简介

项目简介：一个使用网页交互使用 any-to-many 的语音转化模型的展示项目。

- 模型简介参见：https://github.com/thsno02/StarGANv2-VC/blob/main/README.md；
- 模型详情参见：https://github.com/yl4579/StarGANv2-VC 以及 https://arxiv.org/abs/2107.10394；
- 使用指南参见：https://github.com/thsno02/voice-conversion/blob/main/README.md

代码仓库：

- https://github.com/thsno02/StarGANv2-VC 该地址主要存放模型训练和调试相关的代码；
- https://github.com/thsno02/voice-conversion 该地址主要存放整个展示系统的代码。

本项目的任务管理基本使用 GitHub 完成，可参见：

- https://github.com/thsno02/StarGANv2-VC/projects/1 开始阶段的任务；
- https://github.com/thsno02/voice-conversion/projects/1 中后期的任务管理。

本项目遇到的问题基本都罗列在了 GitHub 中，可参见：

- https://github.com/thsno02/StarGANv2-VC/issues；
- https://github.com/thsno02/voice-conversion/issues。

本文访问地址：https://github.com/thsno02/StarGANv2-VC/discussions/47

工作步骤

工作逻辑可以简化为：明确问题 -> 模型选择 -> 数据准备 -> CycleGAN -> 数据准备 -> StarGANv2 和 CycleGAN -> 效果评估及模型选择：StarGANv2 -> 论文详读及代码修改 -> 第二版 StarGANv2 -> 实时推理 -> 展示系统。

时间线

该项目总计花费了六周，时间花费大致如下：

- 第一周进行问题初步的调研，获取少量数据，训练 CycleGAN，同时检索相关的文档和论文，找到并确定了 StarGANv2 的主力地位。接着看 StarGANv2 的 GitHub 和论文中，确定了 StarGANv2 的数据要求，根据数据要求获取了相关的数据，修改代码、调整配置文件后开始第一版少量数据的训练，验证模型效果。
- 第二周主要进行 StarGANv2 正式训练，推理代码的完善，以及确认第二版模型的效果。梳理并完善了批量推理代码后，对第一版模型的训练结果进行检验，确认模型的实际可行性。在沟通后，对数据进行了一些调整，重新确定了发音人的名单。优化代码和重新组织文件结构后，开始了第二版训练。训练期间对推理代码的逻辑和质量进行优化。
- 第三周主要在第二版模型训练结束后，挑选了表现较好的模型 checkpoints，准备实时推理和模型的部署。在 Windows 电脑中部署环境时出现了大量繁琐的问题，花了很多时间解决。逐步优化实时推理的代码，熟悉相关包的使用。进行大量的真人实测，从各个角度发现代码和模型的问题。
- 第四周主要是测试环境，写文档，开始系统整合。对各种可能会影响模型表现的参数都进行了测试和遴选，同时优化推理速度，写模型、训练、和推理相关的工程文档。再和计算部的同事协作沟通，确认了系统的架构和核心的输入逻辑。
- 第五周进行全面的系统整合、测试，写文档。解决了前后端连通中的各种小 Bug，以及工程上设定的错误。完成代码和文档的存档工作。
- 第六周重新进行设备的调试和展示页面的修改。解决了实时转化效果不如在云桌面中转化好的问题，优化代码

逻辑和前端页面，进一步完善文档工作。

明确问题和模型选择

根据展厅的需求，明确了任务是「语音转化（Voice Conversion, VC）」，即在保留语言信息的条件下转化说话人的身份。

在任务最开始的阶段，完成时间很紧张，做了两手准备，用深度学习模型做语音转化以及使用市面上通行的工具进行变声。语音转化的思路最开始是使用科大讯飞的 STT（Speech-to-text）工具将语音转化为文字，再使用成熟的 TTS（Text-to-speech） 工具用目标发音者的声音读出文字，以此完成声音的转化。但这个思路被否决了，因为使用文本标签的 TTS 无法模拟人真实说话的场景，比如语气连词、发声助词、说话停顿等。语音转化的任务暂明确为，使用端对端（end-to-end）的深度学习模型进行建模与训练。

