

2023-10-11



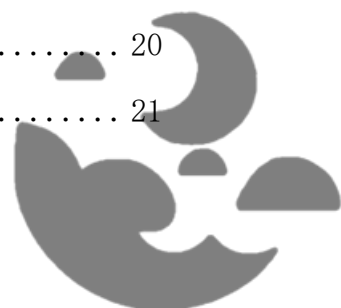
夜雨为伴

开发者文档

两三点·雨夜

目录

夜雨为伴开发者文档.....	3
前言.....	3
总述.....	3
开发硬件要求.....	4
建议的配置要求:	4
开发硬件环境:	4
开发环境部署.....	5
源代码.....	5
Python 环境	6
PyTorch 与 GPU 环境	6
其它依赖项.....	7
开发环境检查.....	9
运行入口.....	10
工作目录.....	11
项目结构介绍.....	11
整体结构.....	11
Source.....	12
Form_UI.....	12
UI (Qt 控件) 命名习惯	13
image.....	13
rvc.....	13
text.....	14
Code.....	14
Ui.....	15
Lib.....	15
GUI.....	20
API.....	21



功能模块实现逻辑（宏观讲述）	21
AI 语言模型	21
AI 音色推理	22
资源包与.json 结构	22
资源包.....	22
Json 文件结构	23
软件封装结构.....	25
嵌入式 python	25
启动器.....	25
主程序.....	26
版本的区分.....	26
未来功能 to do.....	27
结语.....	27



夜雨为伴开发者文档

前言

尊敬的开发者您好，感谢您对于夜雨为伴开发工作的支持。“天降大任于是斯人也，必先……”，不好意思跑题了，在进行夜雨为伴的开发工作、查看源代码之前，您必须完成以下的步骤：

While True:

 阅读并同意《夜雨为伴用户使用协议》

 阅读并同意《ApacheLicense2.0》

好了，如果你阅读完了，那么你可以跳出这个死循环了，接下来，我们进入正题。

这是一份面向开发者的文档，详细讲述了夜雨为伴的代码结构，功能模块，实现逻辑……但是在此之前，您还需要《夜雨为伴使用说明书》。如您所见，这份文档对于“怎么用”并没有半毛钱的介绍，有的只是“怎么做”。

So:

在此之前，了解一下怎么用还是很有必要的。

此外:

古人云：“两百行代码跑功能，两千行防刁民”。如果你看到了很多意义不明的代码，注释掉还不影响程序运行，那么，信我，不要删除。放在那里就好了，说不定哪个刁民会让你极度无语的把注释掉的代码请回来。

古人还云：“程序和程序员有一个能跑就行。”我尽量把开发者文档写的详细，为各位开发、学习、复制/粘贴提供便利。

总述

开源地址: <https://github.com/TTORainyNight/NightRain>

这个玩意的开发对硬件要求还是蛮高的，没有好 CPU 的 30 系以上显卡谨慎操作。
各位程序员小心 GPU 烤肉哦。



整个项目都是用 Python 开发的，虽说运行效率不如 C，但是有 Pytorch 这么个神器，而且面向对象是真的很方便耶，不是么？欸嘿~~

C 语言是世界上最好的语言捏~.py

开发硬件要求

都是同行，咱就别整什么最低配置了，有钱换个好电脑吧，何苦折磨自己呢，直接上推荐配置：

建议的配置要求：

CPU: Intel i5-11400H +

内存: 16GB +

储存空间: 32GB +

显卡 (GPU): RTX 3050 (仅对于 Nvidia30、40 系显卡做维护)

显存: 4GB +

显示器: 分辨率 1920×1080 +

系统: Windows 10 64 位 +

眼熟吧，我直接复制过来的，能偷懒肯定不自己写。为了表示诚意，把我的开发环境告诉各位程序员，辛苦我的小笔记本电脑：

开发硬件环境：

CPU: Intel i7-11800H

内存: 16GB DDR4 2933×2

储存空间: 啊？1024 个 PB 可以不？

GPU: RTX 3050 4GB

系统: Windows 11 home 22H2 x64 (你看到的时候可能已经重装了)



开发环境部署

是的，又到了部署开发环境的时候了。各位老司机应该不陌生了，上车！

源代码

不要去封装好的环境里复制！

不要去封装好的环境里复制！

不要去封装好的环境里复制！

说三遍!!!! 那里的代码是被精简过的，环境也是被精简过的，为了封装包更小，而且代码结构和相关库并不完整。

请向上看，抬头。找到开源的 Github 地址，完整的下载整个源代码包，然后看着这篇文章去构建开发环境。封装的那个小 python 环境能不用就不用，到时候连个 pip 都没有，而且依赖项目还不全，残缺的。可别怪我没提醒你哦~~~

从 GitHub 上面下载好完整的代码后，还不能立即运行。我知道你很急，但你先别急。GitHub 限制文件大小，所以有一个资源文件放不到源代码中，我放在实例(releases)里面了，名字叫 hubert_base.pt。下载后，放到./ Source/rvc 文件夹中。完整的 Source 文件夹是这样的：

名称	修改日期	类型	大
 32k.json	2023/7/24 14:19	JSON 文件	
 40k.json	2023/7/24 18:40	JSON 文件	
 48k.json	2023/7/24 14:19	JSON 文件	
 hubert_base.pt	2022/12/14 0:31	PT 文件	18
 test_voice.mp3	2023/8/10 19:15	MP3 文件	

