不要直接双击打开run07.py

要先打开vscode，打开整个AIAnchor文件夹，然后选择run07.py文件，运行

注意选择的python解释器，目前使用的是AIAnchor里的gpt-sovits里自带的python

注意路径是否要加r或者斜杠、反斜杠的变换

你可以创建一个字典，将代号映射到具体的路径。

# 定义路径字典

model\_paths = {

'Hang\_vocal\_1': r'F:\AIAnchor\Input\GPT-SoVITS\_Input\models\Hang\_vocal\音色\_1\Hang-e15.ckpt',

# 添加其他路径

}

# 通过代号获取路径

def get\_model\_path(code):

return model\_paths.get(code, "路径未找到")

# 示例

path = get\_model\_path('Hang\_vocal\_1')

print(path)

你可以使用 Python 脚本来实现对奇数和偶数 .txt 文件的不同操作。以下是一个示例代码，它将遍历指定文件夹中的 .txt 文件，并根据文件名的奇偶性执行不同的操作：

python

import os

# 文件夹路径

folder\_path = 'F:/AIAnchor/Output/Paragraphs\_Output/test\_20240901\_051709'

# 获取文件夹中的所有文件

files = os.listdir(folder\_path)

# 遍历文件

for file\_name in files:

# 只处理 .txt 文件

if file\_name.endswith('.txt'):

# 获取文件名的数字部分

try:

# 提取文件名中的数字部分

base\_name = os.path.splitext(file\_name)[0]

file\_number = int(base\_name)

# 判断奇偶性并执行不同的操作

if file\_number % 2 == 0:

# 偶数文件，打印 'test' + 文件名

print(f'test{file\_name}')

else:

# 奇数文件，直接打印文件名

print(file\_name)

except ValueError:

# 如果文件名不是数字格式，跳过该文件

print(f'Skipping non-numeric file: {file\_name}')

**代码说明**

1. **导入 os 模块**: 用于处理文件和文件夹操作。
2. **设置文件夹路径**: 修改为你实际的文件夹路径。
3. **列出文件夹中的所有文件**: 使用 os.listdir() 获取所有文件名。
4. **遍历文件**: 过滤 .txt 文件并处理文件名。
5. **提取数字部分**: 使用 os.path.splitext() 分离文件名和扩展名，然后尝试将文件名转换为整数。
6. **判断奇偶性**: 使用取模操作 (%) 判断文件名是否为偶数。
7. **执行操作**: 根据文件名的奇偶性打印不同的内容。
8. **处理非数字文件**: 如果文件名不能转换为整数，则跳过。

请确保文件夹路径使用的是正斜杠 / 或双反斜杠 \\，因为单反斜杠 \ 在字符串中通常用于转义字符。如果文件夹路径包含空格或特殊字符，也可能需要考虑路径的转义或正确处理。

你可以使用 Python 的 os 模块来创建文件夹。以下是一个示例代码，根据给定的 file\_name 在指定路径下创建一个新文件夹：

python

import os

# 文件夹路径和要创建的子文件夹名

base\_folder\_path = 'F:/AIAnchor/Output/SadTalker\_Output/test\_20240901\_065503'

file\_name = 'example' # 替换为你要使用的 file\_name

# 创建完整的文件夹路径

new\_folder\_path = os.path.join(base\_folder\_path, file\_name)

# 创建文件夹（如果不存在）

if not os.path.exists(new\_folder\_path):

os.makedirs(new\_folder\_path)

print(f'文件夹已创建: {new\_folder\_path}')

else:

print(f'文件夹已存在: {new\_folder\_path}')

**代码说明**

1. **导入 os 模块**: 用于处理文件和文件夹操作。
2. **设置基础路径和文件夹名**: base\_folder\_path 是基础文件夹路径，file\_name 是要创建的子文件夹名称。
3. **创建完整路径**: 使用 os.path.join() 合成新文件夹的完整路径。
4. **检查文件夹是否存在**: 使用 os.path.exists() 判断文件夹是否已存在。
5. **创建文件夹**: 使用 os.makedirs() 创建新文件夹（包括任何必要的父文件夹）。

确保将 file\_name 替换为你希望使用的实际文件名。

要打印一个路径，可以使用 Python 的 print() 函数。这里是一个简单的示例：

import os

# 文件夹路径

path = 'F:/AIAnchor/Output/SadTalker\_Output/test\_20240901\_065503/example\_folder'

# 打印路径

print(path)

