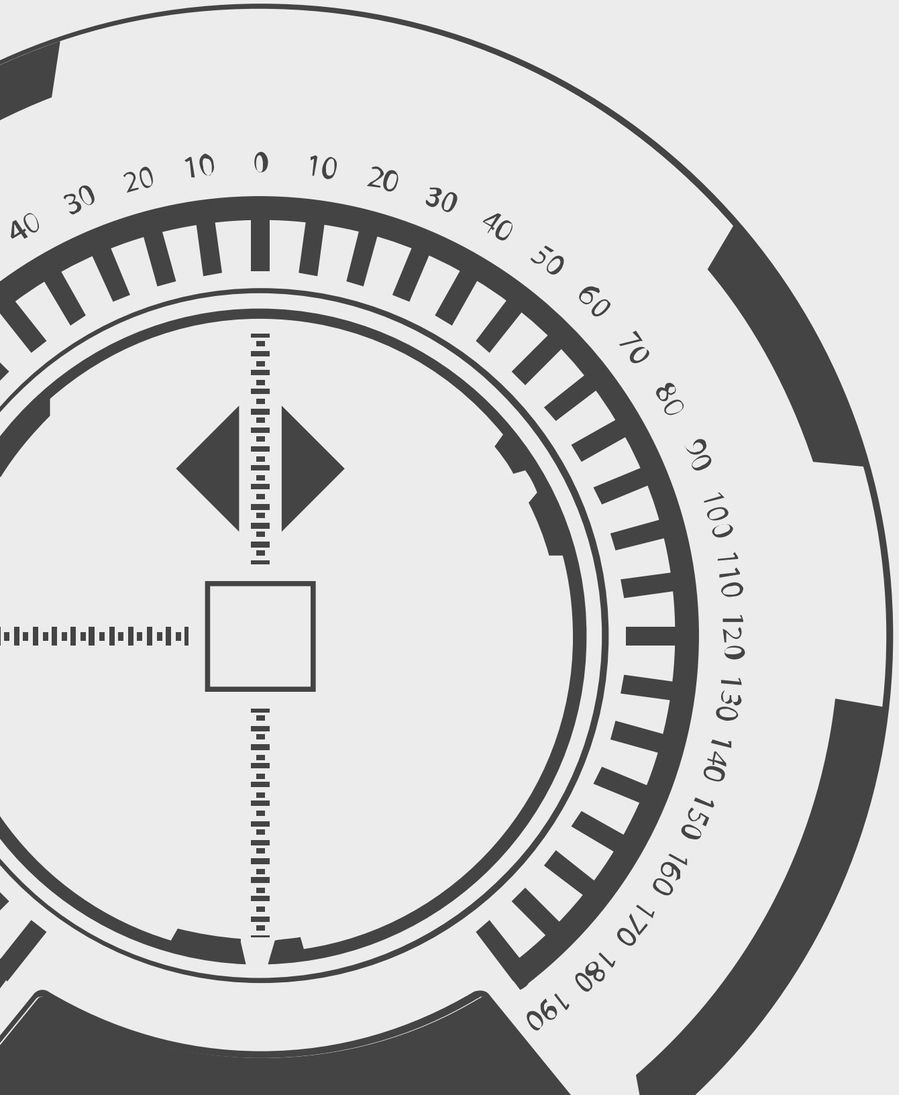


机器翻译

中期报告

聂宇舟 黄乘月 宋祎 唐于程

目录

Part 1

数据预处理

Data Preprocessing

数据集简介-基本属性

- 来源于 The Open Parallel Corpus Project
- 该项目持续从网络中收集翻译文本，整理为数据集

	训练数据	测试数据
句子个数	15886041	4000
句子最大长度	2102	115
至少出现2次单词个数（英文）	300430	4937
至少出现2次词个数（中文）	358108	4796

