

智能翻译

1+1=10组
白昱宇 代豫宁

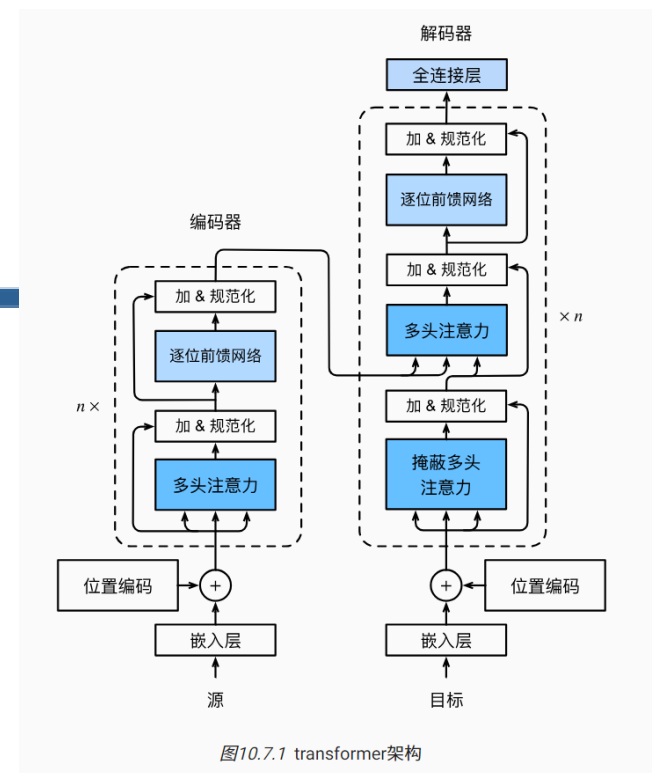


图10.7.1 transformer架构

翻译软件—介绍目录

- 翻译整体流程、框图介绍
- 模块详解
- 程序特点
- 人员分工
- 程序演示



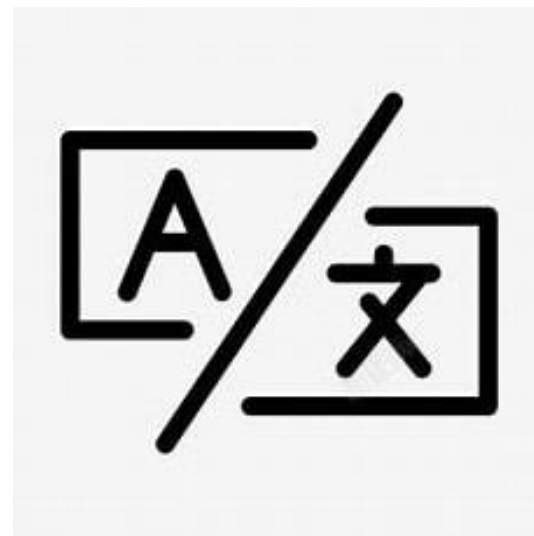
□ 翻译整体流程、框图介绍

□ 模块详解

□ 程序特点

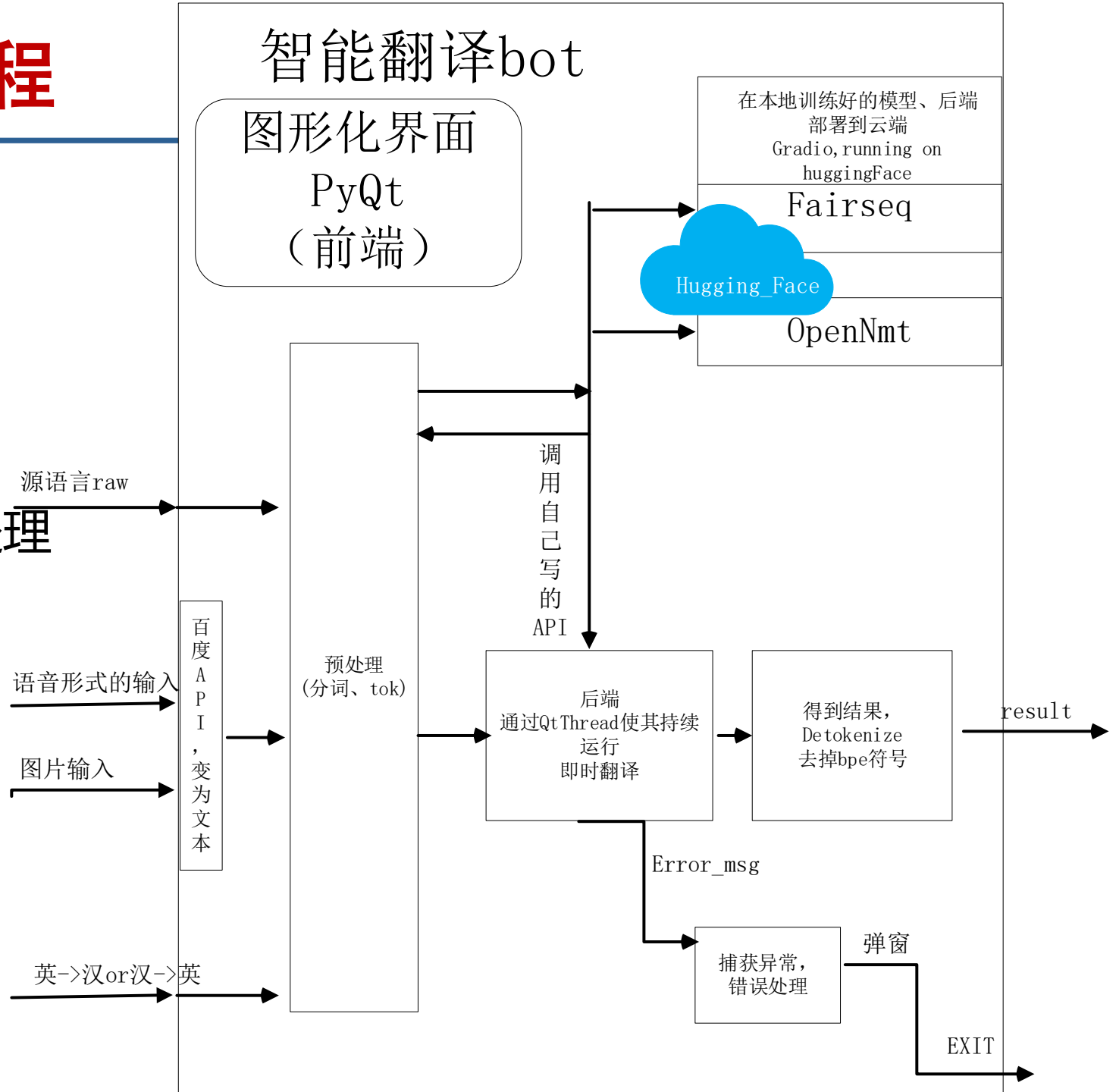
□ 人员分工

□ 程序演示



程序整体、框图、流程