**代码说明**

1. **导入 os 模块**: 这里其实没有用到，但通常用于路径处理。
2. **定义路径**: 将 path 变量设置为你要打印的路径。
3. **打印路径**: 使用 print() 函数输出路径。

以下是几种可以帮助你分段和分配稿子的方法和工具：

**分段工具**

1. **NLTK（Natural Language Toolkit）**
   * **简介**: NLTK是一个强大的Python库，用于处理和分析自然语言文本。
   * **功能**: 你可以使用NLTK的分句器和分段器来分割文本。
   * **示例代码**:

python

import nltk

from nltk.tokenize import sent\_tokenize

# 下载句子分割模型

nltk.download('punkt')

# 读取稿子

with open('稿子.txt', 'r', encoding='utf-8') as file:

text = file.read()

# 分割为句子

sentences = sent\_tokenize(text)

# 进一步分段可以基于一定的规则，例如主题变化、逻辑断点等

1. **spaCy**
   * **简介**: spaCy是一个现代化的Python NLP库，提供高效的文本处理功能。
   * **功能**: 可以使用spaCy的分句和分段功能。
   * **示例代码**:

python

import spacy

# 加载预训练模型

nlp = spacy.load('en\_core\_web\_sm')

# 读取稿子

with open('稿子.txt', 'r', encoding='utf-8') as file:

text = file.read()

# 创建文本对象

doc = nlp(text)

# 分段（可以根据需要进一步调整）

paragraphs = [sent.text for sent in doc.sents]

1. **TextBlob**
   * **简介**: TextBlob是一个简单的文本处理库，适合快速实现基本的NLP任务。
   * **功能**: 提供了文本的分句功能，可以用于分段。
   * **示例代码**:

python

from textblob import TextBlob

# 读取稿子

with open('稿子.txt', 'r', encoding='utf-8') as file:

text = file.read()

# 创建TextBlob对象

blob = TextBlob(text)

# 分割为句子

sentences = blob.sentences

**分配段落**

一旦你将文本分成了段落，你可以根据需求将这些段落分配给两个人。以下是一个简单的分配方法：

1. **均等分配**:
   * 将文本分成段落后，简单地将段落均匀地分配给两个人。例如，前一半段落给一个人，后一半段落给另一个人。

python

num\_paragraphs = len(paragraphs)

mid = num\_paragraphs // 2

person\_1 = paragraphs[:mid]

person\_2 = paragraphs[mid:]

1. **根据内容分配**:
   * 如果有特定的内容或主题分配要求，可以根据这些要求将段落分配给不同的人。
2. **轮流分配**:
   * 如果文本较长，也可以采用轮流的方式分配段落。例如，第1、3、5段给一个人，第2、4、6段给另一个人。

python

person\_1 = [paragraphs[i] for i in range(0, num\_paragraphs, 2)]

person\_2 = [paragraphs[i] for i in range(1, num\_paragraphs, 2)]

**工具和平台**

* **Google Colab**: 你可以在Google Colab上使用Python代码运行上述示例，方便进行文本处理。
* **Jupyter Notebook**: 同样，Jupyter Notebook也是一个进行代码实验和文本处理的好选择。

根据你的需求，你可以选择适合的工具和方法进行分段和分配。如果需要进一步的帮助，请告诉我！

你可以在 Para01.py 文件的开头添加以下声明来指定编码：

python

# -\*- coding: utf-8 -\*-

这样 Python 就会使用 UTF-8 编码来解释文件，从而解决编码错误。

文本分类模型是用于将文本分配到预定义类别的模型。它们在处理各种自然语言处理任务时非常有用，包括文本分段、情感分析、主题分类等。以下是一些常见的文本分类模型及其基本原理：

**1. 朴素贝叶斯（Naive Bayes）**

* **原理**: 朴素贝叶斯分类器基于贝叶斯定理，假设特征之间相互独立。这种方法简单且高效，特别适用于大规模文本分类任务。
* **优点**: 实现简单，计算速度快，对小样本数据表现良好。
* **适用场景**: 垃圾邮件分类、情感分析等。

**2. 支持向量机（SVM）**

* **原理**: 支持向量机通过在高维空间中寻找最佳超平面来进行分类。它特别适用于二分类任务，也可以通过扩展到多分类。
* **优点**: 能处理高维数据，适合文本分类。
* **适用场景**: 文本分类、图像识别等。