因为 DDL 和当时的沟通场景局限，在找模型的阶段直接锁定了 CycleGAN 后便没有进行全面的模型检索和方法比较了。后来 DDL 放宽，根据检索到的论文信息，明确了所需模型的几大特征：无监督、使用非平行语料库、训练速度较快。前两个特征是因为数据集的限制，目前没有可用的数据集，需要自己构建数据集，而给数据打标签和让说话人读同一段话的成本过高；后者是也是因为 DDL 与 CycleGAN 的训练时间过长且难以加速，想要尽快找到模型并落地。在一系列检索后，找到了 StarGANv2，完美契合目前的需求，且模型是多对多（many-to-many）的，即可以将多个人的语音转化为不同的发音者，在展示上更具优势。在查看了作者的 GitHub、 demo 、和展示视频后，决定使用 StarGANv2 作为主力训练模型，替换之前的 CycleGAN 模型。因为该作者目前依然活跃该代码仓库中，Issues 中解决了大量实际工程会出现的问题，demo 的效果很惊艳。

CycleGAN

使用的代码仓库：https://github.com/jackaduma/CycleGAN-VC2

最早的使用的模型，CycleGAN 是一个无监督、非平行语料、一对一的语音转化模型。可能因为模型比较老以及数据集不规整，训练效果不尽人意，而且 CycleGAN 很难使用多 GPU 进行训练，训练的时间开销很大。

因为当时时间比较赶，代码比较可靠，没有遇到什么问题，所以没有对代码进行太多的修改。获取数据，添加日志，修改相关路径后，直接开始训练了。

数据获取

通过石总和李总的视频，获取到相关音频，再根据论文中的设定剪切为 5s 的音频片段并以 wav 格式保存就完成了。

StarGANv2

在训练 StarGANv2 之前，先把论文浏览了一遍，达到了基本能看懂 train.py 和 inference.ipynb 里各个组件的程度，再把 GitHub 里的 Issues 基本都过了一遍，对哪些地方有坑、需要注意的点、如何选择 checkpoints 等等细节的工程问题都有一个大致的了解，然后根据掌握的信息，添加相关注释并更新相关代码，比如在 train.py 中添加注释，支持多显卡（multi-GPU）训练等，接着开始数据准备。

数据准备

根据作者的建议[7]，最好使用 10 个不同的人进行训练，每人提供 20 分钟的音频数据，数据为 2s 的音频片段。根据要求，选取了比较有代表性的十个人：李总、石总、成总、堃总、赵立坚、华春莹、董明珠、马云、李赣，作为训练集[1]。在确定发音者的阶段，本来选取了主席作为发音者，但是因为其说话速度慢、政治敏感度高，所以将其移出了列表，换为马云。

> 此处呈现的发音者列表是最终的列表，如果需要查看迭代历史，查看 #5 中回答的编辑历史即可。

因为模型接受非平行语料，所以只需要从公开网站中获取相关数据。鉴于公司的内部成员数据都是视频形式，为了统一处理和获取数据，决定使用「视频 -> 音频 -> 剪切 -> 筛选」的工作流来建立数据库。所有的视频皆是在 bilibili 中选取，并使用 https://bilibili.iiilab.com/ 下载，再使用 prepreocess.ipynb 中的 `# rename the videos` 代码块将其进行重命名为「姓*名*n.mp4」的形式，姓与名皆为首字母大写；n 表示第几个文件。在视频选取中发现，央视主持人不是好选择，虽然其声音具有高度辨识度。如果是新闻直播的视频，他们的声音会与他人的声音夹杂在一起，筛选数据的成本太高；如果是朗诵或者演讲的视频，常常伴有背景音乐，不利于模型的训练。也因此明确了选取视频的要求：单人长时间演讲且不能有背景音乐。

完成视频数据的获取后，学习使用 ffmpeg ，以便将视频中的音频以特定形式提取出来喂给模型。特定形式是根据论文的设定和作者在 Issues 中的回答确定的，24k 采样率与 2s 的音频片段[1]。使用 prepreocess.ipynb 中的 `# convert video to audio` 代码块将视频文件转化为音频文件，并存放在 `./Data/raw` 文件夹内，命名方式与视频命名方式一致，「姓*名*n.wav」。使用 wav 格式储存音频文件的原因是经验性的，在 CycleGAN 和 StarGANv2 中，皆使用 wav 作为其音频的格式。再使用 prepreocess.ipynb 中的 `# Split the whole audio into 2-second segments` 代码块将音频文件剪切为 2s 的音频片段储存在发音者的文件夹内，比如 `Li_Hua_1.wav, Li_Hua_2.wav` 皆被剪成片段储存到 `./Data/Li_Hua` 文件夹内。