- ❑ 输入形式:
- ❑ 文本/图片/音频文件/实时语音
- ❑ 系统内部:
- ❑ 转文字、token化、推理、后处理
- ❑ 输出:
- ❑ 目标语言

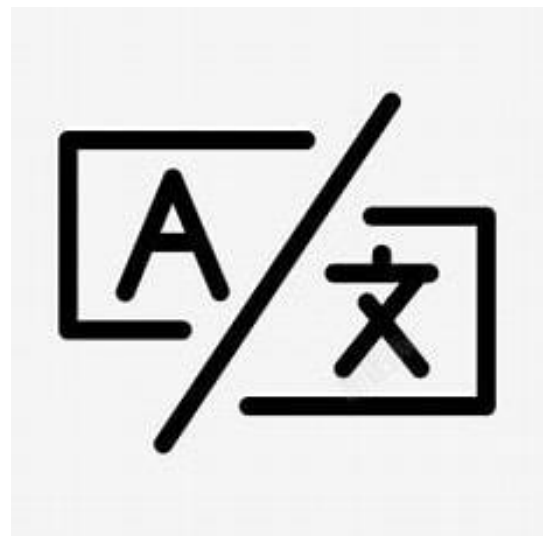


翻译软件

- 翻译整体流程、框图介绍

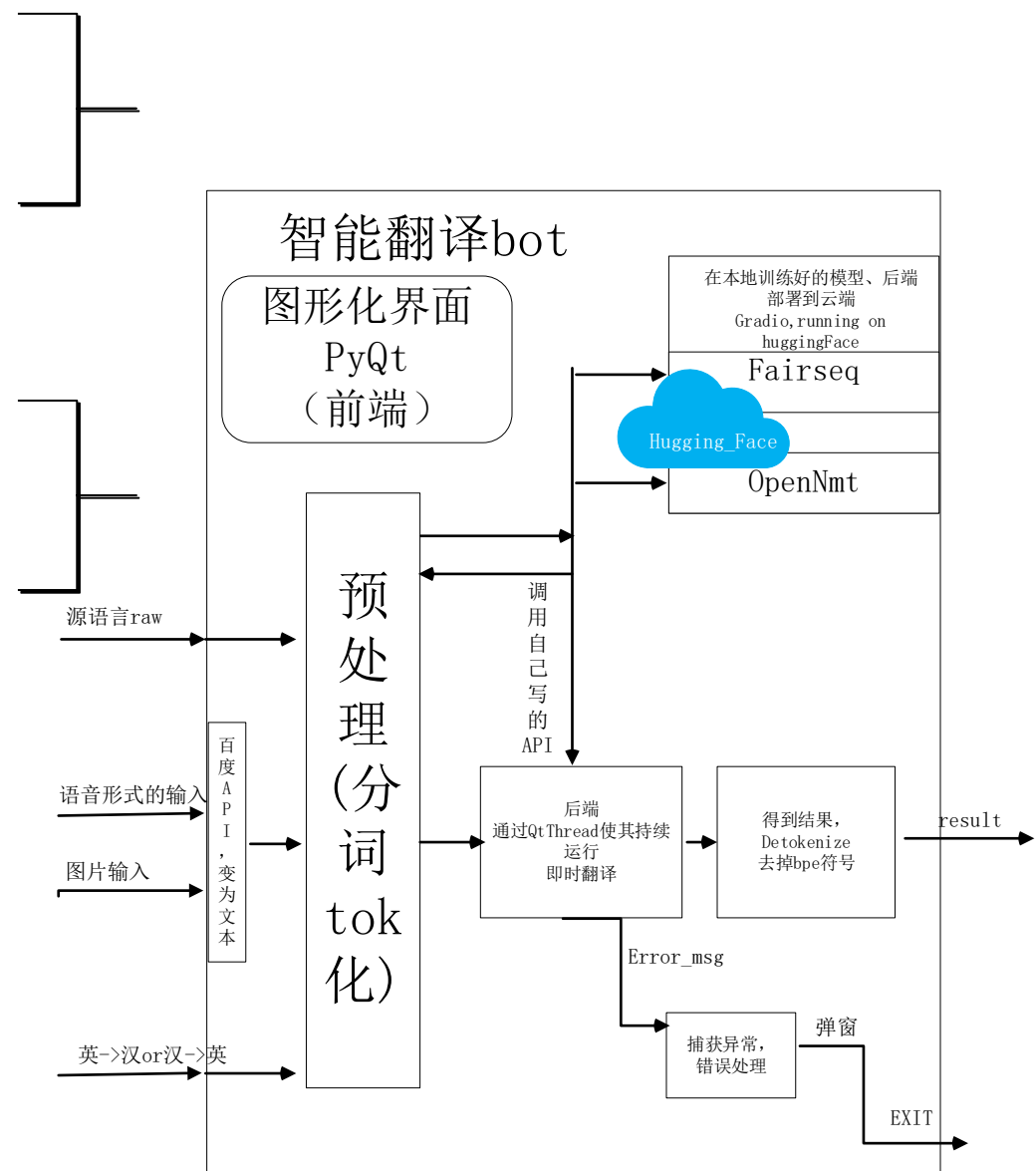
□ 模块详解

- 程序特点
- 人员分工
- 程序演示



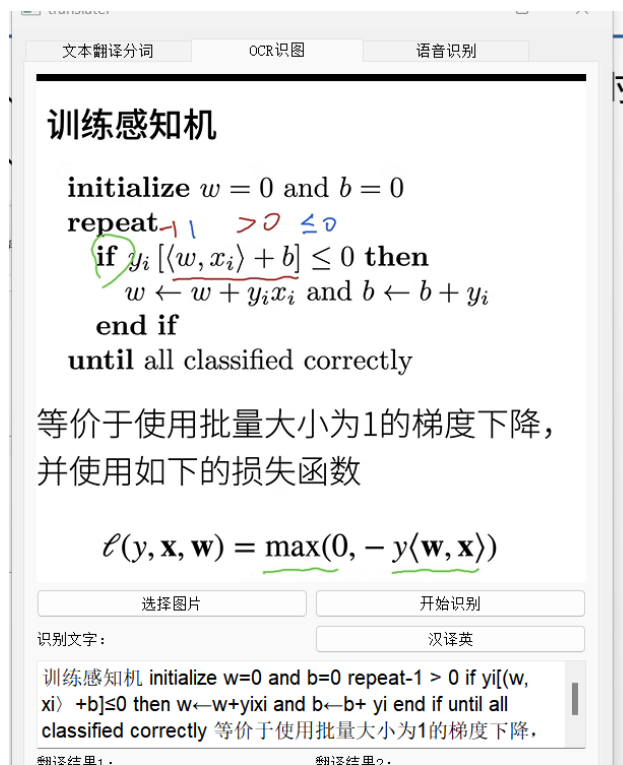
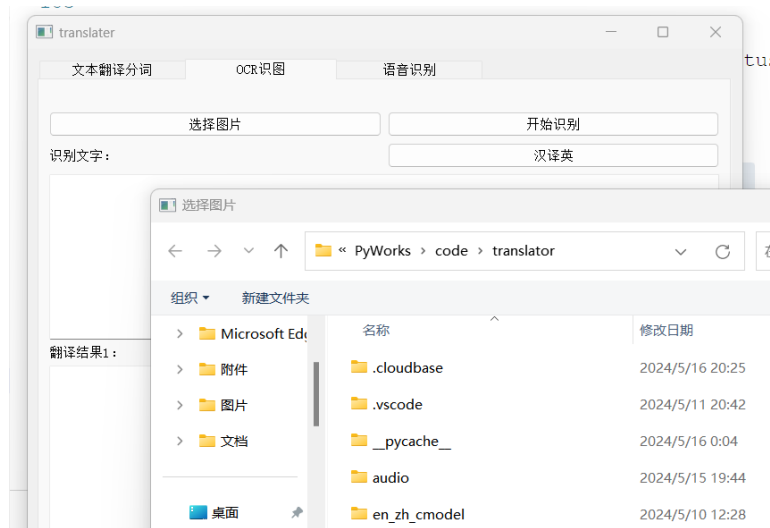