终于拿到完整的项目源代码文件了，接下来，去用户的下载链接，去下载个现成的资源包。随便下载一个就可以，拿来测试程序用。



Python 环境

if 你已经获取到了完整的代码：

开始配置 Python 环境。

else： 返回上一条。

先说一下 Python 版本，首先是范围要求：3.8-3.10 版本，RVC 模块要求不能低于 3.7 版本，但是在 3.11 和 3.12 版本上，有个依赖死活安装不上。也许是开发者还没有做新版本的适配吧。大家别踩坑了。3.9.13 3.10.11 3.10.12，这三个版本是我测试过的，确认可用。

我的开发环境是在 3.10 版本下的，64 位 Python。也正是因为版本原因，使得不支持 Windows 7 系统。对于 32 位 Python，理论上是可以的，没有超过 4GB 的超大文件，但是我没有进行测试。各位开发者可以自己踩踩雷。

可以使用官方 Python 包，此处使用 anaconda 创建 Python 环境。国内使用之前可先切换清华源，加快下载速度，创建新的虚拟环境：

```
Conda create -n NightRain python=3.10
```

之后激活环境：

```
Conda activate NightRain
```

PyTorch 与 GPU 环境

十分不推荐使用 CPU 环境，除非实在是没办法了。在创建完虚拟环境并激活后，首先安装 PyTorch 的 GPU 开发环境。可以到网上搜索教程，还蛮多的，这里给出一些关键步骤，老咸鱼看着这些肯定能自己解决了。

PyTorch 官网：<https://pytorch.org/get-started/>

- 查看版本：首先确定你的 GPU 所支持的 CUDA 版本，然后确定其所对应的 cuDNN。这俩都可以在 Nvidia 官网找到。然后确定 PyTorch 所支持的 CUDA 版本，确定完毕后，进行下一步。

哎呀，不要无脑装最新呀，这个雷帮你们踩过了，呜呜呜呜呜~~

- 安装 CUDA 和 cuDNN



Conda 安装这俩还是很简单的，两句命令即可：

查看可用：conda search cudatoolkit - info

安装 CUDA：conda install cudatoolkit=11.8

查看可用：conda search cudnn - info

安装 cuDNN：conda install cudnn=8.7

如你所见，我用的是 CUDA11.8 + cudnn8.7，然后在 pytorch 官网找到 CUDA11.8 版本对应的 pytorch 安装命令，直接复制到 conda 命令行中：

```
pip3 install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu118
```

PyTorch Build	Stable (2.1.0)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.1	ROCm 5.6	CPU
Run this Command:	<pre>pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118</pre>			

经过漫长的下载、安装后，咱们终于把 pytorch 的 GPU 环境搞完了，接下来安装其他的依赖项目。

其它依赖项

- 不废话了，直接上文件：

,,,

joblib>=1.1.0

numba==0.56.4

numpy==1.23.5

scipy==1.9.3

librosa==0.9.1

llvmlite==0.39.0



fairseq==0.12.2
faiss-cpu==1.7.3
gradio==3.14.0
Cython
pydub>=0.25.1
soundfile>=0.12.1
ffmpeg-python>=0.2.0
tensorboardX
Jinja2>=3.1.2
json5
Markdown
matplotlib>=3.7.0
matplotlib-inline>=0.1.3
praat-parselmouth>=0.4.2
Pillow>=9.1.1
resampy>=0.4.2
scikit-learn
tensorboard
tqdm>=4.63.1
tornado>=6.1
Werkzeug>=2.2.3
uc-micro-py>=1.0.1
sympy>=1.11.1
tabulate>=0.8.10
PyYAML>=6.0
pyasn1>=0.4.8
pyasn1-modules>=0.2.8
fsspec>=2022.11.0



```
abs1-py>=1.2.0
audioread
uvicorn>=0.21.1
colorama>=0.4.5
pyworld==0.3.2
httpx==0.23.0
torchcrepe==0.0.20
fastapi==0.88
PyQt6
PyQt6-tools
edge-tts
pygame
openai
ffmpeg
,,,
```

另存为一个.txt 依赖文件，然后使用命令直接一步到位：

```
pip install -r xxxx.txt
```

- 然后把安装失败的依赖项目手动重装一下，都成功了就不用了：

```
pip install edge-tts
```

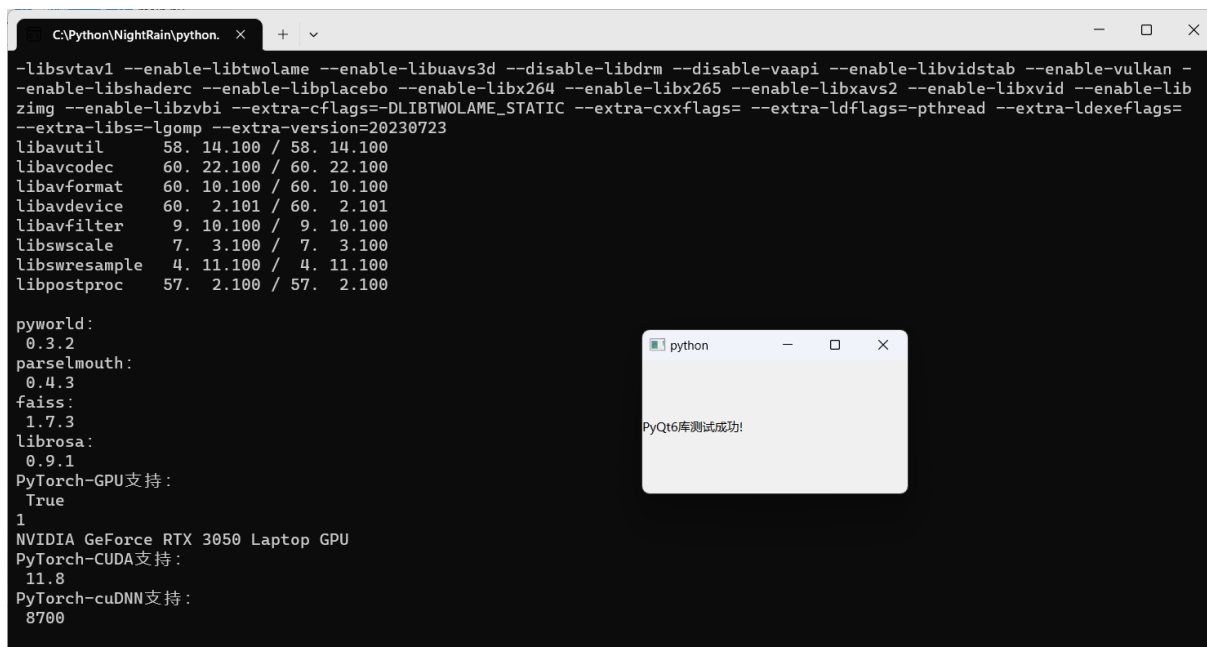
•都弄完之后，找到 Python 环境的目录，然后依次进入包的安装环境，把 QtDesinger 给提取出来，或者直接右键发送到桌面快捷方式。

