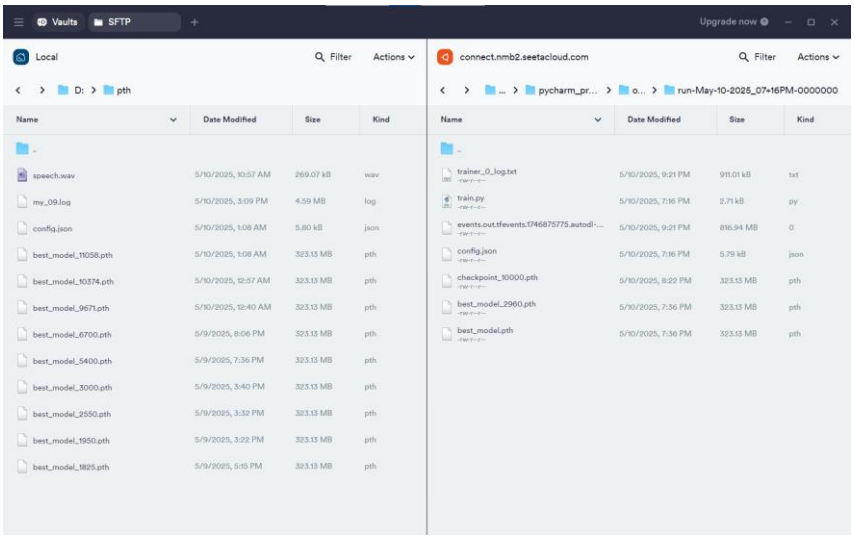
完成上述步骤后，可以在使用服务器端的 GPU 高速运行程序，但此时仍依赖于二者间的 SSH 连接，若出现网络波动导致连接中断，程序也可能终止运行。

为保证程序运行不受本地状况影响，在服务器端运行如下命令：

```
nohup python3 -u /tmp/pycharm_project_893/train.py >>/root/my_new.log
                                                    2>&1 &
```

这样，程序将在服务器端后台运行，不受本地与服务器连接的干扰。

运行完成后，可通过 Termius 将训练得到的模型下载到本地。



之后即可使用训练得到的模型进行语音合成。

4.简易语音合成系统的实现

（1）使用模型合成语音

使用训练得到的模型进行语音合成。在 conda 环境中输入 tts -h，得到模型的使用方法：

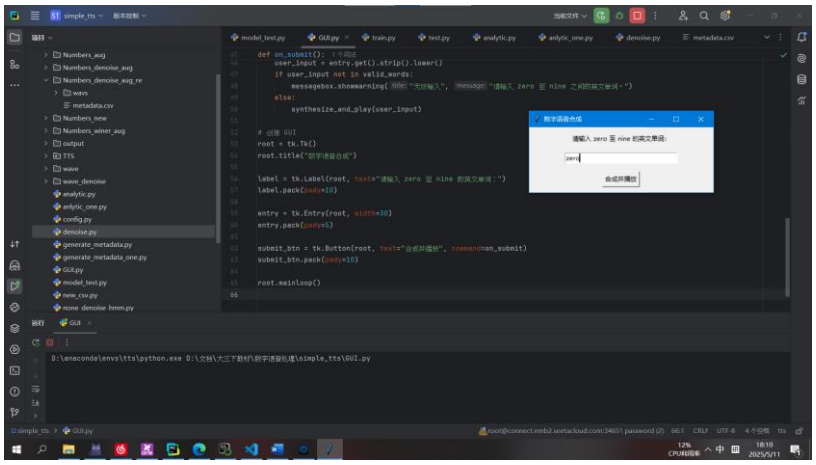
```
tts --text "Text for TTS" --model_path path/to/model.pth --config_path
                  path/to/config.json --out_path output/path/speech.wav
```

输入以上命令即可调用模型将 text 文本转换为语音并保存到特定路径。

通过编写程序，显示使用此命令，分别调用两个训练过程中产生的模型，一次性输出 zero-nine 的语音文件。并于 TTS 库中的预训练模型进行比较。

```
for i, text in enumerate(texts):
    output_path = os.path.join(model["output_dir"], f"{text}.wav")
    cmd = [
        "tts",
        "--text", text,
        "--model_path", model["model_path"],
        "--config_path", model["config_path"],
        "--out_path", output_path
    ]
```