子模块一览

- ❑ 1、输入部分，涉及上传图片、语音实时录制
- ❑ 2、图片、语音转文字，调用百度API
- ❑ 3、文本预处理，分词，token化
- ❑ 4、翻译，实时翻译
- ❑ 5、后处理，detoken
- ❑ 6、输出结果
- ❑ 7、异常处理



输入

- ❑ 1、输入部分，涉及上传图片、语音实时录制
- ❑ 2、图片、语音转文字，调用百度API



内部处理、翻译核心

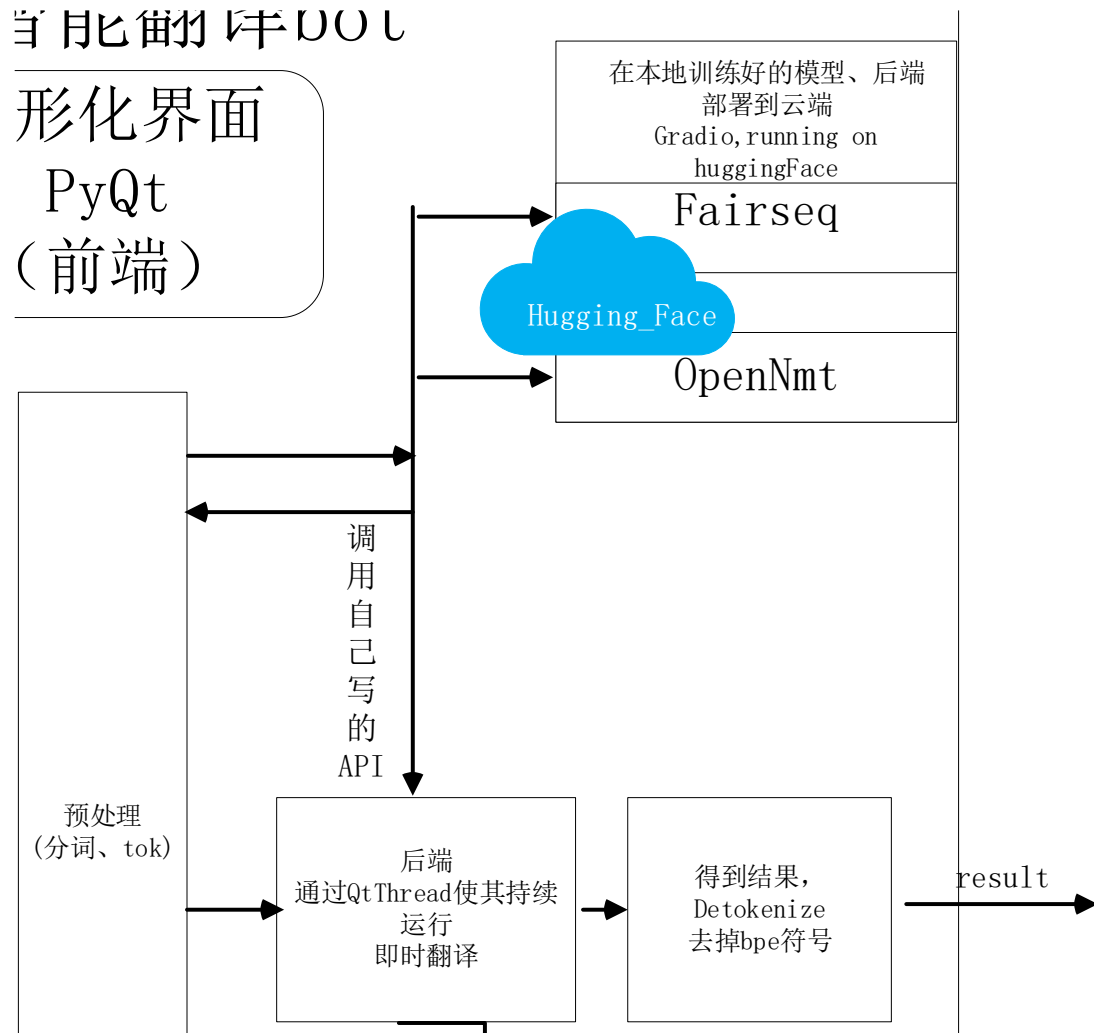
- ❑ 3、文本预处理，分词，token化
- ❑ 4、翻译，实时翻译
- ❑ 5、后处理，detoken