开发环境检查

我所使用的编辑器是 Visual Studio 2023，当然你可以用喜欢的 IDE。在配置好环境后，在的编辑器中打开项目文件夹，然后在 Code 文件夹下，可以找到“环境检测.py”，把它设置为启动文件，以刚刚配置的程序。

运行环境监测：





```

C:\Python\NightRain\python. x + v
-libsvtav1 --enable-libtwolame --enable-libuavs3d --disable-libdrm --disable-vaapi --enable-libvidstab --enable-vulkan --
-enable-libshaderc --enable-libplacebo --enable-libx264 --enable-libx265 --enable-libxavs2 --enable-libxvid --enable-lib
zimg --enable-libzvbi --extra-cflags=-DLIBTWOLAME_STATIC --extra-cxxflags= --extra-ldflags=-pthread --extra-ldexeflags=
--extra-libs=-lgomp --extra-version=20230723
libavutil      58. 14.100 / 58. 14.100
libavcodec     60. 22.100 / 60. 22.100
libavformat    60. 10.100 / 60. 10.100
libavdevice    60.  2.101 / 60.  2.101
libavfilter     9. 10.100 /  9. 10.100
libswscale     7.  3.100 /  7.  3.100
libswresample  4. 11.100 /  4. 11.100
libpostproc   57.  2.100 / 57.  2.100

pyworld:
0.3.2
parselmouth:
0.4.3
faiss:
1.7.3
librosa:
0.9.1
PyTorch-GPU支持:
True
1
NVIDIA GeForce RTX 3050 Laptop GPU
PyTorch-CUDA支持:
11.8
PyTorch-cuDNN支持:
8700

python
PyQt6库测试成功!

```

这里会列出所有包的检查情况，如果你能看到：

一堆正确的信息、一个 PyQt6 窗口、一个 pygame 窗口、GPU 检测成功、没有报错。

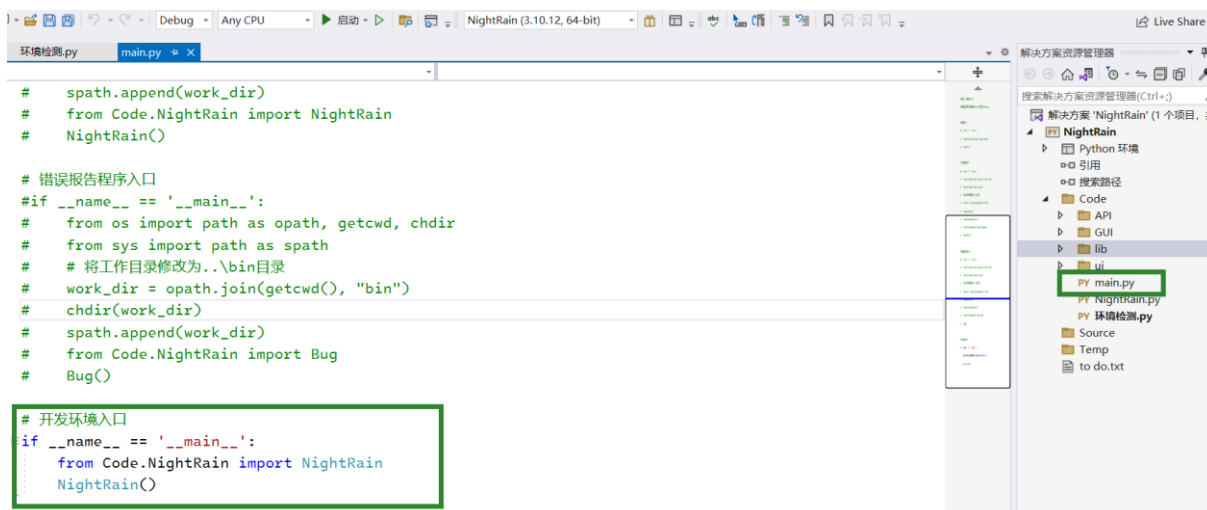
那么恭喜你，环境配置完成，您可以对夜雨为伴进行开发工作了！

运行入口

先不要管代码什么样子，咱们先跑起来再说。

进入工作目录，找到 Code\main.py，把它设置为启动文件。然后，等等，别运行！！！！

我知道你很急，但你先别急。这个文件有好几个入口，那都是封装的时候用的。你需要做的是把他们全部注释掉，仅仅保留“开发环境入口”，比如这样：



```

# spath.append(work_dir)
# from Code.NightRain import NightRain
# NightRain()

# 错误报告程序入口
# if __name__ == '__main__':
#     from os import path as opath, getcwd, chdir
#     from sys import path as spath
#     # 将工作目录修改为..\bin目录
#     work_dir = opath.join(getcwd(), "bin")
#     chdir(work_dir)
#     spath.append(work_dir)
#     from Code.NightRain import Bug
#     Bug()

# 开发环境入口
if __name__ == '__main__':
    from Code.NightRain import NightRain
    NightRain()