数据来源：http://opus.nlpl.eu/

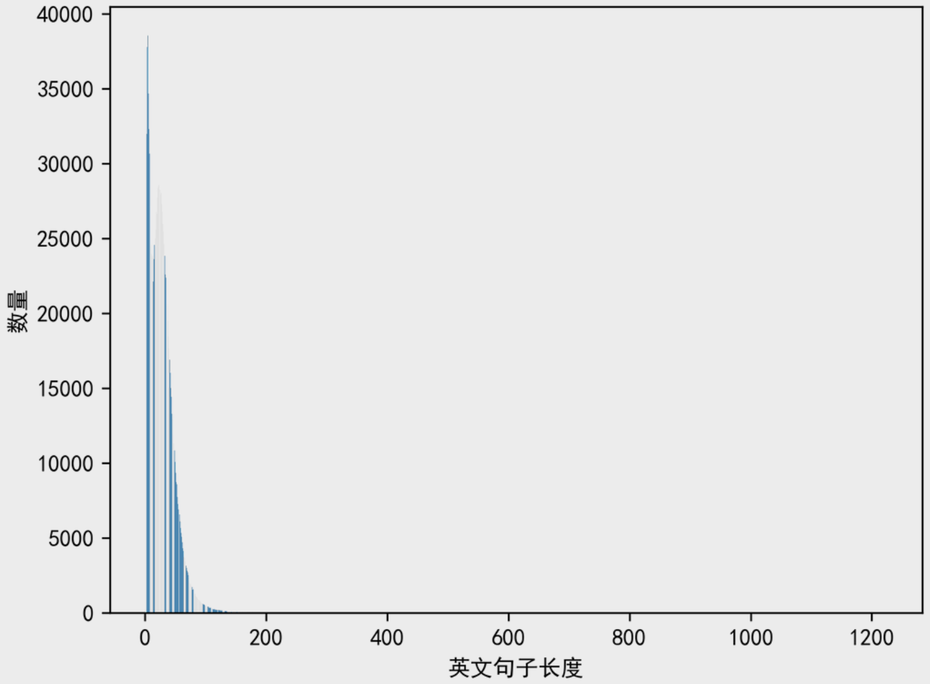
数据集简介-可视化

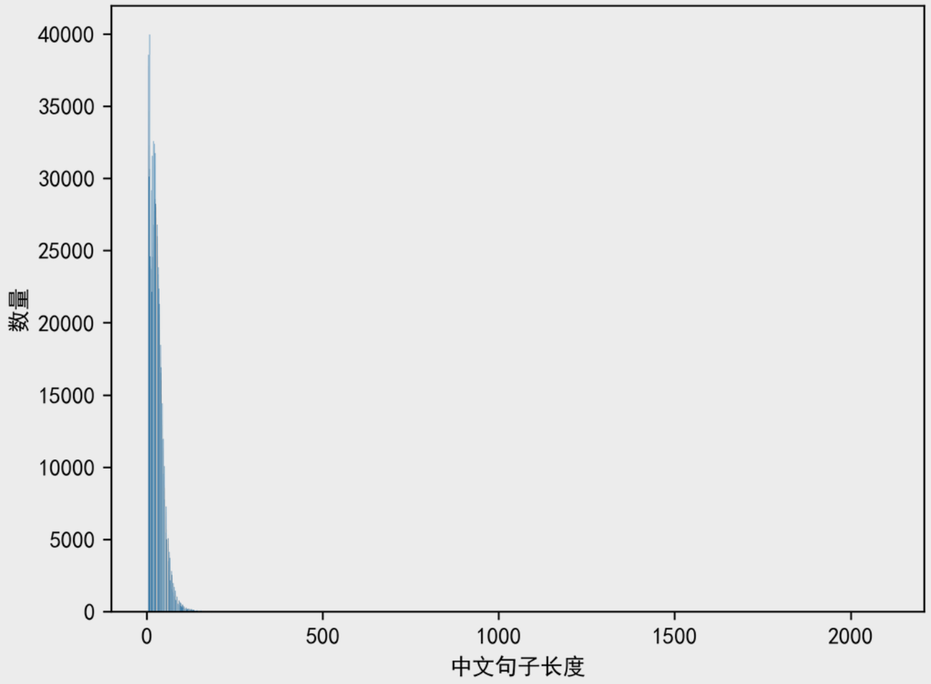
词云图

- 数据中多出现政府、会议、报告等字眼，数据集主题风格较为严肃、正式

数据预处理-数据清洗

- 去掉异常长度文本对





- 英文全部转换为小写，中文将繁体转为简体
  - 大写转小写：torchtext.data.utils.get_tokenizer("basic_english")
  - 繁体转简体：opencc.OpenCC('t2s')

数据预处理-字典构建

- 中文和英文设置两套字典
- 添加特殊符号到字典开头

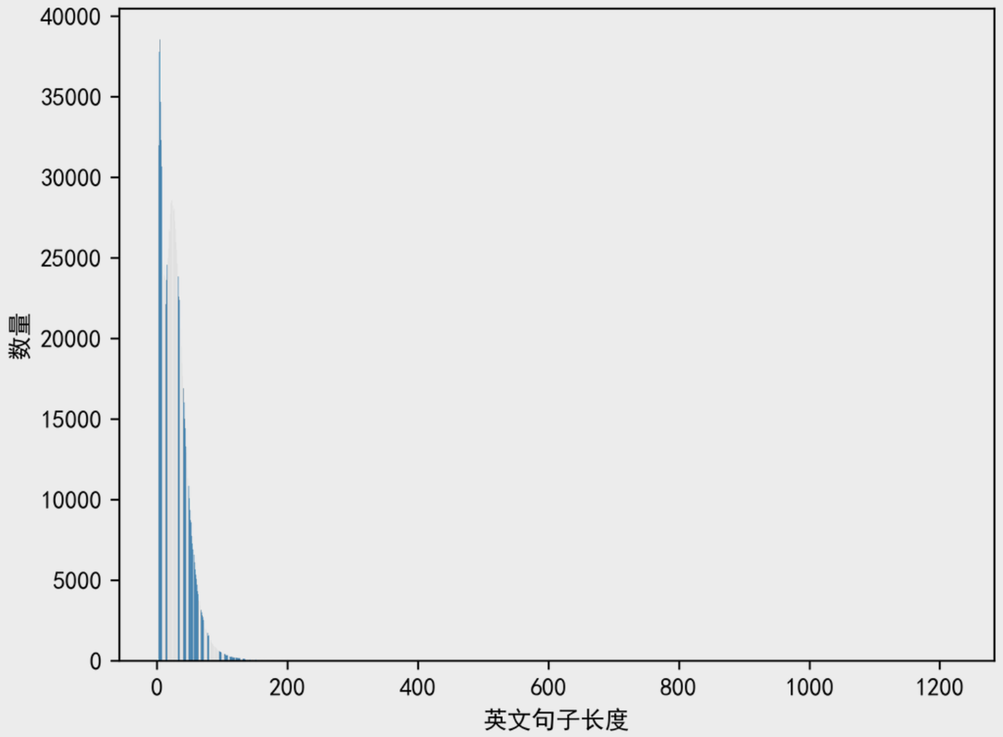
```
vocab_en = build_vocab_from_iterator(map(tokenizer_en, en_iter), max_tokens = args.vocab_size,
                                     specials = ['<unk>', '<pad>', '<bos>', '<eos>'])
vocab_cn = build_vocab_from_iterator(map(tokenizer_cn, cn_iter), max_tokens = args.vocab_size,
                                     specials = ['<unk>', '<pad>', '<bos>', '<eos>'])
```