完成了数据的预处理后，需要人工进行筛选。我在筛选音频的过程中，得到了几个经验性的标准：

1. 每一段音频片段至少需要包括 4 个完整的汉字音节，虽然这个标准相比于论文作者 Demo 里英文音频的含字量少了很多；
2. 尽量选取之前没出现过的完整音节。选取的音频可以都归为演讲类，每个人的风格不太一致，有的人偏向于强化输出一个概念，使得音频中某些字样的出现频率很高；有的人偏向于富信息流的输出，使得音频中字样重复的概率很低
3. 尽量选取少语气助词、人称代词、和停顿的片段；
4. 保证音频片段是连续的，不存在单字的音节被片段切开的情况。

根据以上标准，一些音频因为语速慢、人称代词多、语气助词多、音节分割等因素被抛弃，为每个发音者筛选了严格大于 24 min 的音频数据，确保训练集和验证集有充分的数据。

数据筛选完后，使用 prepreocess.ipynb 中的 `# conduct train_list and val_list` 代码块将数据分为训练集和验证集，分别存放于 `./Data/train_list.txt` 和 `./Data/val_list.txt`。该代码块使用种子确保训练集和验证集在复训时的一致性。

完成了训练数据集的构建后，使用 prepreocess.ipynb 中的 `# construct pred segments` 代码块构建预测集数据，数据储存在 `./Pred/yisa` 中，形式和 `./Data` 中一致。如果使用 `raw` 中的文件直接进行预测，显存可能会直接爆掉，所以直接将原始音频文件切割成 15s 的音频片段，以便后续进行转化测试以检验模型效果。

环境搭建

在虚拟机的环境搭建遇到了一些小问题，比如 #6，#7，之后都比较顺利了。

在 Windows 的环境中遇到了很多问题，比如 #27，需要用 conda install torch 而不是 pip 才能避免包不完整的问题；#30，需要将 NVSMI 添加进系统环境变量才能调用 `nvidia-smi`，和安装包时遇到的一些网络问题，比如 #29，需要关闭 Windows 的所有代理服务后 ClashX 才能接管网络连上 anaconda，同时需要取消 https 的代理设置；#26，共享文件夹最好是写 IP 地址而不是服务器名称，否则容易找不到服务。

通过 setup.sh 实现环境的一键部署，但是需要事先安装 Anaconda 和代理软件，且限定代理端口。简言之，只能完成虚拟环境的布置，不能完成项目从无到有的环境布置。整体过程可参考 #25。

模型训练与评估

在阅读文献和前期准备中已经思考好了模型的设置和数据的设定[2]，根据此修改了 ./Config/config.yml，然后进入项目的根目录中运行

```
python train.py --config_path ./Configs/config.yml
```

开始训练。

模型一共训练了两个版本。第一个版本中包含一些不成熟的数据，文件路径也不规范，对代码的修改主要停留在参数配置上。训练第一版的目的是确认训练代码能否正确运行，确认连续运行中代码是否有问题，计算训练到 250 epoch（论文中提及到的训练数）时间。完成第一版的测试训练后，对模型的表现进行了测试和验证，证实了模型确实是有效的且表现远高于 CycleGAN。在训练时长方面，CycleGAN 暂时没法进行并行上的优化，训练完大致需要 15天，根据第一版的时间测试；根据第一版的测试，StarGANv2 的训练时间也在 15天左右。基于表现和训练时长，取消了 CycleGAN 的后续的优化和训练计划，全力以赴优化 StarGANv2。

> 注，这里的预估时间是错误的，因为第一版中的数据并不充分，没有达到每人 24 分钟。

因为作者提供的推理示例代码是基于他代码仓库的 Demo，无法进行批量的推理，且虚拟机里好像没有声卡，无法播放音频，所以在第一版测试的时候写了批量推理 offline_infer，方便整体评估模型的表现。

第二版的模型中，再次筛选了发音者，对局部数据进行了优化和筛选，步骤与数据准备中一致，迭代过程可参考 #5 中回答的编辑历史。对数据存储的路径、模型存储的路径、注释、批量推理，标签错误 #12 等地方进行了细节上的优化。然后开始了第二版的训练。