```

这时候，再去启动程序。如果你看到了夜雨为伴的 GUI 界面，证明启动成功。

工作目录

如果你出现了找不到包的情况，可能是你所使用的编辑器与 VS 不同，把 `main.py` 所在目录当成工作目录了，VS 是以 `.sln` 作为工作目录的。

所以解决方案也就很简单了，把 `main.py` 拖拽到 Code 文件夹的外面，放到上一级目录，再次启动程序，不出意外的话，就要出意外了，啊，不是，就可以正常运行了。

现在你可以点几个功能试一试，确认运行环境无误。既然都准备好了，那我们进入正题，开始说代码。

项目结构介绍

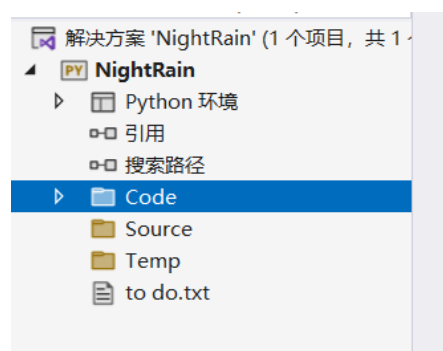
有图有真相，上图！

整体结构

Temp 文件夹一般是空的，这里面存放一些临时文件，里面的东西都可以删光光。

Source 文件夹存放程序运行所必要的资源文件、相关协议、UI 设计。不可以删除哦。

Code 文件夹存放程序的代码部分，是整个项目最主要的文件夹，当然更不能删除了。



Source

Form_UI

存放 GUI 的窗口设计文件，也就是.ui。可以通过 QtDesigner 打开，进行程序的界面设计。

Aboutform.ui “关于”窗口

Helpform.ui “帮助”窗口

Infoform.ui 信息窗口，哪里用到了哪里调用，用来展示大量信息，相当于一个大型的弹窗，没有固定的作用

Mainform.ui 主窗口，就是外面套着的那个壳

Setplusform.ui “高级设置”窗口，在“设置”中

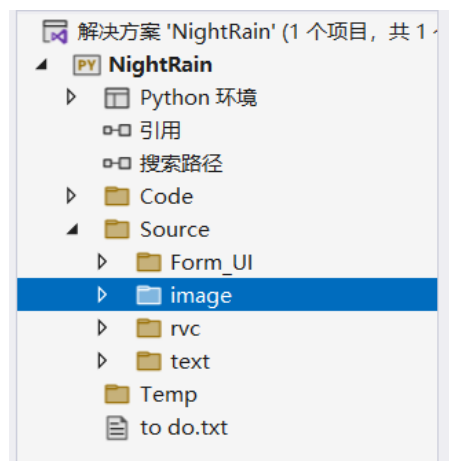
Settingform.ui “设置”窗口

Supportform.ui “赞助”窗口

Talkform.ui “对话”窗口

NightRain.qrc QtDesigner 所需要的资源文件，第一次打开的时候不出意外是会报错的。别紧张，这很正常。你只需要手动去重新指定一遍.qrc 文件位置，就是这个文件。然后就可以正常使用了。

UI 中有以 back 为名的 QLabel，那个是用来做背景的，一个展示背景照片，另一个用来做模糊效果。



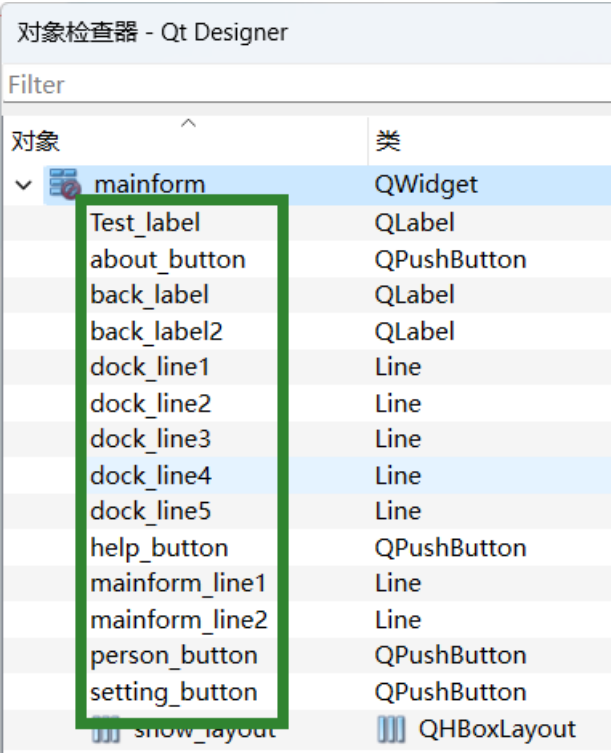
UI (Qt 控件) 命名习惯

emm……其实这个，人各有所好吧。这里说一下本项目的命名习惯，方便大家找控件的时候方便。控件和命名这部分都是在 QtDesigner 里面完成的，在 .ui 文件里面可以都找到。