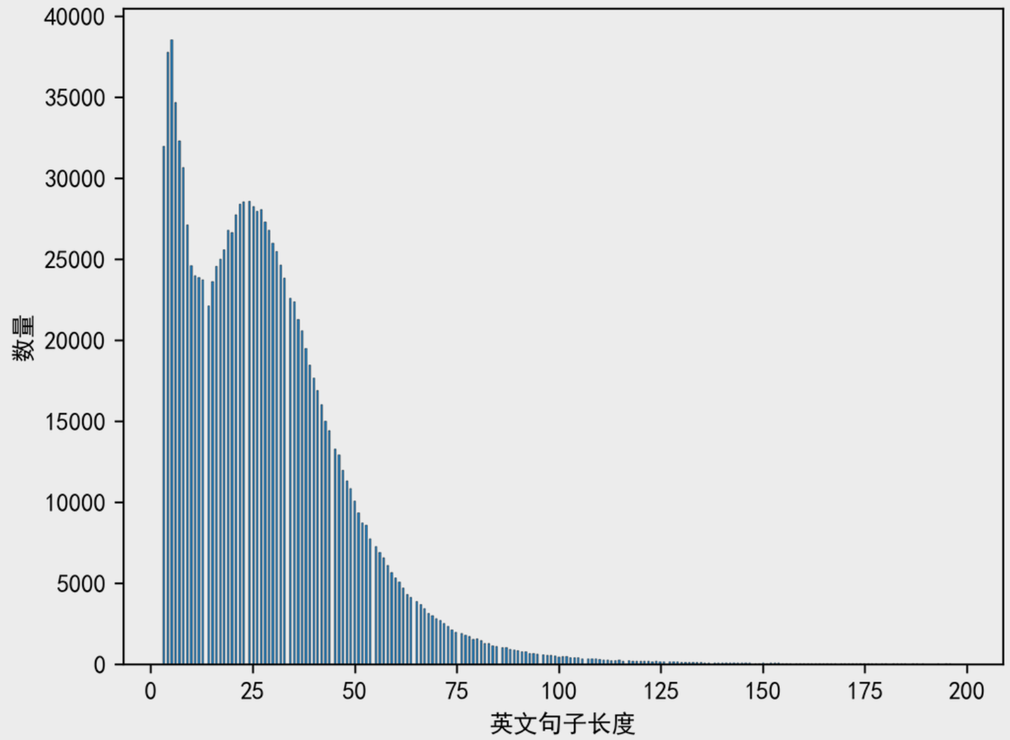
- 设置最大 token 数量为5万
  - 若不限制字典数量，频率高于1的token数量在30万以上，显存过大
  - 大量token为无意义内容

```
'樂': 358084,      '주': 358047,
'힘': 358083,      '죈': 358046,
'휘': 358082,      '졌': 358044,
'회': 358080,      '자': 358041,
'퓔': 358077,      '온': 358037,
'퐸': 358076,      '쎨': 358030,
'쾸': 358064,      '쌈': 358029,
'춘': 358060,      '싘': 358028,
'찼': 358058,      '슜': 358027,
'쬸': 358055,      '슨': 358026,
'쥸': 358048,      '성': 358023,
```

数据预处理-数据集构建

- 对句子进行截断处理





- 选择把句子长度从 128 处进行截断，以保存绝大多数样本的完整信息

数据预处理-数据集构建

- 序列化

Adopted by the Security Council at its 3377th meeting, on 17 May 1994

↓

'adopted', 'by', 'the', 'security', 'council', 'at', 'its', '3377th', 'meeting', ',', 'on', '17', 'may', '1994'

1994年5月17日安全理事会第3377次会议通过

↓

'1994', '年', '5', '月', '17', '日', '安全', '理事会', '第', '3377', '次', '会议', '通过'