当前翻译DOL

形化界面

PyQt

(前端)



Token、transformer

- ❑ Token, 词元
- ❑ 可以是一个一个单词, 比如 "I eat lemon" -> "I" "eat" "lemon",
- ❑ 按空格分, 比较初级

- ❑ Why Tokenize?
- ❑ 不可能直接把一整句交给模型, 模型眼中的一整句毫无意义

- ❑ Tokenize的过程, 就是先把一整句分为一个一个token, 然后将每个token查频, 与数字索引下标对应, 制成词表 (一个字典), 这样便于学习
- ❑ (模型处理tensor肯定比处理char, string要便捷一些。)

tokenize

- ❑ 一个单词一个单词，拿空格去分词比较初级
- ❑ 先进的模型可能使用更高级的词元化技术,why?
- ❑ 26个字母，却有太多太多单词，vocab词表很容易发生OOV问题，把所有单词放到词表里太占地、且没必要
- ❑ 英语里有词缀一说,fortune,unfortune,fortunately,
old,older,oldest,如果以单词呈现，模型是无法把一种单词的这种形式变换泛化到其他单词的。
- ❑ 所以我们引入subword子词，即把一个单词再细分再细分

分词、token化

- ❑ Spacy, NLTK,
- ❑ Jieba----中文分词
- ❑ sacremoses, subword_nmt...

- ❑ Moses是基于大量规则的分词,

Tokenize on
rules

Let 's tokenize ! Is n't this easy ?

- ❑ that's why we still need bpe after mose

Logs

Build

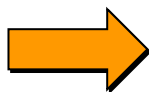
Container

```
[ '我', '吃', '了', '什么' ]
[ '我', '吃', '了', '什' ]
[ '男人', ' ', ' ', '什么', '罐头', '我', '说' ]
[ '我', '看看' ]
[ '我', '看看', '行@@', '不行' ]
[ '而', '我', '却', '伟大', '伟大', ' ' ]
[ '而', '我', '却', '伟大', '伟大', ' ', '萨@@', '达', '撒', '伟大' ]
[ '耀@@', '龇', '伎@@', '俩' ]
[ '我', '吃@@', '了@@', '饭' ]
[ '我', '吃', '了' ]
[ '我', '吃', '了', '很多' ]
[ '高中' ]
[ '高中', '生活', '很', '精彩' ]
[ '我' ]
[ '我', '吃', '柠檬' ]
[ '我' ]
[ '我', '游泳' ]
[ '我' ]
[ '我', '吃', '柠檬' ]
[ '你', '是', '谁' ]
[ '我' ]
[ '我', '吃', '柠檬' ]
[ '你', '是', '谁' ]
[ '你', '是', '谁', ' ? ' ]
[ '我' ]
[ '我', '吃', '柠檬' ]
[ '我' ]
[ '我', '吃', '柠檬' ]
[ '你', '是', '谁' ]
[ '你', '是', '谁', ' ? ' ]
[ '一@@', '二@@', '三' ]
[ '上@@', '山' ]
[ '上@@', '山', '打', '老虎' ]
```

bpe

- 字节对编码 (BPE, Byte Pair Encoding)
- 算法过程不作详细介绍，说结果

WORD	FREQUENCY	WORD	FREQUENCY
deep </w>	3	build </w>	1
learning </w>	3	train </w>	1
the </w>	2	and </w>	1
models </w>	2	deploy </w>	1
Floydhub </w>	1	Build </w>	1
is </w>	1	models </w>	1
fastest </w>	1	in </w>	1
way </w>	1	cloud </w>	1
to </w>	1	Train </w>	1



NUMBER	TOKEN	FREQUENCY
1	</w>	24
2	e	16 - 7 = 9
3	d	12 - 7 = 5
4	l	11
5	n	10
6	i	9
7	a	8
8	o	7
9	de	7
10	s	6
11	t	6

178 @ 136861
179 10 136478
180 i 135337
181 price 134467
182 P@@ 134088
183 page 133666
184 _ 133580
185 Show 132488
186 access 132265
187 without 131921
188 name 131474
189 should 130953
190 o 130689
191 T@@ 130320
192 A 130144
193 K@@ 129510
194 order 128560
195 1 128005