对于窗口：统一使用 form 命名，前半部分是这个窗口的功能。

前半部分是这个控件负责的功能，后半部分是空间的类型，有些是简写的，比如 QPushButton 简写为 button。

有相同功能的一组，就在后面加个数字。额……其实有点强迫症了，这些无关紧要的只是展示个信息的控件，其实可以就扔在这里。



image

项目的图片资源，存放 GUI 运行所必需的图片文件，包括 logo 背景，还有其它需要图的地方。不要乱用哦，图片有版权的！

Load.png 加载界面图片

Logo.ico logo_name.png 程序的 logo 文件

Mianback.jpg 主窗口背景

SupportQR.jpg 捐赠二维码

Xiaomeng.jpg xiaomeng_V.jpg 两种规格的背景，小姐姐好漂亮的，有没有？

rvc

存放 RVC 项目的资源文件，还有个推理模型，具体内容参见 RVC 开源项目，在此处就不多言了。



啊？你问开源地址啊？在“关于”页面里面有。

test_voice.mp3 是程序加载完模型后，自我检测用的音频文件。

text

存放程序内部的一些文件信息，还有程序的协议、开源协议。

ApacheLicense2.0.txt 夜雨为伴用户使用协议.txt 两份协议的副本

about.txt “关于”信息

help.txt “帮助”信息

Code

代码全在这里了，根目录下有三个文件：

- main.py 程序运行入口，整个项目应该从这里开始运行。也是根据程序员们的习惯，放了个 main()。这里面有四个入口，已在注释中标明。

不要全打开啊，调试的时候应该只保留一个入口，其他的注释掉备用。在进行程序开发时，请仅仅保留“开发环境入口”。

“启动器入口”需要 pyinstaller 或其它工具编译，用来做一个 exe 的“套壳启动器”。

“主程序启动入口”这个是封装后的 mian.py，用来在封装后启动主程序，如你所见，它纠正了一些工作目录。

“错误报告程序入口”这个是封装后的 bug.py，用来在封装后启动 bug 报告，程序运行崩溃时，调用它展示一个错误报告。

- 环境检测.py 在构建项目依赖环境的时候使用，确定你的环境是否满足项目需求。
- NightRain.py 这个是真正的运行入口，用于在程序运行之前处理一些事情。比如展示个启动界面，展示一些 bug 提示信息。

什么？你问真正的程序启动入口？不想走这些花里胡哨的流程？

好好好，满足你。在这：.\Code\GUI\GUI_mainform.py。直接在这个代码上添加个 Qt 的 exec()，也可以跑程序，启动界面没有干其它的事情。你如果是想要拆一部分加



到自己的项目里，可以从这里拆。

Ui

这里面全是以“UI_”开头的文件，这些文件是源自 Form_UI 里面的.ui 文件，通过 pyuic6.exe 转化得到。转化完了我就放过去了，没有任何改动。在自己改动 UI 界面后，转化为.py 后直接替换相关文件，就可以看到效果。

值得注意的是，有一部分 ui 是通过 GUI 代码实现的，一般是一些 ui 转化后无法明确指定的文件。

还有就是一些奇奇怪怪的 bug

Lib

程序的核心功能类库，你可以理解成后端。这里面的所有内容，你都可以拿出来自己用。

它们并不依赖 GUI 界面，并且提供了自己的使用示例。

其中，以“nr”开头的，是可以调用的 API 接口。

nrData

实现对于用户数据的操作。

API 请求：

加载 user_data() 对象，使用对应方法。

user_data() 类方法：

save() 保存指定的字典到 json 文件，参数：

userdata 字典。 保存目标；

file 字符串，.json 文件路径，保存文件。

read() 从指定 json 文件读取字典，参数：

file 字符串，.json 文件路径，读取目标。目标 json 文件只能存放字典内容。

返回值 1 字典。读取到的字典。



check() 检测即将保存的设置参数是否存在错误。

输入参数列表：

json_file, name, tempdir, voicemodel_pth, voicemodel_index, prompt, device, method, upkey, temperature, address, indexrate, filter_radius, resample, rmsmix, protect

返回值 1 error_list 字典。存在错误的参数。键：错误参数，值：错误信息

返回值 2 rewrite_list 字典。需要纠正的参数。纠正一些可能被纠正的错误，请使用纠正列表的参数而不是原参数。键：已纠正参数，值：修正值。已经被纠正的参数不会出现在错误列表中。

相关示例请查看 API 部分。

nrGPT

实现语言模型的对话功能，通过 ChatGPT 实现

API 请求字典描述：

API_Key 字符串，以“sk-”开头。填写您的 Openai-API-Key，可以在 Openai 官网找到；

API_Base 字符串，api 网址，“https://”开头，结尾不含“/”。指定 Openai 的 API 接口或镜像网站；

System_Prompt 字符串。书写系统级 Prompt，一般是对 ChatGPT 的命令限制；

User_Prompt 字符串。用户级别提示词，会被添加到历史中，参与用户设定的对话；

History_File 字符串，.txt 文件路径。开启 History 时才有意义，可以不写。存放对话历史，注意此处的历史是面向 ChatGPT 的，如果要呈现给用户，请另作修改；

Error_File 字符串，.txt 文件路径。用于保存错误信息；

ChatModel 字符串，Openai 指定，例如“gpt-3.5-turbo”。用于指定使用的模型，目前支持 ChatGPT3.5 与 4.0 系列；

History 布尔型。用于指定是否启用历史功能；

Temperature 浮点型，0~1。指定 GPT 模型回答的随机性，越大随机性越强，可能性越多；



Visible 布尔型。用于指定是否启用请求过程可视化，为调试程序提供便利；

TimeOut 整型，非负整数。开启 Visible 时才有意义，可以不写。用于指定请求之前的延时，在此之前，展示输入信息；

Input 字符串。用户的输入内容。

单独调用的方法：

model_check() 需要 ChatModel Visible API_Key API_Base，检查 ChatGPT 的对话模型是否可用

model_list() 需要 Visible API_Key API_Base，返回所有该 Key 的可用模型

别骂了，别骂了。鬼知道我当时怎么想的写成这样，凑合用吧。相关示例还是去看 API 示例，后面可能会重构。

nrMusic

音频处理类，包含如下的模块：

音频混流 VoiceMix()