（2）分别调用表现较好的模型，实现交互的孤立词语音合成

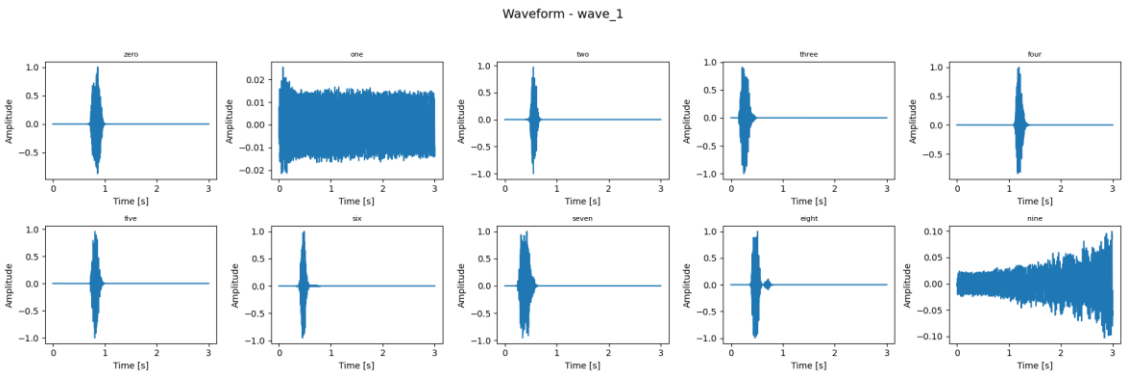


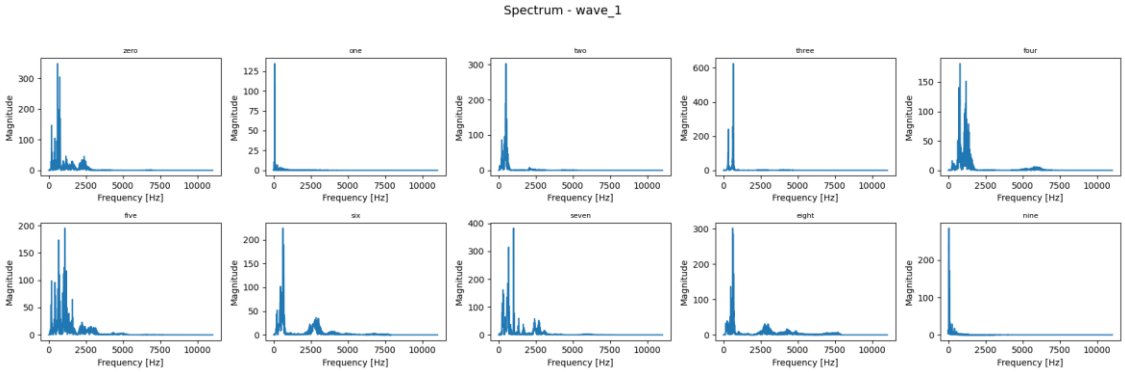
实验结果及分析

1.比较不同模型与预训练模型效果的差别

（1）模型 1

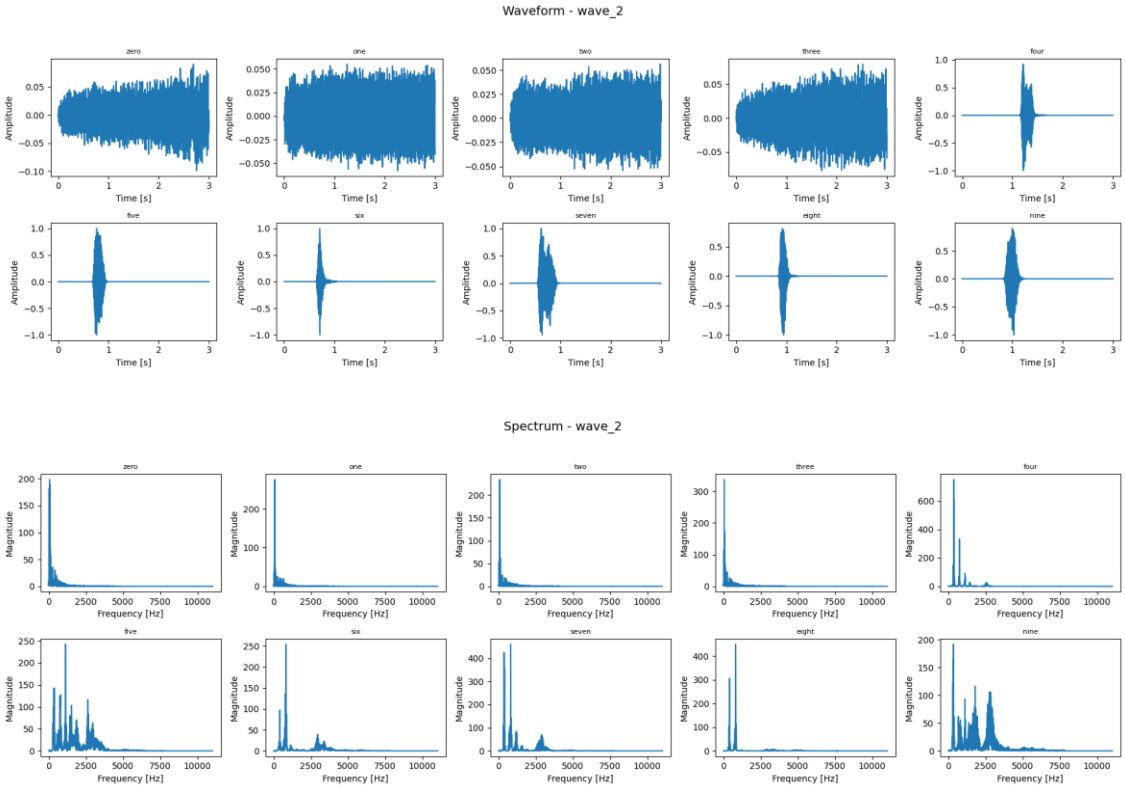
使用数据集经过 640 个 epoch 得到。





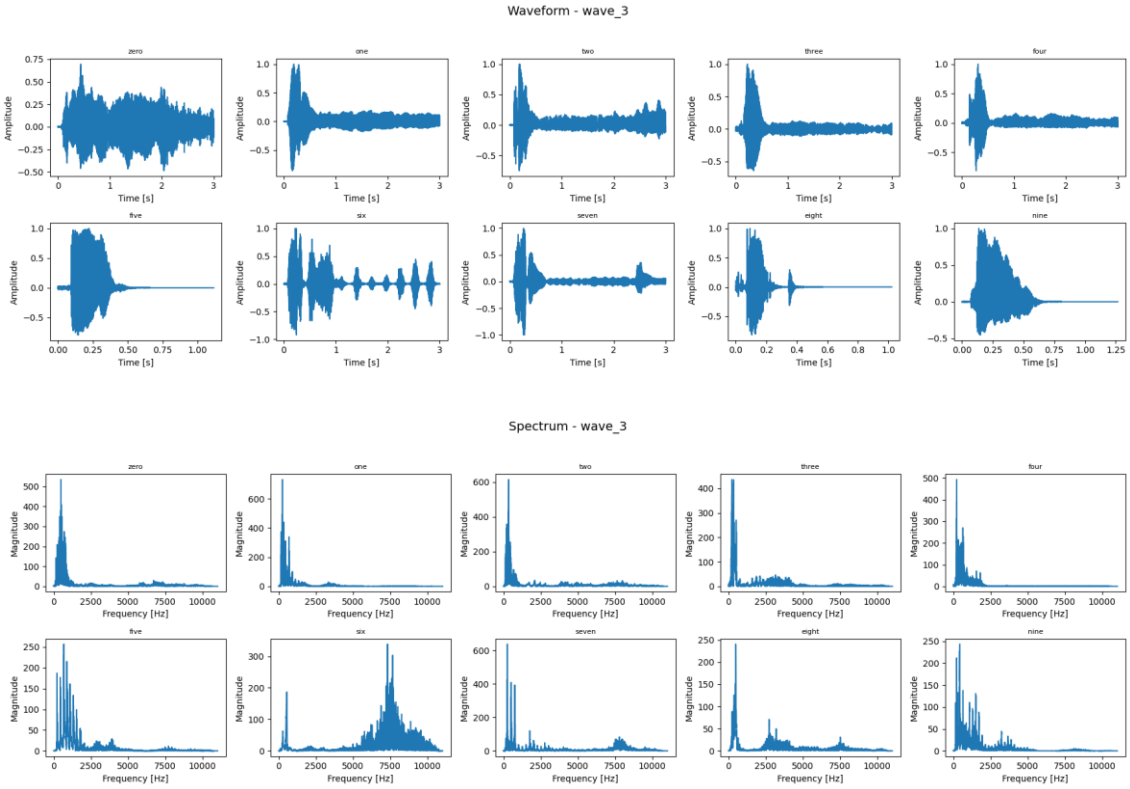
（2）模型 2

在训练模型 1 完成后继续训练产生的 checkpoint 文件。



（3）预训练模型

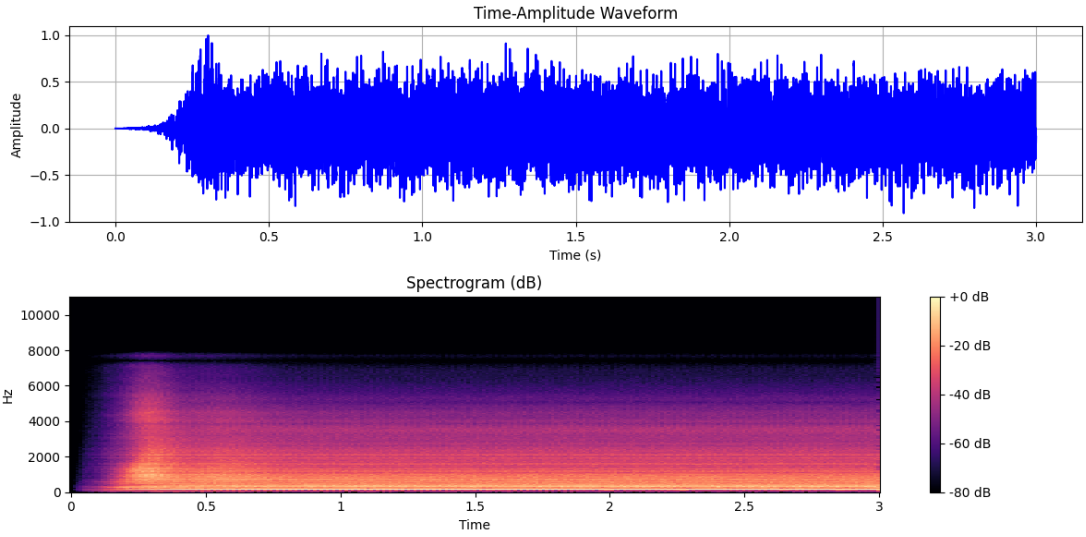
Coqui TTS 库提供的使用 LJSpeech 训练的完整模型。



（4）比较与排查

通过与预训练模型的对比，可以看出模型二虽然迭代次数要多于模型一，但语音合成的效果却不如模型一，推测是在训练过程中发生了过拟合。

两个模型对数字“one”的合成效果均不佳。为排除样本数引起的问题，使用额外数据集的大样本量重新进行 one 的模型训练，分析新模型合成的 one 的音频：



效果依然不佳，推测是 one 频谱特征较为不突出，不易学习。

查看日志文件，发现程序在运行到 epoch=190 时就已经拟合，故排除学习不充分的问题。