中文

- ❑ BPE 一般适用在欧美语言拉丁语系中，因为欧美语言大多是字符形式，涉及前缀、后缀的单词比较多。
- ❑ 而中文的汉字一般不用 BPE 进行编码，因为中文是字无法进行拆分。对中文的处理通常只有分词和分字两种。
- ❑ 理论上分词效果更好，更好的区别语义。分字效率高、简洁，因为常用的字不过 3000 字，词表更加简短

transformer

- 训练时，把最后一遍bpe处理后的喂给transformer去训练
- ppl,loss...不做赘述

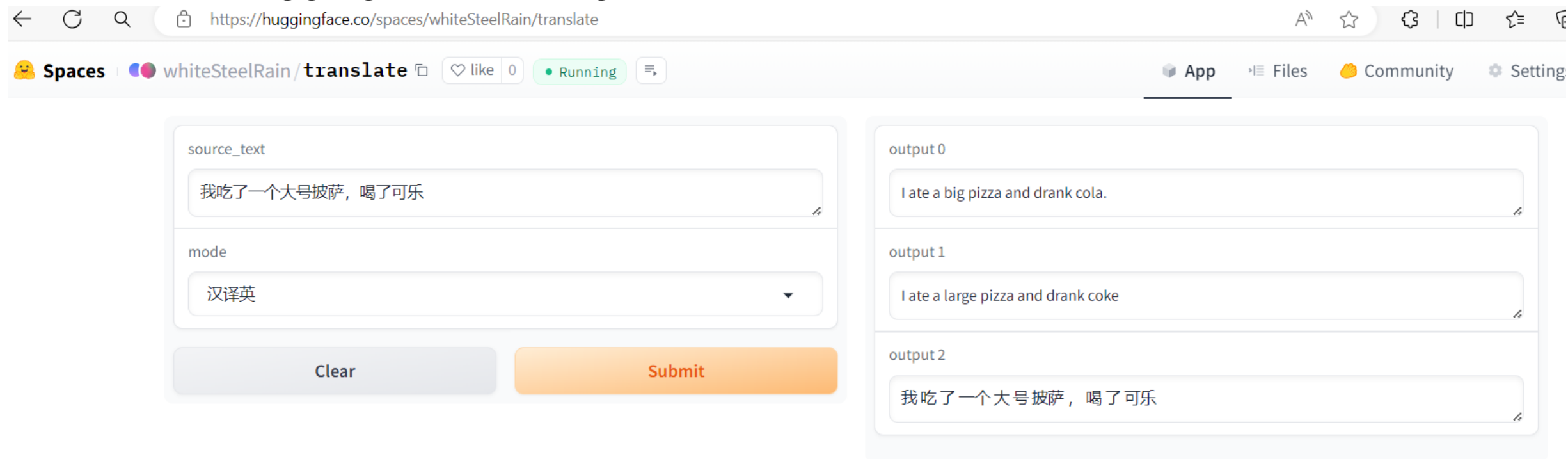
1	epoch 020		valid on 'valid' subset		loss 4.366		nll_loss 2.652		ppl 6.29		wps 50387.3		wpb 8026		bs
2	epoch 021		valid on 'valid' subset		loss 4.36		nll_loss 2.647		ppl 6.27		wps 51992.7		wpb 8026		bsz
3	epoch 022		valid on 'valid' subset		loss 4.361		nll_loss 2.644		ppl 6.25		wps 49009.9		wpb 8026		bs
4	epoch 023		valid on 'valid' subset		loss 4.369		nll_loss 2.65		ppl 6.28		wps 51878.9		wpb 8026		bsz
5	epoch 023		valid on 'valid' subset		loss 4.369		nll_loss 2.65		ppl 6.28		wps 51878.9		wpb 8026		bsz

- 最后汉英/英汉各自跑了60个epoch，dataset选择了大概45w的语料库，n卡V100跑了6天
- 代同学的语料库100w

推理、翻译

□ 使用ctranslate2，把transformer模型转为ctranslate2框架下的，从而可以利用cpu加速transformer的推理、且推理命令及其简洁

□ 部署到hugging face，利用gradio，从而做了一个API



The screenshot shows a web browser interface for a Hugging Face Spaces application. The URL is <https://huggingface.co/spaces/whiteSteelRain/translate>. The application is titled "whiteSteelRain/translate" and is currently "Running". It features a "source_text" input field containing the Chinese text "我吃了一个大号披萨，喝了可乐". Below this is a "mode" dropdown menu set to "汉译英". There are "Clear" and "Submit" buttons. On the right, there are three output fields: "output 0" showing "I ate a big pizza and drank cola.", "output 1" showing "I ate a large pizza and drank coke", and "output 2" showing the original Chinese text "我吃了一个大号披萨，喝了可乐".

source_text

我吃了一个大号披萨，喝了可乐

mode

汉译英

Clear Submit

output 0

I ate a big pizza and drank cola.

output 1

I ate a large pizza and drank coke

output 2

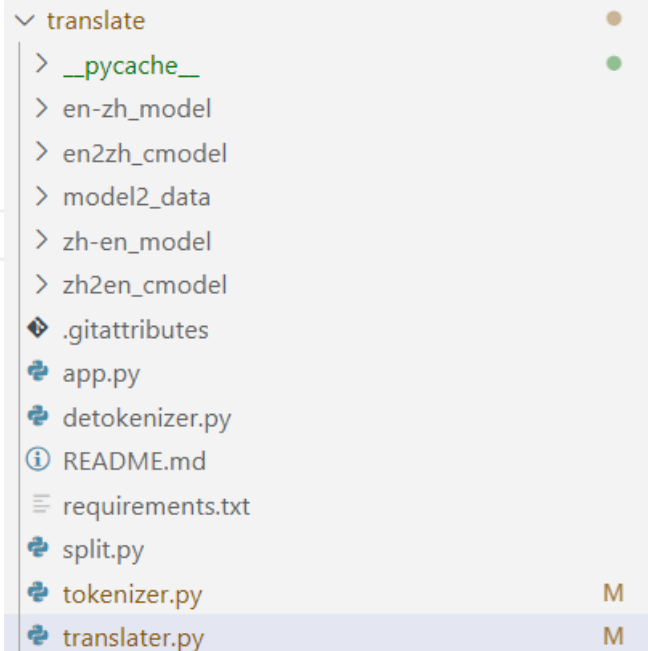
我吃了一个大号披萨，喝了可乐

后端

□ 调用API的方式交互

```
demo = gr.Interface(fn=run,  
                    inputs=["text", gr.Dropdown(["汉译英", "英译汉"])],  
                    outputs=["text", "text", "text"],)
```

```
if self.source_text: # 检查是否有待翻译的文本  
    try:  
        # 执行翻译操作  
        client = Client("whiteSteelRain/translate")  
        #Client("http://127.0.0.1:7860")  
  
        result = client.predict(  
            source_text=self.source_text,  
            mode=self.mode,  
            api_name="/predict"  
        )  
        return result
```