文字转语音 TexttoVoice()

音频播放 PlayVoice()

TexttoVoice()

需要网络，调用微软的 API 接口

text 字符串。要转换的文本内容；

voicesource 字符串。指定 edge-tts 语音模型，可以通过 python 命令“edge-tts --list”查看，复制 name 后面的那一串；

voicefile 字符串，.mp3 文件路径。指定保存到的文件；

rate 字符串。百分比语速控制。

PlayVoice()

voicefile 字符串，.mp3 文件路径。指定要播放的 mp3 文件；

请注意，该进程会执行到音频结束后退出。



nrRVC

这部分确实蛮复杂的,后面可能会更换更底层的模型,优化算法和适应性。这里只讲 API,如果需要研究原理,移步其它开源项目。

实现语音的转化,通过 RVC 模块实现

API 请求过程:

将 `nrVoiceChange()` 类作为对象加载

调用上述类的 `pre_voicechange()` 方法,搭建好转化环境

调用 `voicechange()` 方法,进行语音转化

`pre_voicechange()` 参数说明:

`device` 字符串, "cuda:0"或"cpu"。指定推理的设备 cuda:0 表示 GPU(如果可用)

`is_half` 布尔型。是否使用半精度,建议开启。但只有 GPU 支持。

`model_path` 字符串, .pth 文件路径。指定推理的声音模型

`visable` 布尔型。用于指定是否启用请求过程可视化,为调试程序提供便利

`timeout` 整型,非负整数。开启 Visable 时才有意义,可以不写。用于指定请求之前的延时,在此之前,展示输入信息

`voicechange()` 参数说明:

`input_audio` 字符串, .wav/.mp3 文件路径。待处理的音频文件

`f0up_key` 字符串,数字, "-12"到"12"。控制转化变调,男变女+12,女变男-12,同性别 0,可以微调音调

`f0method` 字符串, "pm"或"harvest"或"crepe"。推理模式, pm:CPU 高速低质, harvest:CPU 低速高质, crepe:GPU(推荐,如果可用)

`index_path` 字符串, .index 文件。声音模型 index 文件

`index_rate` 浮点数, 0.00-1.00 检索特征占比

`filter_radius` 浮点型, 0.0-7.0 中值滤波半径,削弱哑音, >=3 时会开启滤波



`resample_sr` 整型。重新采样率，0 表示不采样

`rms_mix_rate` 浮点型，0.00-1.00 输入音源替换为输出包融合占比，值越大越倾向于使用输出包

`protect` 浮点型，0.00-0.50 保护清辅音和呼吸声，拉满表示不开启。增加保护力度会削弱索引效果

`opt_path` 字符串，.wav 文件路径。输出文件

`visable` 布尔型。用于指定是否启用请求过程可视化，为调试程序提供便利

`timeout` 整型，非负整数。开启 `Visable` 时才有意义，可以不写。用于指定请求之前的延时，在此之前，展示输入信息

返回值为一个数组，它们分别表示[加载音频时间，加载 Huber 时间，声音转换时间]。
若返回 `False` 则表示转化失败，请检查目标文件。

nrStarter

`pyinstaller` 封装太坑，`pytorch` 丢.dll，`ffmpeg` 也丢。而且本来是开源的，也不怕别人拿源代码，所以就直接源码封装了，后面也好维护。

这里是封装的时候使用的，你可以单独提取出去做成万能套壳 `python` 启动器，然后制作一个 `main.py` 和 `bug.py` 就可以了。

这个没有 API 示例，因为是封装后用的，在开发环境下不好弄，如果不封装的话，忽略就好了。

nrUpdate

请注意，版本是通过“关于”中的文本首行获取的，不是指定的文件。后面可能会考虑更改。

更新模块，请求新版本

API 方法说明：



`get_update()`: 从网络上获取更新 json;

`is_update()`: 对比版本信息, 检测是否存在新版本;

`get_version()`: 从当前文件获取版本信息。

API 参数:

`Update(visable)`: `visable` 布尔型。指定是否开启访问过程中的可视化。

`get_update()` 参数说明:

`urls` 数组, 元素为网址。指定获取 json 的网址;

`max_frequency` 整数。最大尝试次数。

返回值 1 布尔型。是否获取到更新, 如果为 `False` 请检查网络状态;

返回值 2 字典。获取到的更新 json。

`is_update()` 参数说明:

`update_json` 字典, 网络上的更新字典;

`version` 浮点数, 数字, 当前的版本号。

返回值 1 布尔型。代表是否需要更新。

`get_version()` 参数说明:

`about_file` 字符串, 指定一个 .txt 文件路径。一般为本地 `about.txt`, 第一行存放版本信息;

返回值 1 `is_error` 布尔型, 是否出现错误;

返回值 2 `beta` 布尔型, 是否为内测 beta 版本;

返回值 3 `information` `is_error = True` 时, 字符串, 存放错误信息; `is_error = False` 时, 浮点型, 存放版本号。

详细示例请查看 API。

GUI

这里面是主要的前端代码实现, 与之匹配的是 `Code\ui` 里面的 `ui` 文件, 以及 `lib`



中的“con_”开头的文件，你可以理解成接口转接吧。

其中，所有 GUI 类具有统一的 ui 参数，也就是“self.ui”。这里面对应 ui 中的各种控件，与类名字长得最像的那个方法(比如 GUI_mainform 对应的 self.mainform())。把它看成 form_load 槽函数，实现窗口加载时的功能。