**3. 词袋模型（Bag of Words, BoW）**

* **原理**: 词袋模型将文本表示为词的集合，并忽略词的顺序和语法。每个文档被表示为一个词频向量。
* **优点**: 简单易懂，适用于许多文本分类任务。
* **适用场景**: 基本文本分类任务，如情感分析、新闻分类。

**4. TF-IDF（Term Frequency-Inverse Document Frequency）**

* **原理**: TF-IDF 结合了词频（TF）和逆文档频率（IDF）来衡量词在文档中的重要性。它能有效减少常见词对分类的影响。
* **优点**: 能提高模型对关键字的识别能力。
* **适用场景**: 文本分类、信息检索等。

**5. 词嵌入模型（Word Embeddings）**

* **原理**: 词嵌入模型将词映射到高维连续向量空间，这些向量能够捕捉词之间的语义关系。常见的词嵌入模型有Word2Vec、GloVe等。
* **优点**: 能捕捉词义和上下文信息，提升分类精度。
* **适用场景**: 语义理解、文本分类等。

**6. 深度学习模型**

* **卷积神经网络（CNN）**: 通过卷积层提取文本中的局部特征，适合处理文本分类任务。
* **循环神经网络（RNN）**: 处理序列数据，能够捕捉文本的上下文关系。长短期记忆网络（LSTM）和门控递归单元（GRU）是改进的RNN结构。
* **Transformer 模型**: 如BERT、GPT等，通过自注意力机制处理文本序列中的关系，适合处理复杂的文本分类任务。

**7. 预训练语言模型**

* **BERT (Bidirectional Encoder Representations from Transformers)**: 通过双向编码器捕捉上下文信息，适合多种自然语言处理任务。
* **GPT (Generative Pre-trained Transformer)**: 基于生成模型的预训练，能够生成连贯的文本和进行文本分类。
* **RoBERTa、XLNet、T5**: 这些是BERT和GPT的变体，针对不同的任务进行了优化。

**应用示例**

假设你有一篇长篇稿子，你可以使用以下步骤来分段并分配给不同角色：

1. **文本预处理**: 使用分词工具将长篇稿子拆分为句子或段落。
2. **文本表示**: 使用词嵌入模型将文本转换为向量表示。
3. **分类模型**: 使用预训练模型（如BERT）或训练的分类器对文本进行分类，将其分配到不同的段落或主题。
4. **分段与分配**: 根据分类结果，将文本分成不同的部分，并分配给相应的角色或类别。

这些模型和工具可以根据需求进行选择和调整，以实现最佳的文本分段效果。

以下是几个推荐的模型，用于智能分段和文本分类：

1. **BERT (Bidirectional Encoder Representations from Transformers)**:
   * **优点**: 强大的上下文理解能力，适用于多种文本处理任务，包括文本分类和段落分割。
   * **适用**: 需要深入理解文本的任务，如主题分类和文本摘要。
2. **GPT-3/GPT-4**:
   * **优点**: 强大的生成和理解能力，可以用于文本生成、分类、摘要等任务。
   * **适用**: 生成式任务和需要较高理解能力的文本处理。
3. **RoBERTa (Robustly optimized BERT approach)**:
   * **优点**: 改进版的BERT，处理效果更强，特别是在大规模数据上表现出色。
   * **适用**: 需要处理大规模数据的任务，如大文档的智能分段。
4. **DistilBERT**:
   * **优点**: BERT的精简版本，计算效率更高，适合资源有限的环境。
   * **适用**: 对性能要求高但计算资源有限的应用。
5. **XLNet**:
   * **优点**: 结合了自回归和自编码模型的优点，更好地捕捉文本的依赖关系。
   * **适用**: 需要对文本进行深层理解和生成的任务。
6. **T5 (Text-To-Text Transfer Transformer)**:
   * **优点**: 将所有NLP任务统一为文本到文本的问题，适应性强。
   * **适用**: 多任务处理，包括分段、分类、摘要等。

这些模型可以使用Hugging Face的Transformers库进行访问和应用，它提供了预训练模型和简单的接口，适合各种文本处理任务。根据具体需求，你可以选择合适的模型并进行微调。

以从指定的文本文件读取内容。以下是包含此功能的完整代码示例：

python

import os

def read\_text\_from\_file(file\_path):

"""

从指定的文本文件读取内容。

参数:

file\_path (str): 文本文件的路径。

返回:

str: 文件中的文本内容。

"""

with open(file\_path, 'r', encoding='utf-8') as file:

return file.read()

def save\_paragraphs\_to\_files(paragraphs, folder\_path):