实时翻译

- ❑ 通过thread，使得用户不用每次都按button，自动翻译
- ❑ 主线程是GUI图形界面线程，如果在图形化中直接放入翻译逻辑会卡死
- ❑ 于是设置第二个线程—翻译线程，通过Qthread，mutex.lock和unlock，循环的搭配使得持续运行。

- ❑ 在thread的run方法中，循环内容：

检测文本->发送给hugging_Face

→得到结果->发射信号signal

->在文本框中输出

```
def run(self):  
  
    while self.running:  
        self.mutex.lock()  
        if self.source_text: # 检查是否有待翻译的文本  
            try:  
                # 执行翻译操作  
                client = Client("whiteSteelRain/translate")  
                #Client("http://127.0.0.1:7860")  
  
                result = client.predict(  
                    source_text=self.source_text,  
                    mode=self.mode,  
                    api_name="/predict"  
                )  
            except Exception as e:  
                self.translation_finished.emit("Error", str(e))  
                self.mutex.unlock()  
                #self.stop()  
                break # 退出循环，停止线程  
            self.target_text,self.target_text2,self.source_tokenized_text = result  
            self.tokenization_finished.emit(self.source_tokenized_text) #发送分词结果  
            信号  
            self.translation_finished.emit(self.target_text, self.target_text2) #  
            发送翻译结果信号  
            self.source_text = ""  
        self.mutex.unlock()
```

后处理、去token化

- ❑ 先detokenize, 去除空格等等
- ❑ 应用正则表达式, 去掉bpe符号 (@@)
- ❑ 顺序与我们tokenize的过程是一致的

```
answer_zh_bpe = md_zh.detokenize(tokens, return_str=True)
text = re.sub(r"@@ ", "", answer_zh_bpe)
```

输出结果

- 互为对比，双输出
- 调用大厂API，可作为baseline

Text Translation and Word Segmentation Interface

Input Text: 高中生活很精彩

Word Segmentation Result: 高中 生活 很 精彩

Translation Result 1: High school life is wonderful.

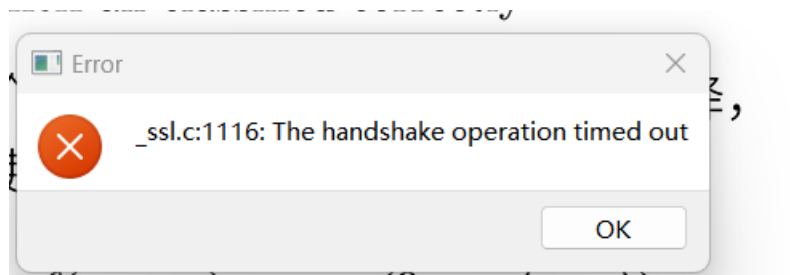
Translation Result 2: high School Life is wonderful

异常处理

Error:



用户没有开启网络代理

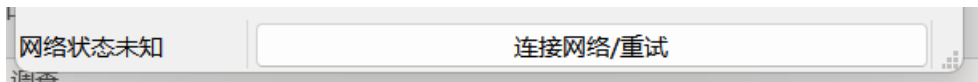


由于潜在的网络质量问题造成的如
handshake operation timed out之类

异常处理

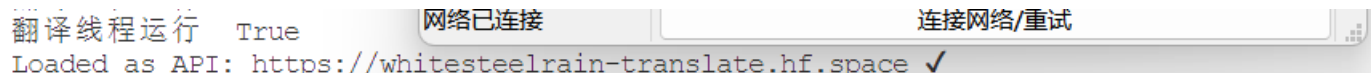
- Solution
- 通过requests库,
- 发送一个get请求, 权当测试;
不超时发送即为连接
同时该检测事件绑定button

```
def check_proxy(self, timeout=5):  
    try:  
        # 尝试向Hugging Face服务器发送一个GET请求  
        requests.get('https://www.huggingface.co', timeout=timeout)  
        return True  
    except requests.exceptions.RequestException:  
        pass  
    return False
```

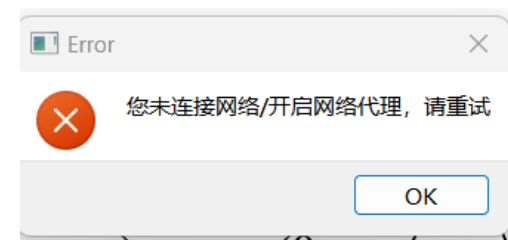
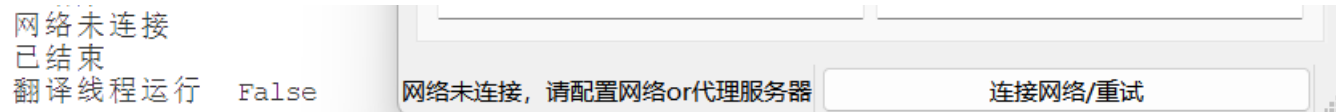


效果:

连接时

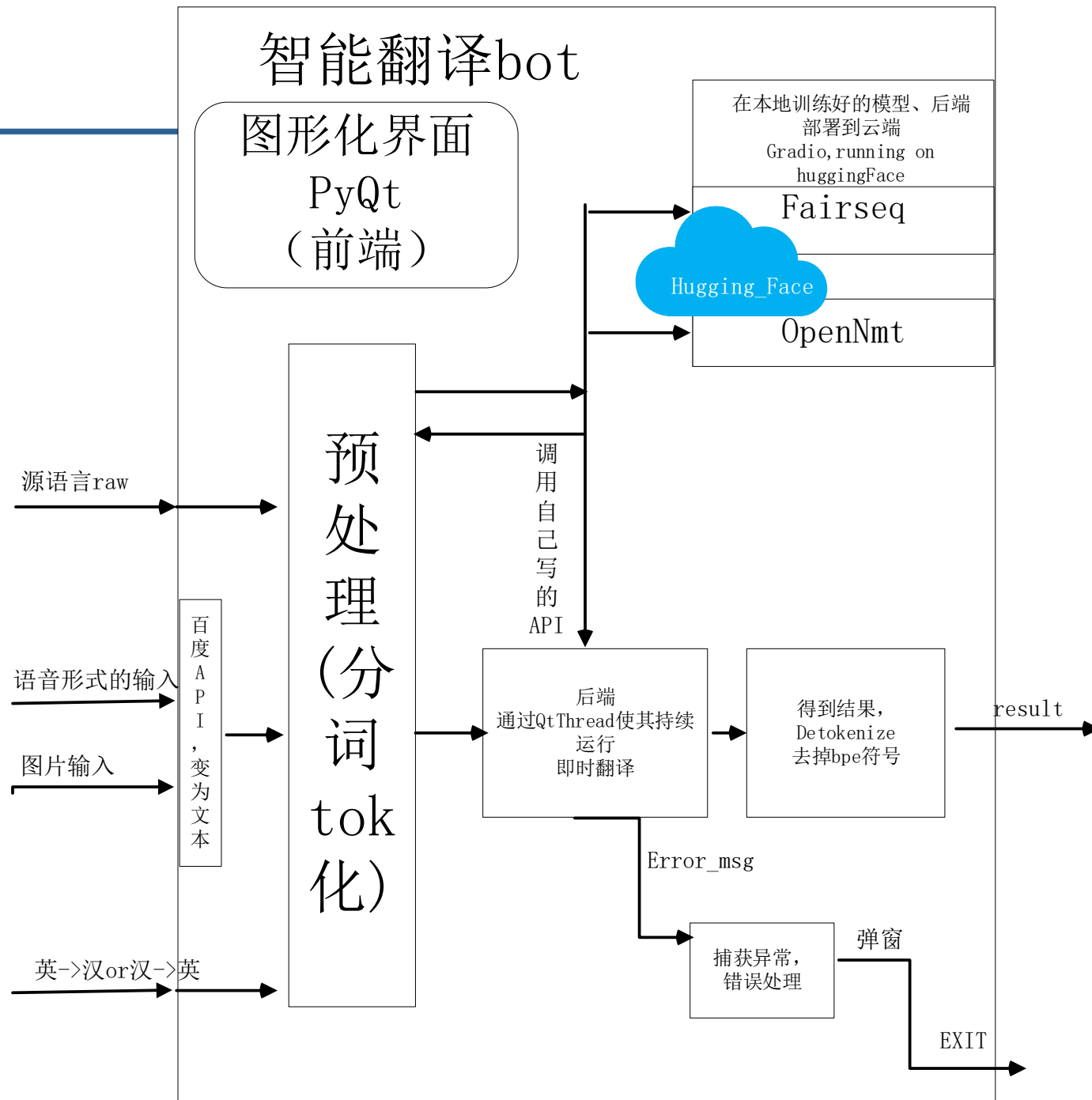


未连接时主动停止线程, 避免卡死,
同时弹窗告警用户



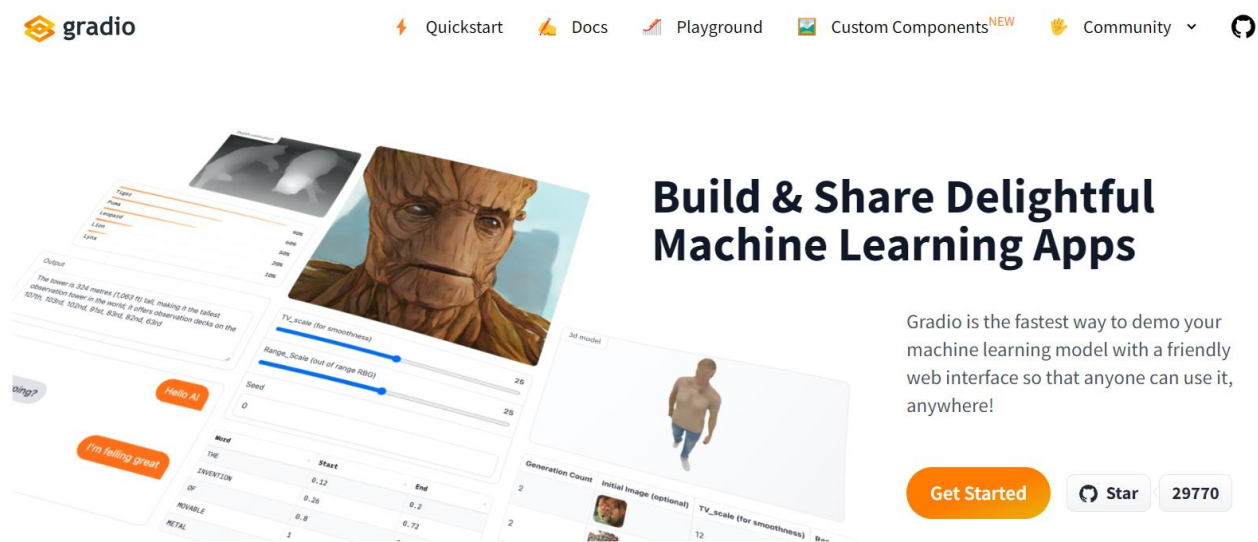
总结

- ❑ 模块化思想，将输入输出、图形化界面、翻译逻辑、预处理、后处理、API调用、异常处理进行模块化，便于维护。



总结

- ❑ 为了节省空间、使软件轻量化，将transformer权重文件、后端大部分逻辑都部署到云端，利用hugging_Face和gradio，实现类似于创建自己的API，并通过调用API的方式来进行后端逻辑部分的处理。



Permanent hosting

Once you've created an interface, you can permanently host it on Hugging Face.