在模型训练期间和训练完之后，听转化音频，对比版本，选取最佳 checkpoint。因为对抗神经网络难以使用客观的损失函数来评估模型的表现效果，需要花大量的时间听音频，对比版本之间的差异、同一版本不同转化器的差异、转化音频与原声之间的差异。评估的结果显示，epoch 248 的表现最佳，mapping 转化器的表现全方位好。mapping 比 style 的表现好的原因暂时推测为 style 使用英文预训练模型的风格模型，在纯中文环境中表现不佳，而 mapping 没有使用[3]。

实时推理

确定好模型版本之后，开始写实时推理部分的代码。这部分的代码涉及了一些工程问题和思考，比如：

- #10 ，首先思考了实时推理部分代码的逻辑，再根据逻辑思考相关的工程方面的问题和相应的解决方案。
- #13，如何使用 Python 捕获音频数据并进行播放。因为论文作者提供的实体推理代码 realtime_infer.py 使用的 `sounddevice` 包，我直接沿用了。在 souddevice.ipynb 中熟悉该包的功能和原理，比如 #28，以及进行一些测试，比如声道设置、设备选择、检验数据储存形式确保数据流动的一致性、音频流测试等等。
- #15，如何开始和结束展示系统，最开始考虑的是使用关键词激活技术（Keyword Spotting, KWS）开始语音转化系统，再使用 KWS 结束系统。这期间也遇到了一个类似的问题 #19，如何选取发音者。所以从 KWS 技术过渡到 TTS，让一个 TTS 系统一直在后台运行，建立一系列关键词规则用于激活和关闭系统，再用一系列规则用于选择发音者。但这意味着需要再训练一个模型，且需要后台一直跑一个 TTS 服务消耗显存，从训练和部署时间成本和预设的计算成本来说，有些得不偿失。所以最初阶段和长时间采用的是在命令行里自行选择发音者。
- 使用什么设备接收和输出语音。最开始的设想是皆使用蓝牙设备，因为蓝牙设备方便携带，且可以直接别在说话人的衣服上，方便收音。在测试阶段发现，如果蓝牙设备在程序运行中途断开连接或蓝牙匹配在程序运行之后，程序都没法成功调用蓝牙设备。除非写相应的监听脚本，#24 确保蓝牙设备的连接稳定性 ，如果断连，快速重连，重启程序；以及在程序运行的时候检查脚本，蓝牙设备是否已经成功连接。基于我的工程能力和时

间成本，暂时抛弃了蓝牙使用蓝牙设备的计划，拥抱有线设备。在这期间，#16 尝试自动识别并连接任意的可输入/输出的设备，但因为难以实现且用处不大放弃了。

- #23 使用单例模式（Singleton pattern）实现进行多次不同数据的推理，但只调用一次模型，节省模型加载的时间和显存的压力，推理速度得到明显的提高，10s 音频推理速度从单次 2s 左右进入到首次 1.5s 左右，复用分秒级。

代码迭代的思路如下[5]：

- 第一版：
  - 输入设备：PC 内置麦克风；
  - 输出设备：PC 内置听筒；
  - 推理模式：离线、预定义说话时间的推理；
  - 发音者选择：手动；
  - 启动模型：手动。
- 第二版：
  - 输入设备：**外置麦克风**；
  - 输出设备：**外置听筒**；
  - 推理模式：离线、**任意**说话时间的推理；
  - 发音者选择：手动；
  - 启动模型：手动。
- 展示系统：
  - 输入设备：外置麦克风；
  - 输出设备：外置听筒；
  - 推理模式：离线、任意说话时间的推理；
  - 发音者选择：**网页交互，参与者选择**；
  - 启动模型：**网页交互，参与者选择**。