"""

将段落保存为txt文件，依次命名为1、2、3等等。

参数:

paragraphs (List[str]): 需要保存的段落列表。

folder\_path (str): 文件夹目录路径。

"""

os.makedirs(folder\_path, exist\_ok=True)

for i, paragraph in enumerate(paragraphs, start=1):

file\_path = os.path.join(folder\_path, f'{i}.txt')

with open(file\_path, 'w', encoding='utf-8') as file:

file.write(paragraph)

# 指定文本文件路径

text\_file\_path = 'input\_text.txt'

# 从文本文件读取内容

text = read\_text\_from\_file(text\_file\_path)

# 自动分段（指定自定义分隔符并保留分隔符）

paragraphs\_custom\_delimiter = auto\_split\_text(text, max\_length=10, preserve\_delimiters=True)

# 指定文件夹路径

folder\_path = 'output\_folder'

# 保存段落到txt文件

save\_paragraphs\_to\_files(paragraphs\_custom\_delimiter, folder\_path)

print(f"段落已保存到文件夹：{folder\_path}")

**更新内容：**

1. **read\_text\_from\_file**: 从指定路径读取文本文件内容。
2. **text\_file\_path**: 设定输入文本文件的路径。
3. **text**: 从文件中读取文本，然后进行分段。

确保将 'input\_text.txt' 替换为你实际的文本文件路径。

我希望在F:\AIAnchor\Output\Paragraphs\_Output文件夹里，新建一个以选择的txt文件的文件名称和当前时间的结合的文件夹，将生成的那些txt文件保存到这个文件夹里

2024-09-01 03:27:24

你可以使用以下代码来实现你的需求：

python

import os

import datetime

def read\_text\_from\_file(file\_path):

"""从指定的文本文件读取内容。"""

with open(file\_path, 'r', encoding='utf-8') as file:

return file.read()

def save\_paragraphs\_to\_files(paragraphs, folder\_path):

"""将段落保存为txt文件，依次命名为1、2、3等等。"""

os.makedirs(folder\_path, exist\_ok=True)

for i, paragraph in enumerate(paragraphs, start=1):

file\_path = os.path.join(folder\_path, f'{i}.txt')

with open(file\_path, 'w', encoding='utf-8') as file:

file.write(paragraph)

def create\_output\_folder(base\_folder, file\_name):

"""创建以文件名和当前时间结合的文件夹。"""

timestamp = datetime.datetime.now().strftime('%Y%m%d\_%H%M%S')

folder\_name = f"{os.path.splitext(file\_name)[0]}\_{timestamp}"

folder\_path = os.path.join(base\_folder, folder\_name)

os.makedirs(folder\_path, exist\_ok=True)

return folder\_path

# 指定文本文件路径

text\_file\_path = 'input\_text.txt'

base\_output\_folder = 'F:\\AIAnchor\\Output\\Paragraphs\_Output'

# 从文本文件读取内容

text = read\_text\_from\_file(text\_file\_path)

# 自动分段（假设有实现）

paragraphs\_custom\_delimiter = auto\_split\_text(text, max\_length=10, preserve\_delimiters=True)

# 创建以文件名和当前时间结合的文件夹

file\_name = os.path.basename(text\_file\_path)

output\_folder = create\_output\_folder(base\_output\_folder, file\_name)

# 保存段落到txt文件

save\_paragraphs\_to\_files(paragraphs\_custom\_delimiter, output\_folder)

print(f"段落已保存到文件夹：{output\_folder}")

**更新内容：**

1. **create\_output\_folder**: 创建一个以选择的txt文件的文件名称和当前时间结合的文件夹。
2. **base\_output\_folder**: 基础输出文件夹路径。
3. **output\_folder**: 生成的文件夹路径，用于保存段落文件。

确保将 'input\_text.txt' 替换为你实际的文本文件路径。

我想要在python里表示一个路径，其中不变的为F:\test，变化的为test里面的文件，里面有很多文件，比如名称为1.txt，那么我将1.txt记作a，那该怎么表示这个路径呢

2024-09-01 02:58:19

你可以使用字符串格式化来表示这个路径。假设文件名是一个变量，你可以这样做：

python

file\_name = '1.txt'

path = f'F:\\test\\{file\_name}'

这样，path 就会变成 F:\test\1.txt。如果你要处理多个文件，只需将 file\_name 更改为相应的文件名即可。

2024-09-01 02:58:37

这里的f的作用是什么呢

2024-09-01 02:58:37

在 Python 中，f 是指 **f-string** 或 **格式化字符串字面量**，是一种在字符串中嵌入表达式的简洁方法。这是在 Python 3.6 中引入的一种功能。

f-string 的作用是使字符串格式化变得更直观和易读。通过在字符串前加上 f，你可以在字符串中直接嵌入变量和表达式，它们会被自动替换为对应的值。

以下是 f-string 的一些关键特性：

1. **简洁的语法**：你只需在字符串前加上 f，然后在字符串中使用花括号 {} 包围变量或表达式。例如：

python

name = "Alice"

age = 30

message = f"My name is {name} and I am {age} years old."