- 在头尾和中间分别增加特殊符号 <BOS>、<EOS>、<PAD>

'<bos>', 'adopted', 'by', 'the', 'security', 'council', 'at', 'its', '<unk>', 'meeting', ',', 'on', '17', 'may', '1994', '<eos>'

↓

'<bos>', '1994', '年', '5', '月', '17', '日', '安全', '理事会', '第', '<unk>', '次', '会议', '通过', '<eos>'

数据预处理-数据集构建

- 设置 batch size
  - 统一设置字典容量50000、句子长度128
  - 使用 NVIDIA 3090 24G 显存显卡进行实验

显存占用结果如下

Batch Size	Transformer 128	Transformer 96	Transformer 64	Seq2Seq+Attn
24	21.05G	15.79G	10.53G	5.16G
32	OOM	22.98G	15.32G	9.01G
64	OOM	OOM	OOM	18.02G
96	OOM	OOM	OOM	OOM

采用build包进行流式读取构建迭代器

后续可考虑使用 DataParallel，用多张显卡拆分数据集训练

Part 2

模型构建与训练

Modeling and Training

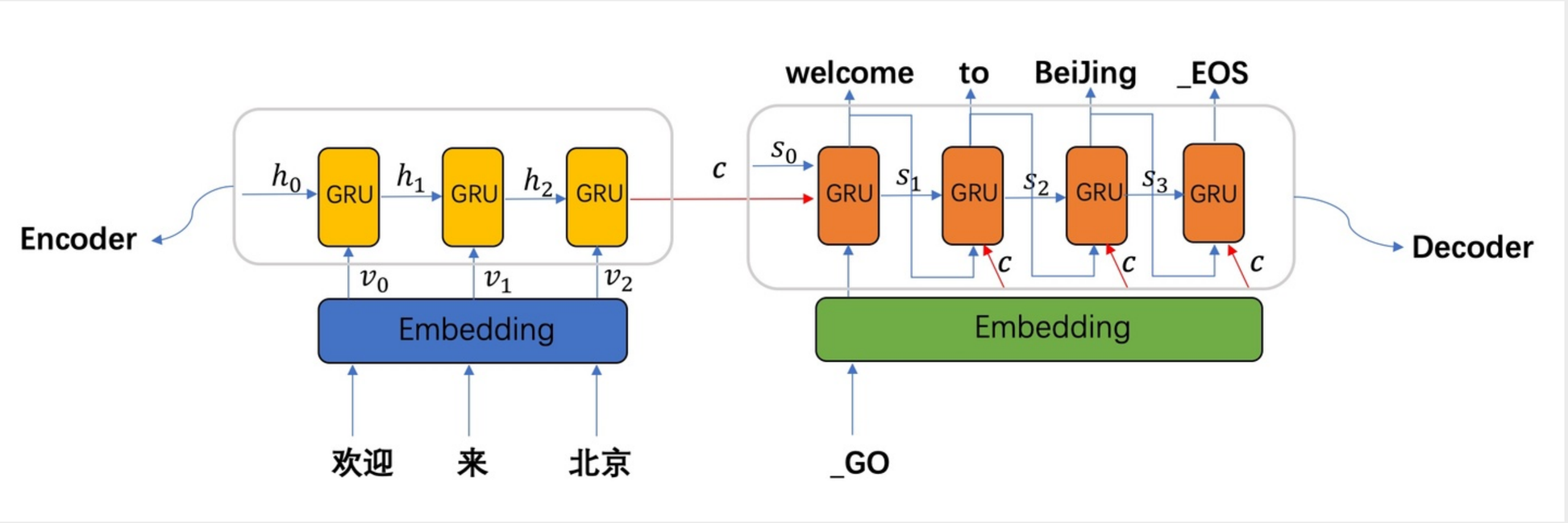
背景介绍

- 2014年，Google Mind发表论文 Recurrent Models of Visual Attention ，这篇论文采用了RNN模型，并加入了Attention机制来进行图像的分类。
- 2015年，Bahdanau等人在论文 Neural Machine Translation by Jointly Learning to Align and Translate 中，将 attention 机制首次应用在 nlp 领域，其采用Seq2Seq+Attention模型来进行机器翻译，并且得到了效果的提升。
- 同年，Luong等人在论文Effective Approaches to Attention-based Neural Machine Translation中提出第二种attention机制，进一步提升了Seq2Seq+Attention模型机器翻译效果。
- 2017年，Google 机器翻译团队发表的 Attention is All You Need 中，完全抛弃了RNN和CNN等网络结构，而仅仅采用Attention机制来进行机器翻译任务，并且取得了很好的效果。