值得注意的是，GUI 调用后端库时，也就是 lib 里面的 API，并不会带有任何的详细注释，看不懂的话可以去 API 部分，有单独的调用示例。

除了各窗口对应的 GUI_xxxform.py 之外，还有一个过别的文件：

GUI_control.py 这里面放了一些 GUI 界面的接口，主要是提高一些 GUI 代码的复用性，供给给其它界面直接调用。

API

找不到这个文件夹？你是不是直接从封装好的程序扒的代码？火速去 Github 下载完整包，都告诉你了，封装的代码残缺的，不完整哦。

相信这个地方不用过多介绍了，每个文件都带有详细的注释，说明，使用。它们针对于 Code\lib 中的每一个可调用的 API。你可以通过这些 API 文件，去分离出你自己想要的功能。

代码中书写的很详细，所以在文档中不多介绍。

功能模块实现逻辑（宏观讲述）

本程序对于虚拟人的实现方式，是通过 AI 大语言模型+AI 声音音色推理模型。

用户输入信息 (Qt) → 组合 prompt → 联系上下文 → 发送大语言模型 (ChatGPT) → 回复文本 → 转化基础语音 (edge-tts) → 音色推理拟合 (rvc) → 播放声音 (pygame)

AI 语言模型

此处使用 openai 的 ChatGPT 接口，通过 system_prompt 去指明虚拟人身份，限制虚拟人的行为。命令禁止虚拟人的部分行为，同时指定基础操作，让大语言模型明白自己在干什么。这部分指令不经过文件，直接写死在代码中。



哎呀，这段提示词确实得优化，在改啦，不要催啦！

虚拟人的身份使用 `user_prompt` 实现，即用户自主编写的提示词文件，此处指明虚拟人的身份等信息，指明回答的风格，允许用户自由发挥。

历史记录部分，通过 `user_prompt` 发送所有的对话信息，实现上下文功能。这个方法其实很笨，很废 API，不过 openai 官方就是这么指定的，我也没办法。同时有对于 GPT 历史长度的检测，为了保证虚拟人“脑子”的质量，达到一定长度的历史后，将会强制结束，令用户清空历史。

其他参数可以指定大语言模型的行为。

AI 音色推理

此处使用 RVC 开源项目，对于已有的声音，结合用户指定的声音文件，进行音色推理，达到拟合虚拟人声音的功能。

同时支持对于 GPU 和 CPU 设备的检测，当然最好使用 GPU 了，玩 AI 没点好卡还真是不太行。CPU 推理的速度质量其实也还可以，属于能用的阶段。

资源包与 .json 结构

资源包

说一下用户资源包吧，这里以纳西妲的资源包为例，仅使用纳西妲作为示例，无任何侵权意图。（保命保命，毕竟咱也没有侵权的任何意图，保护好自己嘛）



此电脑 > My (B:) > NightRain > Alnaxida				
名称	修改日期	类型	大小	
Alnxd.index	2023/4/22 12:39	INDEX 文件	139,510 KB	
Alnxd.pth	2023/4/22 12:39	PTH 文件	53,827 KB	
nxd.json	2023/9/20 21:47	JSON 文件	1 KB	
prompt.txt	2023/9/20 21:53	文本文档	1 KB	
vhuman.jpg	2023/8/26 22:25	JPEG 图像	169 KB	
纳西妲.json	2023/9/18 20:39	JSON 文件	1 KB	
纳西妲.json.bak	2023/8/26 21:40	BAK 文件	1 KB	
声明.txt	2023/9/17 13:01	文本文档	1 KB	

纳西妲.json，这是虚拟人的主要配置文件，相关的内容在接下来展开说明。

Alnxd.index Alnxd.pth 这两个文件是一体的，是用户自己训练的声音模型。兼容所有 RVC 模型，可以直接导入使用。

Vhuman.jpg 此文件将会在后续版本删除，展示一个人物照片作为背景，很呆的，后面会考虑换成建模，并且加相应动作，实现更加进一步的虚拟人体体验。

prompt.txt 给 ChatGPT 看的，指定其虚拟人的身份。这里用户可以自主去编写，写成什么样子就看自己的想象力了。

声明.txt 对于程序运行是没啥用的，在这里主要对资源包的版权做一些声明，以及提示用户去遵守一些协议什么的。

其他文件是我自己调试用的，直接忽略。

Json 文件结构

可以打开看一下，这里面放着一个 python 字典。




```
{
  "VHumanName": "纳西妲",
  "ChatModel": "gpt-4",
  "Address": "https://chatgpt1.nextweb.fun/api/proxy",
  "Temperature": 1.0,
  "History": true,
  "UserPromptFile": "prompt.txt",
  "TempDir": "Temp",
  "VoiceModel_pth": "Alnxd.pth",
  "VoiceModel_index": "Alnxd.index",
  "Device": "GPU",
  "Method": "crepe",
  "Upkey": 0,
  "IndexRate": 0.75,
  "Filter": 3.0,
  "Resample": 0,
  "Rmsmix": 0.25,
  "Protect": 0.33
}
```