上述代码会生成字符串 "My name is Alice and I am 30 years old."。

1. **表达式支持**：你可以在花括号内使用表达式，不仅限于简单的变量。例如：

python

x = 10

y = 5

result = f"The result of {x} + {y} is {x + y}."

这会生成字符串 "The result of 10 + 5 is 15."。

1. **格式化**：你可以使用格式化选项来控制数字的显示格式。例如：

python

pi = 3.14159

formatted\_pi = f"Value of pi: {pi:.2f}"

这会生成字符串 "Value of pi: 3.14"，即将 pi 格式化为两位小数。

总之，f-string 使得字符串插值更为直观和便捷，是处理字符串的强大工具。

save\_dir = os.path.join(args.result\_dir, strftime("%Y\_%m\_%d\_%H.%M.%S"))

2024-09-01 03:01:10

在你的代码中，os.path.join 和 strftime 用于生成一个包含时间戳的目录路径。让我们逐步解析一下这段代码：

python

save\_dir = os.path.join(args.result\_dir, strftime("%Y\_%m\_%d\_%H.%M.%S"))

1. **os.path.join**:
   * 这是一个用于智能地拼接路径的函数。它会根据不同操作系统的路径分隔符自动插入正确的分隔符（如 / 或 \），以确保路径在不同系统上都能正确使用。
2. **strftime**:
   * strftime 是一个格式化时间的函数，通常使用 datetime 模块中的 strftime 方法。这个函数用于将时间格式化为字符串，根据指定的格式输出时间。
3. **args.result\_dir**:
   * 这是一个变量，表示结果目录的路径。这个路径是你传递给程序的参数之一。
4. **strftime("%Y\_%m\_%d\_%H.%M.%S")**:
   * 这里的 strftime 是用来生成当前时间的字符串表示。具体的格式 "%Y\_%m\_%d\_%H.%M.%S" 会将时间格式化为 年\_月\_日\_小时.分钟.秒 的形式。例如，2024\_09\_01\_12.34.56。

不过，要使这段代码正常工作，strftime 应该是 datetime 模块的一个方法。你需要先导入 datetime 模块并获取当前时间。以下是完整的代码示例：

python

import os

from datetime import datetime

# 假设 args.result\_dir 是你要保存结果的目录

args = type('Args', (object,), {'result\_dir': 'F:\\test'}) # 示例 args 对象

# 获取当前时间并格式化

timestamp = datetime.now().strftime("%Y\_%m\_%d\_%H.%M.%S")

# 拼接路径

save\_dir = os.path.join(args.result\_dir, timestamp)

print(save\_dir)

在这段示例代码中：

* datetime.now() 获取当前的日期和时间。
* strftime("%Y\_%m\_%d\_%H.%M.%S") 将当前时间格式化为 年\_月\_日\_小时.分钟.秒 形式的字符串。
* os.path.join 将 args.result\_dir 和格式化的时间戳拼接成完整的路径。

这样，你就能生成一个包含当前时间戳的目录路径，用于保存文件或结果。

os.makedirs(save\_dir, exist\_ok=True) 是用于创建目录的函数，具体功能如下：

* **os.makedirs**: 这个函数会递归地创建目录路径。如果路径中间的任何目录不存在，它们都会被创建。
* **save\_dir**: 这是要创建的目录路径，通常由 os.path.join 拼接得到。
* **exist\_ok=True**: 这个参数确保如果目录已经存在，函数不会抛出异常。设置为 True 时，函数会忽略已存在的目录，并且不会影响其他目录的创建。