模型构建-Seq2Seq

模型介绍

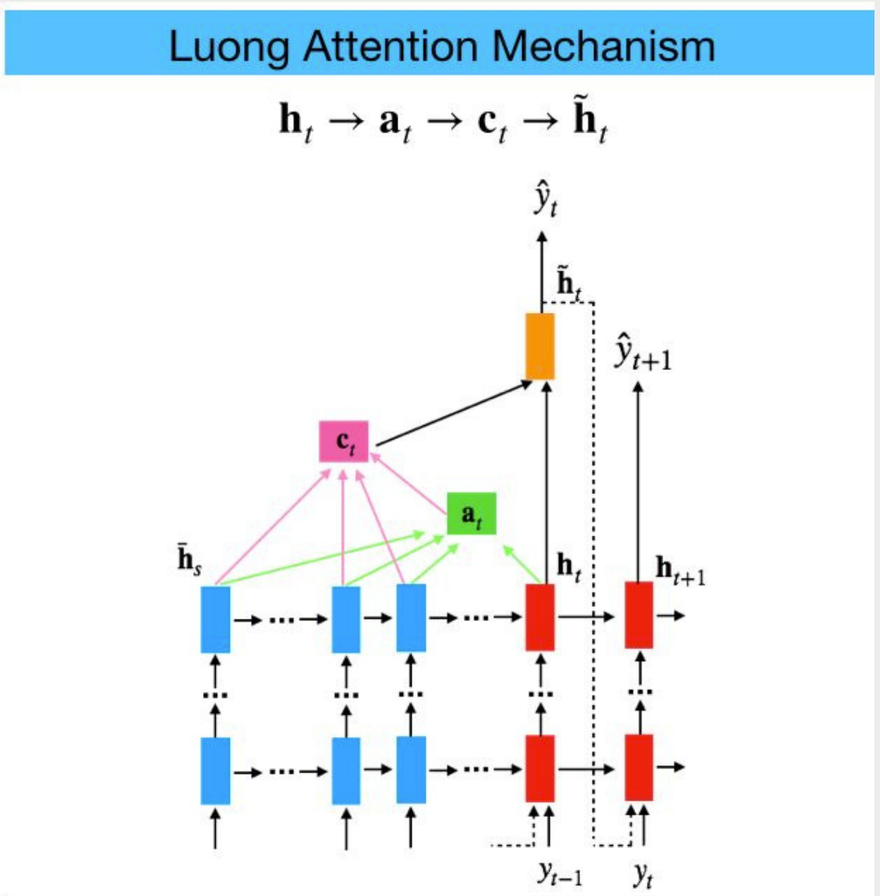
Seq2Seq为Encoder-Decoder结构的网络，输入和输出都是序列，Encoder和Decoder都是GRU结构。



模型构建-Seq2Seq+Attention

模型介绍

引入attention机制，Encoder中每一个输出在解码过程中权重不同。



$$a_t(s) = \frac{\exp\left(\text{score}\left(h_t, \bar{h}_s\right)\right)}{\sum_{s'} \exp\left(\text{score}\left(h_t, \bar{h}_{s'}\right)\right)}$$

$$c_t = \sum a_t \bar{h}_s$$

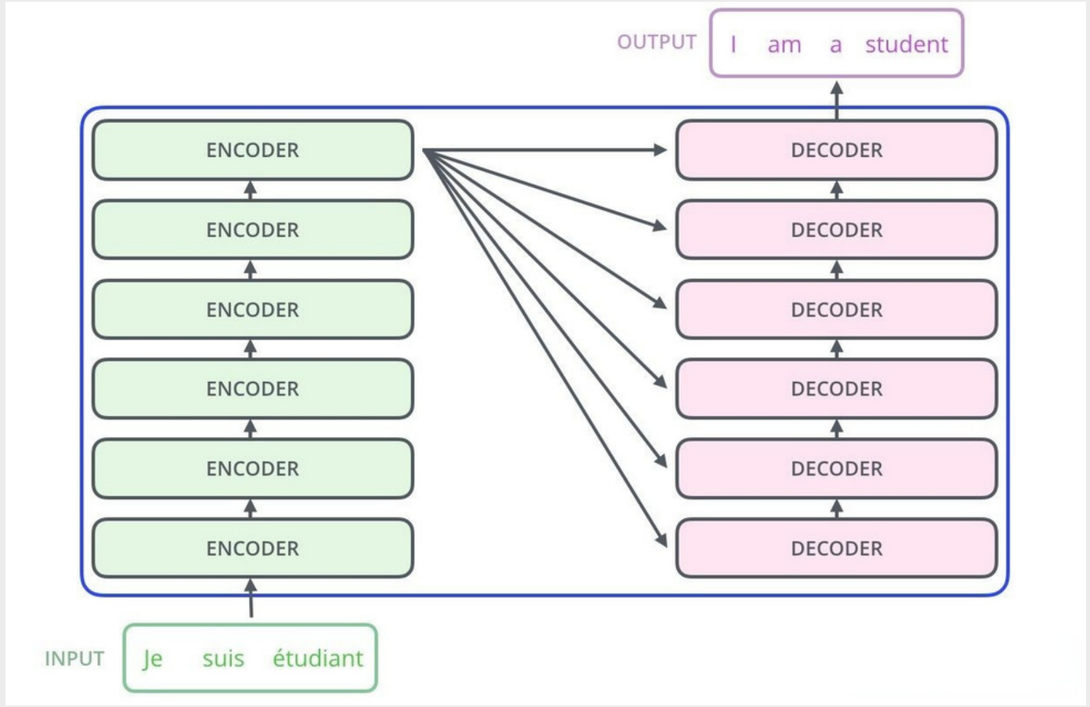
$$\tilde{h}_t = \tanh(W_c\,[c_t; h_t])$$