2.实验结论

Tacotron2 模型能够较好地学习语音的频谱特征。在样本足够大且丰富的情况下，合成的语音较为自然，语调真实。如 TTS 库中使用 LJSpeech 训练的预训练模型。但在小样本的情况下（如本实验使用的 Numbers 数据集），尤其是音频种类不够丰富的情况下，难以准确地学习语音的特征，会丢失大量的高频分量，导致语音短促而不自然。

3.收获

本次实验中，我们三名组员通过分工合作，解决了相当多在实验出现的问题。根据具体情况修改库函数、租用服务器解决算力问题、逐步排查语音合成效果不佳的原因等；提升了自身解决学习中实际问题的能力，也对不同的方向如机器学习在语音处理领域的应用等有了一些初步的了解，受益颇多。也再次感谢闫老师和助教老师在平时提供的指导和帮助，对我们更好地探索语音处理这一领域打造了一定基础。

参考资料

- 博客类（数量较多，不一一例举）
  1. *云服务器上运行 python 程序（PyCharm 本地编辑同步服务器+Anaconda）挂载跑实验详细教程*
  2. *服务器中安装和配置 Conda 环境*
  3. *Tacotron2 看这一篇*
  4. *AutoDL 帮助文档*
  5. *不同服务器之间迁移 conda 环境*

  ……

- Git 仓库
  1. ***coqui-ai/TTS: 🐸💬 - a deep learning toolkit for Text-to-Speech, battle-tested in research and production (github.com)***
  2. ***PaddlePaddle/PaddleSpeech: Easy-to-use Speech Toolkit including Self-Supervised Learning model, SOTA/Streaming ASR with punctuation, Streaming TTS with text frontend, Speaker***

Verification System, End-to-End Speech Translation and Keyword Spotting. Won NAACL2022 Best Demo Award. (github.com)

3. *eisneim/cytron_tts_gui: TTS(Text to speech) GUI using Baidu TTS api, currently only support Chinese; 将文字转换为语音 mp3 文件，自动拆分较长文本文件，适合用于生成有声小说（github.com)*

   ……

- 书籍文献
  1. *数字语音处理及 matlab 仿真*
  2. *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions*
  3.