这行代码的作用是确保 save\_dir 指定的目录存在。如果目录不存在，它会创建所有必要的中间目录；如果目录已经存在，它不会进行任何操作，也不会抛出错误。

os.path.join(GSV\_output\_path,'output.wav')可以这么写吗

2024-09-01 04:19:09

是的，os.path.join(GSV\_output\_path, 'output.wav') 这种写法是正确的。它将 GSV\_output\_path 和 'output.wav' 拼接成一个完整的路径。这是生成文件路径的常用方法，能确保在不同操作系统上使用正确的路径分隔符。

我在文件夹F:\AIAnchor里创建了一个python文件，我现在希望这个python 文件有这样一个函数功能，能够使用cmd命令，先进入F:\AIAnchor\GPT-SoVITS-beta0706，然后输入"F:\AIAnchor\GPT-SoVITS-beta0706\runtime\python.exe" "F:\AIAnchor\GPT-SoVITS-beta0706\GPT\_SoVITS\inference\_cli.py" --gpt\_model "F:\AIAnchor\GPT-SoVITS-beta0706\GPT\_weights\Hang-e15.ckpt" --sovits\_model "F:\AIAnchor\GPT-SoVITS-beta0706\SoVITS\_weights\Hang\_e8\_s96.pth" --ref\_audio "F:\AIAnchor\GPT-SoVITS-beta0706\trained\_models\Hang\_vocal\参考音频\_还体现出管理能力，对于活性染料染色缸差的控制。.wav" --ref\_text "F:\AIAnchor\GPT-SoVITS-beta0706\test\参考音频内容.txt" --ref\_language "中文" --target\_text "F:\AIAnchor\GPT-SoVITS-beta0706\test\测试用.txt" --target\_language "中文" --output\_path "F:\AIAnchor\GPT-SoVITS-beta0706\test"，完成后退回到F:\AIAnchor，输入"F:\anaconda3\envs\sadtalker03\python.exe" "F:\SadTalker\SadTalker\inference.py" --driven\_audio F:\AIAnchor\GPT-SoVITS-beta0706\test\output.wav --source\_image "F:\test\_sadtalker\Hang2.png" --result\_dir "F:\AIAnchor\Output\SadTalker\_Output" --still --preprocess full --enhancer gfpgan，运行完之后退出cmd。

2024-09-01 00:53:51

你可以使用 Python 的 subprocess 模块来实现这一功能。以下是一个示例函数，按你的需求执行命令：

python

import subprocess

import os

def run\_commands():

# 第一个目录和命令

work\_dir1 = r'F:\AIAnchor\GPT-SoVITS-beta0706'

cmd1 = [

r'F:\AIAnchor\GPT-SoVITS-beta0706\runtime\python.exe',

r'F:\AIAnchor\GPT-SoVITS-beta0706\GPT\_SoVITS\inference\_cli.py',

'--gpt\_model', r'F:\AIAnchor\GPT-SoVITS-beta0706\GPT\_weights\Hang-e15.ckpt',

'--sovits\_model', r'F:\AIAnchor\GPT-SoVITS-beta0706\SoVITS\_weights\Hang\_e8\_s96.pth',

'--ref\_audio', r'F:\AIAnchor\GPT-SoVITS-beta0706\trained\_models\Hang\_vocal\参考音频\_还体现出管理能力，对于活性染料染色缸差的控制。.wav',

'--ref\_text', r'F:\AIAnchor\GPT-SoVITS-beta0706\test\参考音频内容.txt',

'--ref\_language', '中文',

'--target\_text', r'F:\AIAnchor\GPT-SoVITS-beta0706\test\测试用.txt',

'--target\_language', '中文',

'--output\_path', r'F:\AIAnchor\GPT-SoVITS-beta0706\test'

]

# 第二个目录和命令

work\_dir2 = r'F:\AIAnchor'

cmd2 = [

r'F:\anaconda3\envs\sadtalker03\python.exe',

r'F:\SadTalker\SadTalker\inference.py',

'--driven\_audio', r'F:\AIAnchor\GPT-SoVITS-beta0706\test\output.wav',

'--source\_image', r'F:\test\_sadtalker\Hang2.png',

'--result\_dir', r'F:\AIAnchor\Output\SadTalker\_Output',

'--still',

'--preprocess', 'full',

'--enhancer', 'gfpgan'

]

# 运行第一个命令

os.chdir(work\_dir1)

subprocess.run(cmd1, check=True)

# 运行第二个命令

os.chdir(work\_dir2)

subprocess.run(cmd2, check=True)

if \_\_name\_\_ == "\_\_main\_\_":

run\_commands()

这个函数首先切换到第一个工作目录，运行第一个命令，然后切换回主目录，运行第二个命令。