[Hugging Face Spaces](#) will host the interface on its servers and provide you with a link you can share.

总结

- 同时，为了达到学习、对比的效果，特设置两个输出框，互相对比，达到一个baseline的效果，体现模型训练结果在实际应用中的表现。

文本翻译分词 OCR识图 语音识别

输入文字:

分词结果:

高中生活很精彩

高中 生活 很 精彩

翻译结果1:

High school life is wonderful.

翻译结果2:

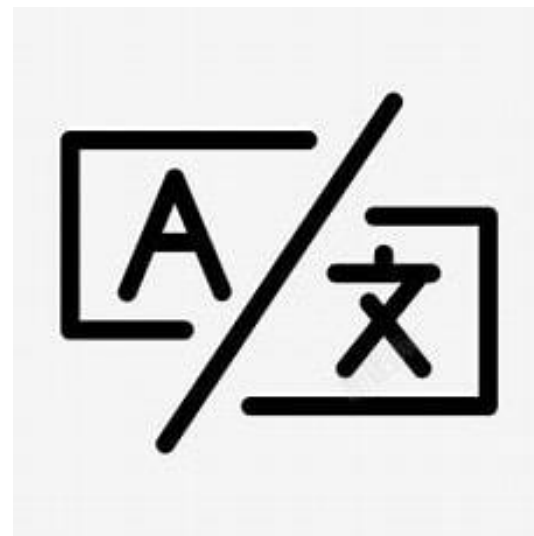
high School Life is wonderful

翻译软件

- 翻译整体流程、框图介绍
- 模块详解

□ 程序亮点

- 人员分工
- 程序演示



亮点 && Special Point

- 1、实时翻译 -----Thread的设计
- 2、多种交互方式，不只是文本 -----支持语音，图片输入
- 3、轻量化设计，后端+模型放在云端，降低对用户端的要求。
- 4、人机结对编程



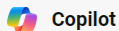
人机结对编程

有时胡说八道、有时很有用，
不建议抛弃GPT和搜索引擎中的任何一个
学知识方面互联网搜索、博客更有优势，
在随便实现个初版框架方面，gpt更有优势



你

我有一些疑惑，希望你能解答一下。我这个fairseq的transformer是在nVidia V100上训练的，训练很久，然后每次推理基本也需要一阵子吧。那么对于没有V100的用户/没有什么GPU的用户，他们怎么办呢



Copilot

✓ 正在为你生成答案...

对于没有高性能GPU（如NVIDIA V100）的用户，他们仍然可以在CPU上运行模型，但是可能会比在GPU上运行慢一些。虽然训练深度学习模型通常需要GPU，但是在CPU上进行推理是完全可行的¹²。

在fairseq中，你可以通过设置 `--cpu` 参数来在CPU上运行模型¹。例如，如果你正在使用 `fairseq-interactive` 或 `fairseq-generate` 进行推理，你可以添加 `--cpu` 参数来在CPU上运行¹。

此外，你还可以考虑使用模型压缩技术，如量化和剪枝，来减少模型的大小和计算需求³。这些技术可以帮助模型在资源有限的设备上运行，如没有GPU的个人电脑或移动设备。

不是，sorry哈，关于前面timer和采样那个，我还是想问。每一次timer时间到头的时候，会执行 `stream.read`，也就是理解为录音吧，那么timer之间的差别不就是代表了采样率吗



我明白你的疑问了，让我们再仔细探讨一下这个问题。

在音频处理中，**采样率**（sampling rate）指的是在一秒钟内对模拟信号进行数字化采样的次数，通常以赫兹（Hz）为单位。例如，一个采样率为44100 Hz的音频流意味着每秒钟会进行44100次采样。

另一方面，**定时器间隔**（timer interval）是指在应用程序中，用于定期执行特定任务（如检查状态、更新UI等）的时间间隔，通常以毫秒为单位。

在你的代码中，使用定时器每10毫秒执行一次 `stream.read`，这确实会在一定程度上影响采样的行为，但它们并不完全相同。以下是两者之间的区别：

- 定时器的作用**：定时器用于周期性地触发一个函数（在这个例子中是 `check_recording`），以检查当前是否正在录音，如果是，就从音频流中读取数据。这主要用于GUI应用程序，以确保在执行I/O操作（如读取音频流）时，UI仍然能够响应用户的操作。
- 采样率的决定因素**：真正的采样率是由音频硬件（如麦克风、声卡）和音频库（如PyAudio）共同决定的。在初始化音频流时，你指定了音频的采样率，这个采样率决定了音频信号被转换成数字信号的速度。
- 定时器间隔与采样率的关系**：虽然定时器的间隔可以看作是程序尝试读取音频数据的频率，但它并不直接决定采样率。音频流的采样率是固定的，而定时器间隔可能只是简单地与采样率对齐。

翻译软件

- 翻译整体流程、框图介绍
- 模块详解
- 程序亮点

□ 人员分工

- 程序演示



人员分工

- ❑ 白：写了很raw的整体初版代码、写了/优化了分词部分的逻辑/代码，使用fairseq框架进行训练，API的接入部分，写了qt界面输入输出（录音实时录制、图片）的代码、异常处理
- ❑ 代：写了很规范的整体代码、代码模块化、使用openNmt框架进行训练、云端hugging_face仓库以及gradio、API自己后端部分的创建者、应用打包者、以及baseline腾讯API应用

展示