在代码的迭代和测试中，出现的问题和思考经历：

- #45 选择哪一个 Windows 音频驱动能得到更好的效果。~经测试，MME 比 Windows DirectedSound 效果更好，WASAPI 和 WMD-KD 无法自定义采样率为 24k，使用这两个驱动意味着需要对音频数据 downsampling 到 24k 喂给模型，再将转化音频 upsampling 到 48k 还给驱动，在这个过程中会导致声音变形很多。~
- #41 在第一版的时候，采用的是预设音频时长，没有想到好的方法提示参与者什么时候可以开始说话，提示参与者可以结束说话了，以及选择发音者 #19。当时想靠展厅人员进行人工的语音提示，然后剪切掉提示的语音。在第二版中，使用音频流将「需要提示参与者录制结束」的问题转化为「什么是哦呼告诉系统录制结束了」的问题，省下了后续的提示。但这两个版本皆需要人工在后面进行操作，在实际展示中颇为不便。版本三采用了网页交互，让参与者自行选择开始和结束时间，以及发音者，彻底解决了需要人工后台操作的问题。
- #32 在传递数据流的时候，频繁报错，通过打印，发现 `sounddevice` 录制音频存储在 `numpy.array` 里是 2 维的，而非 1 维，因此需要对数组进行拉平。它使用 2 维数组的原因是，输入的声道可能为双声道，1 维数组无法储存双声道信息。同时，#34 在录制音频的时候，只能使用单声道。如果使用双声道，会导致拉平后的数组是单声道的两倍，变相拉长了音频的时长，导致声音失真。#40 思考能否将双声道转化为单声道，看起来这样捕获了更多的音频数据。考虑到项目的实际场景，电信诈骗，手机通话时通常都是单声道进行输入和输出，因此这里没有必要再进行额外优化了。
- #33 无法正常地播放转化后的音频。 `sounddevice.play()` 执行不等待的话，会直接到下一行代码，这里可能是因为进程的问题。在 `.play()` 之后加上 `sounddevice.wait()` 完美解决。
- #35 在测试阶段，发现前几秒音频的转化效果都很差，在原始音频前加入一个 1s 的空白区，局部缓和了这种现象。

- #39 无法用 `logging` 写 log，因为 StarGANv2 在其它地方已经调用过 `logging` 包了，这时候执行会导致其它的 log 全都打印到该 log 文件里，无法得到推理专属的 log 文件，所以自己写了一个 log 函数。

基于这些思考、流程、问题，逐步完成了实时推理部分的代码 real_time_infer.py。

- 思考逻辑见 #10；
- 任务管理见 https://github.com/thsno02/StarGANv2-VC/projects/1；
- 版本迭代见 https://github.com/thsno02/StarGANv2-VC/discussions/22。

虽然名字是实时推理，但实际效果并不是实时推理，即在说话时即时输出转化音频。这种实时效果可以做，模型推理速度上没有任何问题，根据多次的测试，10s 的音频数据首次推理时间为 1.5±0.2s，复用时推理时间为 0.5±0.2s。在展示的时候，实时推理需要输入和输出设备分离，避免输出音频污染输入音频，换言之，需要说话者戴耳机才能实现真正的即时推理效果。然而这在展示的时候不方便，需要参与者佩戴两个设备，且需要准备若干耳机才能获得比较好的效果，在经济上也不划算。在真实的电信诈骗场景中，需要这种即时效果；在展示中，暂不需要。

目前实现的实时推理是指，参与者完成说话后，播放其转化后的整条语音。经过测试，1070 支持时长不超过 60s 的音频转化，更新后的设备尚未测试，但 3070Ti 理应支持更长时间的音频转化，这应该完全满足展示需求了，毕竟很少有参与者会说超过 60s 的音频。

在测试中，得到了一些经验性的窍门[4]：

- 参与者的说话速度中等或慢一些，有助于提高转化的音频质量。这可能和训练的样本有关，因为训练数据中 2s 的音频大多只含有 4 到 6 个字的完整音节，少数人的样本有更多。
- 参与者的口音不能太重，即语调不可过高或者过低；语音尽量平缓。这可能和训练样本以及预训练模型有关，因为 F0 预训练模型是基于开源的英文数据集做的，数据集中的数据大量是平坦（plain）的语音语调，很少有高亢或者低沉的数据；风格预训练模型同样是基于英文做的，虽然有语音语调，但是音高同样是较为平坦的，而训练数据中大部分也是如此。
- 参与者可以尝试使用方言。普通话与四川话能取得可接受的转化效果，山东话尚未测试。

展示系统

展示系统地址：https://github.com/thsno02/voice-conversion

系统架构

目前展示系统的核心逻辑是用户访问网页，通过网页按键交互控制录音的开始与结束，同时通过网页选择按键挑选中意的发音人。后端程序接收到开始信号和发音人后开始录制用户的音频，当接收到结束信号后结束音频录制，并进行语音转化与播放转化后的语音。