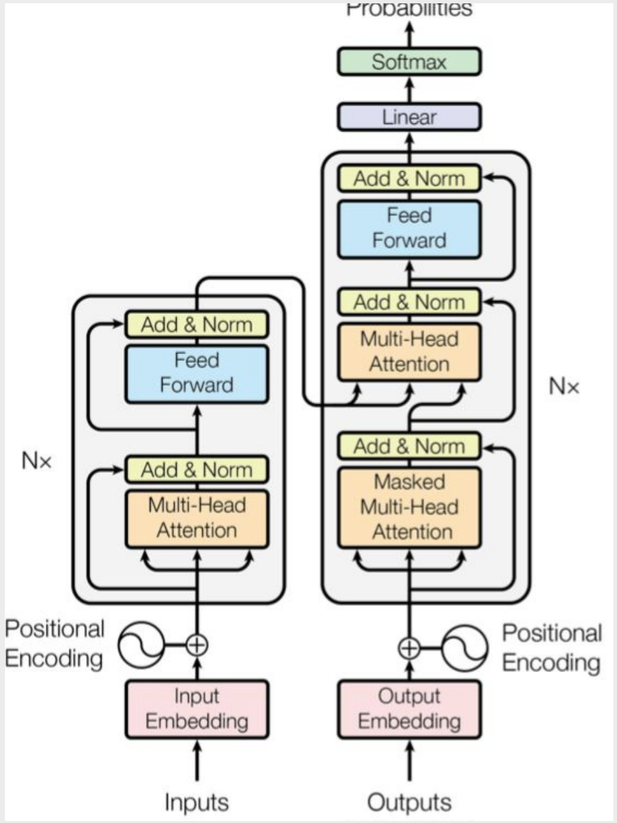
$$\text{score}\left(\boldsymbol{h}_t, \overline{\boldsymbol{h}}_s\right) = \begin{cases} \boldsymbol{h}_t^\top \overline{\boldsymbol{h}}_s & \text{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W}_a \overline{\boldsymbol{h}}_s & \text{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_a \left[\boldsymbol{h}_t; \overline{\boldsymbol{h}}_s\right]\right) & \text{concat} \end{cases}$$

模型构建-Transformer

模型介绍

Transformer抛弃了传统的CNN和RNN，整个网络结构完全是由Attention机制组成，将multi-headed的机制增加到self-attention上，以提高attention 的效果。





模型构建-Transformer

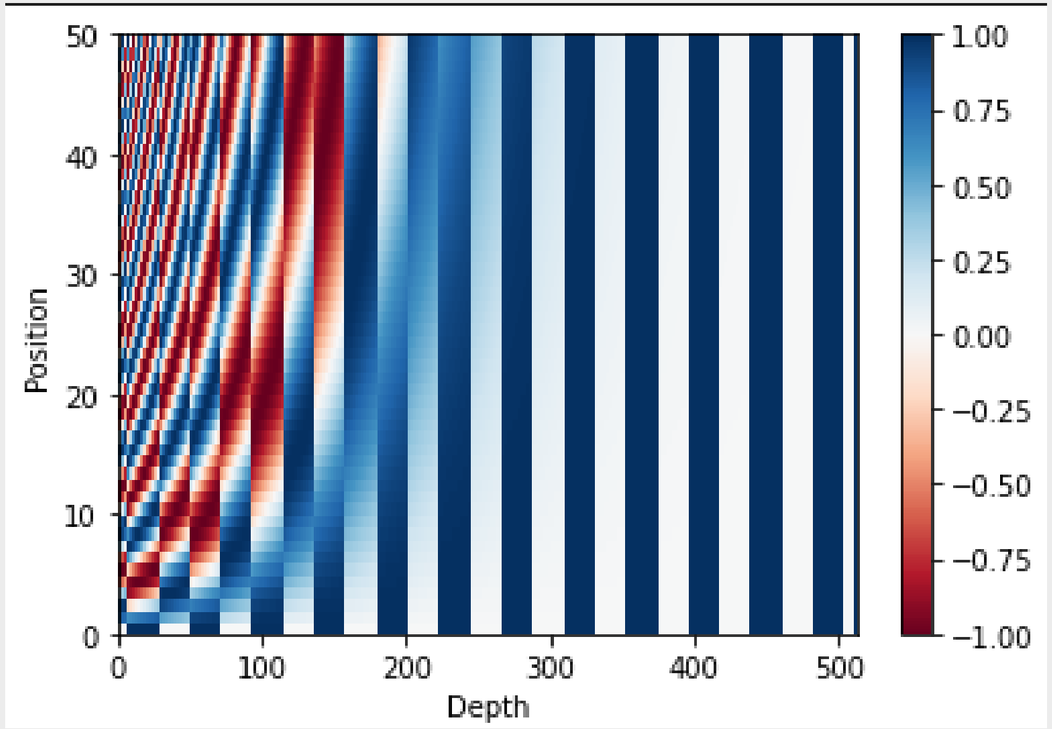
位置编码

- 由于 self-attention 不清楚单词之间的位置信息，需要在 embedding 层加入位置编码
- Transformer采用绝对位置编码，即

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{\text{model}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{\text{model}}}\right)$$