"VHumanName": 虚拟人的名字，指定展示的身份

"ChatModel": ChatGPT 的调用模型，GPT-4 的效果是真好，不过也是真贵

"Address": ChatGPT 请求地址，这里可以换成国内镜像呀，就不用翻墙用了，你还可以改成一些第三方的转发地址，实现对于国内 API 的支持。

"Temperature": ChatGPT 参数，越大越随机。

"History": 历史功能的开关，最好是打开哦，经费吃不消了就关掉吧。

"UserPromptFile": 指定 prompt 提示词文件位置。

"TempDir": 临时目录，存放一些程序的缓存。

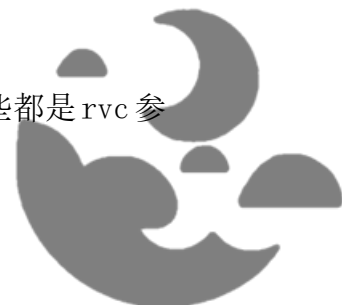
"VoiceModel_pth": 指定声音模型的位置

"VoiceModel_index": 指定声音模型的位置

"Device": 指定推理设备

"Method": 指定推理方式

"Upkey" "IndexRate" "Filter" "Resample" "Rmsmix" "Protect": 这些都是 rvc 参



数了，可以去用户文档里面查看他们的说明，我是真的不想再写一遍了，好麻烦的哦。

软件封装结构

我直接用的源码封装，别问，问就是太麻烦了。

直接用 pyinstaller 各种丢.dll，给我整麻了都，大抵是 ffmpeg 的问题，以后慢慢研究。而且本来就是开源项目，也不怕偷代码。源码封装还好改，升级包也可以做的很小。

嵌入式 python

这里使用的是嵌入式 python，对这个感兴趣的话，可以自己去找一找相关的文章教程，基本思路是获取同版本的嵌入式 python，获取 pip，只安装运行所需的包，然后删减不需要的包和文件，最后调整文件结构，把源代码扔进去。再删减一些不需要的源代码。

我也是站在巨人的肩膀上，源码封装，香！好用的哦！

到这里，你知道为什么不要用封装环境的 python 去搞开发了吧，虽然是源代码封装，但是删减掉了大量的内容，甚至连 pip 依赖都是不全的。所以请从头开始，构建完整的开发环境。

启动器

源码封装，用过的都知道，会残留一个.bat 处理文件，指向相对应的 python.exe 和.py。那这个肯定不行呀，太丑了。

所以我造了个小启动器，它就是个外壳，这样用户在运行的时候，就可以直接双击.exe 了，是不是瞬间好看多了？

启动器的源代码只有两个文件：main.py nrStarter.py 如果你的项目也需要源代码封装，你大可以把这俩文件给复制出来用。

启动器很简单，首先纠正工作目录，然后调用 pthonw.exe，这个解释器不会展示控制台界面，比较好看；python.exe 会展示控制台界面的，让这个嵌入式 python 解释器



去执行我的源代码。这个启动器是直接通过 `pyinstaller` 编译的，同时启动器具有捕获和提示异常错误的功能，这也是程序必备呀，毕竟谁也不知道会蹦出什么始料未及的 bug 来。

当然，更多的还是防刁民。

只需要一句 `pip install pyinstaller`，然后在命令中调用 `pyinstaller` 就可以了，相关参数可以参考其官方文档。

主程序

说一下封装结构吧，启动器没什么好说的，就是打包完了直接复制过来的，然后在目录中添加 `.runtime`，把嵌入式 python 塞进去。再加一个 `.bin`，把源代码塞进去，然后把上文提到的运行入口拆出来，保留 `main.py` 和 `bug.py`。把这两个文件扔到外面，供启动器去调用。

是的哦，这里就不能用开发环境入口了，需要把 `main.py` 中，注释掉的那些入口改回来。

所以，`.bin` 文件夹里面就是源代码（封装版本）了，封装的时候只是改了改启动入口。其中 `bug.py` 会直接调用 `GUI_infoform.py`，去展示一些错误日志和信息。你就把它当成一个大号弹窗就可以了。

版本的区分

本程序的版本读取、更新功能写在 `nrUpdate` 模块中，版本的读取是通过 `about.txt` 文件的首行去获取的。如果需要更改版本，那么直接改这个文件的第一行就可以。

数字部分代表当前程序的版本号；`V` 没啥意义，纯属好看；`beta` 后缀代表内测版本，会影响主页的内测水印和关于中的版本显示。

例如：“`V 1.3`” “`V 1.4beta`”

更新的 `json` 文件直接放在 Github 仓库里面了，买不起服务器，Github 就挺好用的。

主要是方便呀，欸嘿~



只是获取一下更新信息，也不会对 Git 造成很大的负担，主要是图个稳定。那个更新的.json 我也是通过 nrData 直接写出来的，如果你要自己写一些程序，复制我的代码的话，请注意把地址改成你自己的仓库，请求的时候加了两个国内镜像，防止网络原因获取不到更新。

未来功能 to do

我愿称之为画饼模块# to do

忽然想起几年前的项目还有 to do 没解决……

好了，言归正传，to do_list 也就是计划功能列表，想要添加，有这个想法，但是还没有加进去，打算在后续版本添加的功能。或者是有一些调试代码，要注释掉的部分。

你可以在整个项目中搜索关键词“to do”来寻找这些点，找到我已经在程序中留下的坑，或许可以把它填起来。也有还有一些因为能力不足而无法添加的功能。

- 流传输模式（大幅提升体验的好嘛）
- 抛弃 pygame（好像有其他的声音模块了，重复了）
- 抛弃 edge-tts，直接使用 AI 模型转化
- 抛弃人物图片，替换为建模与动作

题外话：这部分我真的不会，目前仍在学习，如果你有对于 OpenGL 和 MMD 的经验，那么请联系我，共同把夜雨为伴做得更好。

- 优化“设置”机制（把一些设置项从资源包里面拆出来）
- 添加新手指引
- 优化启动速度（虽然现在已经接近秒开了，但是我认为可以更快）
- 精简应用大小（torch 库太大了……主要是我还不删……）

结语

开发者文档到这里就结束了，再次感谢您对于夜雨为伴开发工作的支持。也欢迎您通过程序内的联系方式与我进行交流。

好啦，没什么说的了，最后祝你生活愉快吧！编程之余，多陪陪家人哦~



2023 年 10 月 11 日星期三

两三点·雨夜