该展示系统的迭代逻辑与上述提到的代码迭代逻辑一致，从必须要工作人员在后台控制程序的开始结束，到用户使用网页自己控制录音的开始与结束。该架构使得展示系统能独立运行而不依赖于工作人员的指令，同时交互性使展示更有沉浸感与体验感。

演示流程

1. 扫描 二维码；
2. 根据网页提示，选择「发音者」；
3. 点击「开始」，当「开始中」变为「结束」后，即可开始说话；
4. 点击「结束」，结束说话，等待片刻后会自动播放转化后的音频；
5. 重复 2 - 4 即可。

实施部分

展示系统设计的工程问题比较多：

- https://github.com/thsno02/voice-conversion/issues/11 展示系统的核心问题是，如何按需求获取音频流数据。最开始的阶段，通过为输入音频流单独启动一个进程来获取相关的数据。但这样，音频流的数据只放在了该进程里，取数据需要额外再进行操作，整体比较笨重且不简洁。接着过渡到使用为输入音频流启动一个线程，这样不需要额外引入第三个变量来存储整体的音频流数据，写法也更简明。在调试输入音频流的时候，遇到了：
  - https://github.com/thsno02/voice-conversion/issues/9 无意间注释掉某一行代码，导致函数参数设定出错；
  - https://github.com/thsno02/voice-conversion/issues/12 `audio` 中无法获取到正常长度的音频数据，因为使用了两次 `queue.get()`，过度提取数据了；
  - https://github.com/thsno02/voice-conversion/issues/13 调试时，开始和结束太快了，无法完成音频的录入，因为中间没有使用 `time.sleep()` 进行休息。
- 在打通前后端的连接时，解决了一些繁琐的问题，比如，https://github.com/thsno02/voice-conversion/issues/7 在 html 里设置了本地地址导致非 host 的机器无法访问；https://github.com/thsno02/voice-conversion/issues/14 没有根据 flask 要求设置相关的静态资源，导致无法加载 `play.html`；https://github.com/thsno02/voice-conversion/issues/23 前端和后端数据的格式与字典内容不一致。
- 在整体的调试中，import 问题出现了多次。https://github.com/thsno02/voice-conversion/issues/2 无法调用与父级文件夹平行文件夹内的文件，这是因为我的文件组织方式背离了 Python import 的设定，按照 https://github.com/thsno02/voice-conversion/issues/20 中的形式更改了文件夹结构和添加文件夹的前缀，彻底解决了 import 问题。
- https://github.com/thsno02/voice-conversion/issues/8 开始 flask 调试模式后，flask 会加载文件两次。再启动 flask 服务后再加载模型，这样能避免模型的多次加载问题。
- https://github.com/thsno02/voice-conversion/issues/24 取消模型部分的 logging 指令后，可以正常些日志了。
- https://github.com/thsno02/voice-conversion/issues/28 一键激活展示系统，并在激活的时候自动显示二维码。

贡献

展示系统由多人贡献而成。@monchickey 提出了展示系统的基本构架和运行逻辑，优化了输入音频流的逻辑，发现并解决了若干工程问题；冯晓坤提供了输入流部分的 demo，demo 指明了使用进程作为输入音频流的容器，并用 flag 开启和结束进程的逻辑，以及在最开始解决父级文档的 import 中提供了帮助；@zhengsonglinge 提供了所有前端部分的代码。

调优与测试

在第六周，对模型和系统进行了较为全面的测试，主要涉及：

- 找到各个发音者对应的最佳模型 checkpoint；
- 测试、比较不同外接设备的实际转化表现；
- 用各种形式测试现有模型的鲁棒性。

实验设计：

- #48 在云桌面里进行模型 checkpoints 选择的时候，发现相同的发音者在不同的 checkpoints 中表现有好有坏，且看起来有新的可用的发音者出现 。
- #50 根据这个发现，我决定对 checkpoints 的表现进行一个全面的测试，以期为每一个发音者找到起对应的可用的 checkpoint。
- #52 设计完实验并进行了一些测试后，发现「完备性」的时间开销太大了，需要至少 10 小时才能完成，于是转变了思路，直接使用我的语音进行转化，找到表现最好的发音者和 checkpoint。在这期间，实验设计流程出现了问题，我一度想把模型数据都取出来，然后逐一进行转化测试，没有考虑到我本地没有运行 CUDA 的条件，也没考虑到使用展厅电脑完成测试的便利程度。