- 在 d_model=500, max_length = 50 情况下可视化如图所示



模型构建-Transformer-XL

选用动机

- 原始 Transformer 需要固定编码长度，无法建模超过固定编码长度的文本，且长句编码效果差；本数据集存在超长句情况
- 原始 Transformer 对于 self-attention 计算复杂度为$\mathcal{O}\left(L^2\right)$
  Transformer-XL 使用层级的思路提升了速度。

模型改进

- hidden state reuse
- 使用 relative positional encoding 相对位置编码

模型训练-训练细节

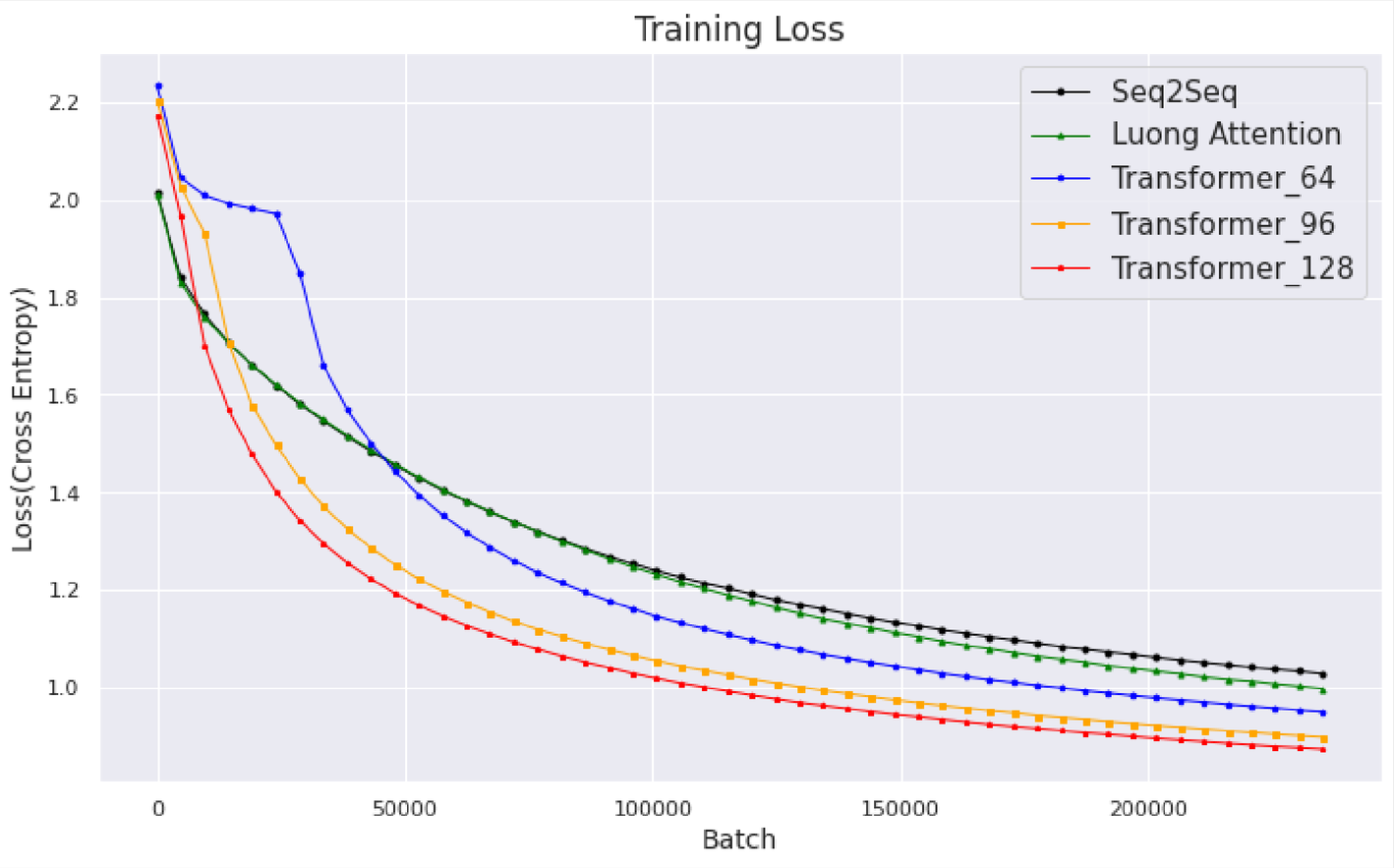
- 模型在 GPU: RTX3090 上进行训练，所用时间如图所示

模型	训练时间 per epoch
Transformer 128	**30.4h**
Transformer 96	**24.9h**
Transformer 64	**21.4h**
Seq2Seq(+Attn)	**9.7h**