实验结果[8]：

- 在云桌面里，看起来真的出现了很多「可用」的发音者，但是没考虑云桌面里的发音者都是模型见过的，而本任务的背景和目的是「any-to-many」而不是「many-to-many」。我当时被模型的转化效果惊艳到了，没有对此进行思考和验证，为什么表现提高了，具体在哪方面提高了，慌忙之中就汇报了成果。根据实验设计对我的音频进行转化后发现，模型的泛化性没有提高，epoch 742 和 epoch 248 在泛化性上没有本质区别，但是随着 epoch 的增长，模型在「many-to-many」任务中表现越发出色。
- 模型的表现不稳定。模型对音量是敏感的，在低音量下，模型几乎不能正常工作；在正常（中等）音量下，模型能正常工作；在高音量下，模型偶尔能正常工作。模型对语音语调有一定的鲁棒性，用普通话和四川话都能得到可接受的转化效果，能明显听出来是发音者的语音语调。模型的表现依赖于外接麦克风的采样质量。

外设测试：

- 背景：根据之前的测试和实验结果表明，麦克风的采样质量会极大影响模型表现，所以找到最佳的外设设备是非常重要的。
- #54 逐步确定了测试和实验的逻辑。首先明确的是，各个设备的音响都不会对模型表现造成影响。#51 其次经过长时间的测试发现，500款的设备表现是最好的，但是依然会出现发音不完全的问题，相比电脑内置的麦克风。将设备在笔记本和台式机上进行实验后，发现录音不完全和电脑本身没有关系，同时都出现发音不完全（录音不完全）的问题，唯一的可能是，这些外设都自带了驱动，而这个驱动的在被 python 调用的时候会出现采样的问题。
- 为验证上述猜想，我使用 3.55mm 的耳机作为麦克风，结果发现台式机无法正常录入音频，numpy 里录入的音频数据出现了大量一致的数据且分布很均匀，相比正常的音频。简入言之，3.55mm 的耳机插入台式机后没法正常工作，但是在 mac 上能正常工作。目前只能认为是 windows 有问题了，因为进行麦克风测试的时候，音量大小始终提不上去。

综上所述，多训练没增强模型的泛化性，可用的发音者依然只有两个。windows 对 python 调用设备录音的支持不太好。最好使用 3.5mm 接口的设备，使用 windows 内置的音频驱动会提升录音效果（推测）。

UPDATE：

- https://github.com/thsno02/voice-conversion/issues/27 windows 调用外置麦克风的时候，采样率必须与设备本身的采样率一直，否则无法成功使用设备。这是 Windows 和 Python 一起埋的坑。

对系统整体也进行了一些调优：

- https://github.com/thsno02/voice-conversion/issues/31 一键启动脚本无法正常展开二维码，因为 flask 服务还没有结束。将展示二维码写入 `server.py`，单独起一个进程保证其一直挂起。
- 直接展示的二维码并不好看，虽然有提示，但是中间的 logo 是 Chrome 的 logo 而不是公司的 logo。https://github.com/thsno02/voice-conversion/issues/34 将公司的 logo 插入到二维码的中间。https://github.com/thsno02/voice-conversion/issues/39 尝试在二维码上方添加指引文字，但是 pillow 对中文文字的渲染效果支持太差了，放弃了这个想法。将 logo 调为圆角，并加上白框，再使用 conner.py 将加框后的 logo 调为圆角。

- 运行时遇到了一些小问题：https://github.com/thsno02/voice-conversion/issues/35 没有使用 pip 安装对应的包；https://github.com/thsno02/StarGANv2-VC/issues/55 `librosa` 已经不负责音频的 IO 了，转而使用 `soundfile`；https://github.com/thsno02/StarGANv2-VC/issues/56 `soundfile` 在读取音频时不能限制采样率；https://github.com/thsno02/voice-conversion/issues/36 实时获取当前机器的 ip 地址，并生成对应的二维码，提高项目的鲁棒性。
- https://github.com/thsno02/voice-conversion/issues/37 对前端页面进行了修改，使其在手机端扫码访问时能有更好的体验。
- https://github.com/thsno02/voice-conversion/issues/38 修改了一下多模型切换的问题，极大降低了切换模型带来的时间损耗。

反思

- 建立宏观认知和思考是做好项目和解决问题的充分必要条件。在拿到问题后，搞透彻问题的核心是什么，再对其整体的调研和思考，绝不能简单的使用路径依赖和看到什么用什么。
- 根据论文复现结果，还是得看懂论文，只在宏观层面上大致明白模型架构有时候是行不通的。
- 在决定使用一个论文模型之前，仔细研究它的论文、GitHub、社区里对论文的评价和讨论，是极其有帮助的。能够少走很多弯路，也能对这个模型有更多的思考和理解。
- 文件结构要尽量简洁，能只用一层就不要用二层，尤其是对训练数据和代码包的组织。
- 工程问题消耗的时间远大于训练模型消耗的时间，比如数据准备大约花了一周，模型花了一周，而工程代码和整合花了三周。工程代码不比模型代码，一定要多次 review，不要到报错的时候才会去看代码，多看几遍再执行，做到心中有数。
- 学会看报错信息对快速定位问题非常有帮助。
- 团队合作和及时沟通很重要，能节约大量的时间。多向团队中的人学习。
- 在知道逻辑后，写不出代码 or 写不好代码很令人烦躁。要多学习。
- 写文档比写代码还要快乐一点。
- 用 GitHub 组织项目和进行任务管理还是不错的，各个模块连通做的不错。
- 测试和实验是一项很消耗精力和体力的活动。精力主要体现在实验设计上，在实验设计中始终需要贯彻「控制变量」这个原则，再结合实验目的和现有的资源，设计可执行的实验方案。在最开始的时候，想对模型进行完备的测试，是一种理想主义和不切实的想法；如果有一个实验要求的时间和资源特别大，那应该进行一个小规模的实验，验证一下可行性和效率，同时进行一些剪枝。
- 「控制变量」的意思是，对「每一个」变量都进行「详尽」的考察。

结论及发展

本项目的模型完全基于 @yl4579 优秀的论文及代码，展示系统基于网页交互。因为模型的特性，可以实现任意人的语音实时转化为特定发音者，在声调与语调较平坦的声音中能得到不错的转化效果。网页交互使得展示能够较为智能地进行，不需要人在后台操作。

模型只训练了两个版本，且只采用了 10 个发音者的声音。根据论文作者最近在 GitHub 的回复，可以适量地增加发音者的数量，这样能使得模型的泛化性更强[6]。当前版本的训练数据不够规整，存在大量的噪音以及有不合适的发音者，后续可以继续对数据进行筛选和优化。模型的并行训练效果不太好，因为 CPU 超负荷了，但是 GPU 的负荷却很小，后续可以优化模型的并行训练效率，节省模型迭代的时间。目前的网页界面非常简约，后续可以优化交互界面与增加展示页面的内容。

UPDATE:

经过更为详尽的测试，目前的模型表现只能说「可用」，在实际运用中的鲁棒性不够强。这可能训练数据集的低质量、没有微调 Vocoder 和 F0 模型、训练人数较少都有关系。实际生产环境和给定数据集的学术环境确实大有不同。

致谢

在完成语音转化项目中，感谢圭圭姐在前期发起了很多高质量的讨论，使我能快速定位到问题和找到相关的模型；同时，圭圭姐也提供了相关的视频数据和视频到音频的脚本。感谢石总、胡承圆周在模型测试时提供大力支持与诸多建议，使我能更全面和细致地分析模型表现，更进一步优化模型的实际效果。感谢 Simon 指出使用单例模式加载模型，极大优化了模型的推理速度。感谢曾总 @monchickey 提出的网页交互方案，并同郑松岭 @zhengsonglinge、冯晓坤在后续的网页搭建与系统整合中给出大量宝贵的建议与提供大力的技术支持。感谢武总在项目的各个阶段参与讨论，明确了后续模型的优化方向。非常感谢上述各位的贡献！

脚注

[1]: https://github.com/thsno02/StarGANv2-VC/issues/5

[2]: https://github.com/thsno02/StarGANv2-VC/issues/4

[3]: https://github.com/thsno02/StarGANv2-VC/issues/11

[4]: https://github.com/thsno02/StarGANv2-VC/discussions/38

[5]: https://github.com/thsno02/StarGANv2-VC/discussions/22

[6]: https://github.com/yl4579/StarGANv2-VC/issues/51

[7]: https://github.com/yl4579/StarGANv2-VC/issues/17

[8]: https://github.com/yl4579/StarGANv2-VC/issues/51#issuecomment-1192343647