- 模型训练时间和参数量基本成正比
- 由于时间问题，每个模型只训练 2 个epoch
- 截止到周五早上，Transformer-XL 还没有完成训练

模型训练-训练结果

- 各模型的训练loss变化图如图所示



- 模型训练误差和模型复杂度基本成正比
- 训练误差还没有看到收敛迹象

Part 3

模型评测

Model Evaluation

模型评测-BLEU对比

指标介绍

BLEU 评分是由 Kishore Papineni 等人在 2002 年的论文 BLEU: a Method for Automatic Evaluation of Machine Translation 中提出的。

BLEU 编程实现的主要任务是对候选翻译(candidate)和参考翻译(reference)的 n 元组进行比较, 并计算相匹配的个数。BLEU有以下特点：

- BLEU得分与具体语言无关
- 匹配个数与元组的位置无关，匹配个数越多, 表明候选翻译的质量就越好。
- 取值为[0,1],完美匹配得分为1。

本项目计算BLEU采用的是nltk.translate.bleu_score模块的corpus_bleu函数。

模型评测-BLEU对比

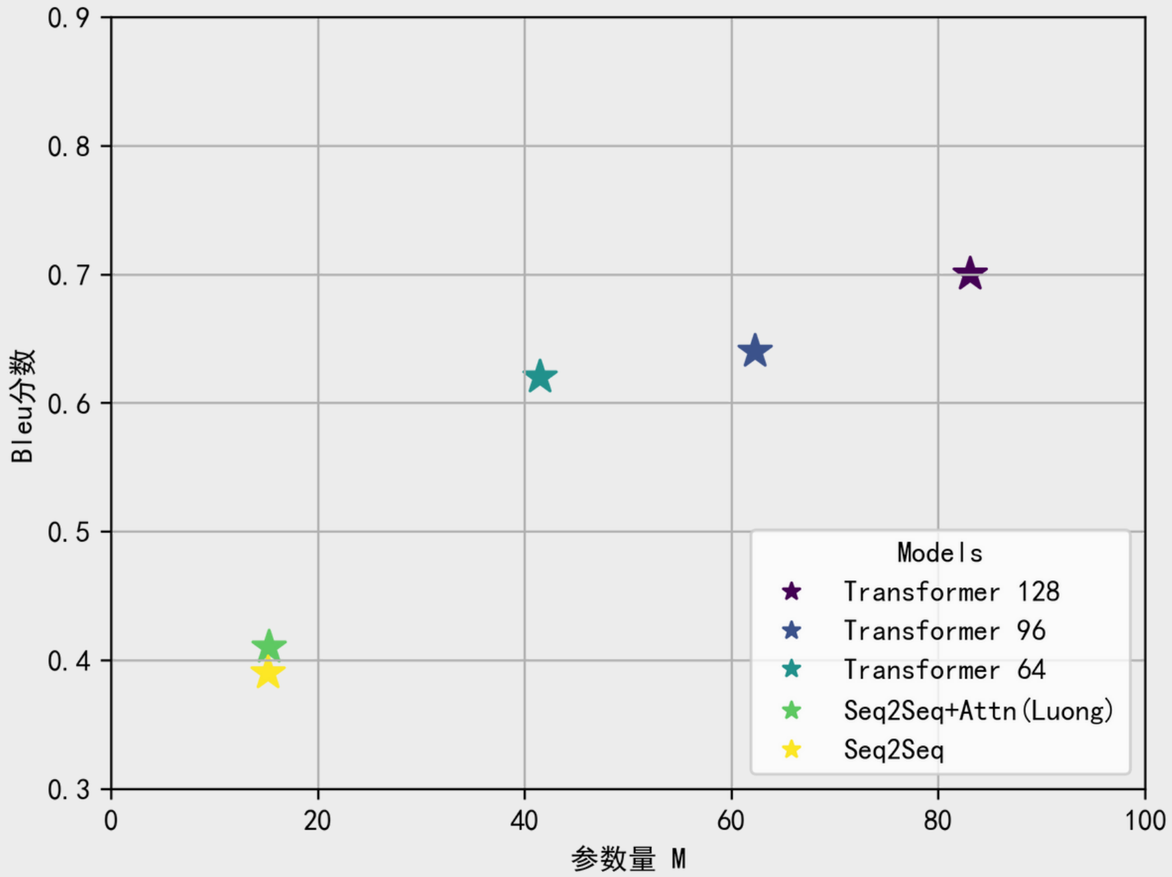
具体结果

模型	测试集 loss	Bleu 分数
Transformer-XL (部分训练)	0.8	0.67
Transformer 128	0.78	0.7
Transformer 96	0.93	0.64
Transformer 64	0.94	0.62
Seq2Seq+Attn (Luong)	2.34	0.37
Seq2seq	2.37	0.36

- Loss和Bleu分数的变化情况基本保持一致
- 随着模型复杂度增长，效果在不断上升

模型评测-BLEU对比

效果与参数量的关系



- 随着参数增长，Bleu分数仍在在不断增加
- 当前选用模型仍未充分利用数据集信息

模型评测-实际结果

- 翻译正确

```
Input: This model also seems to work well on complex sentence.
Predicted translation: 这一模型似乎还很好地在复杂的句子上工作。
```

- 同义词错误

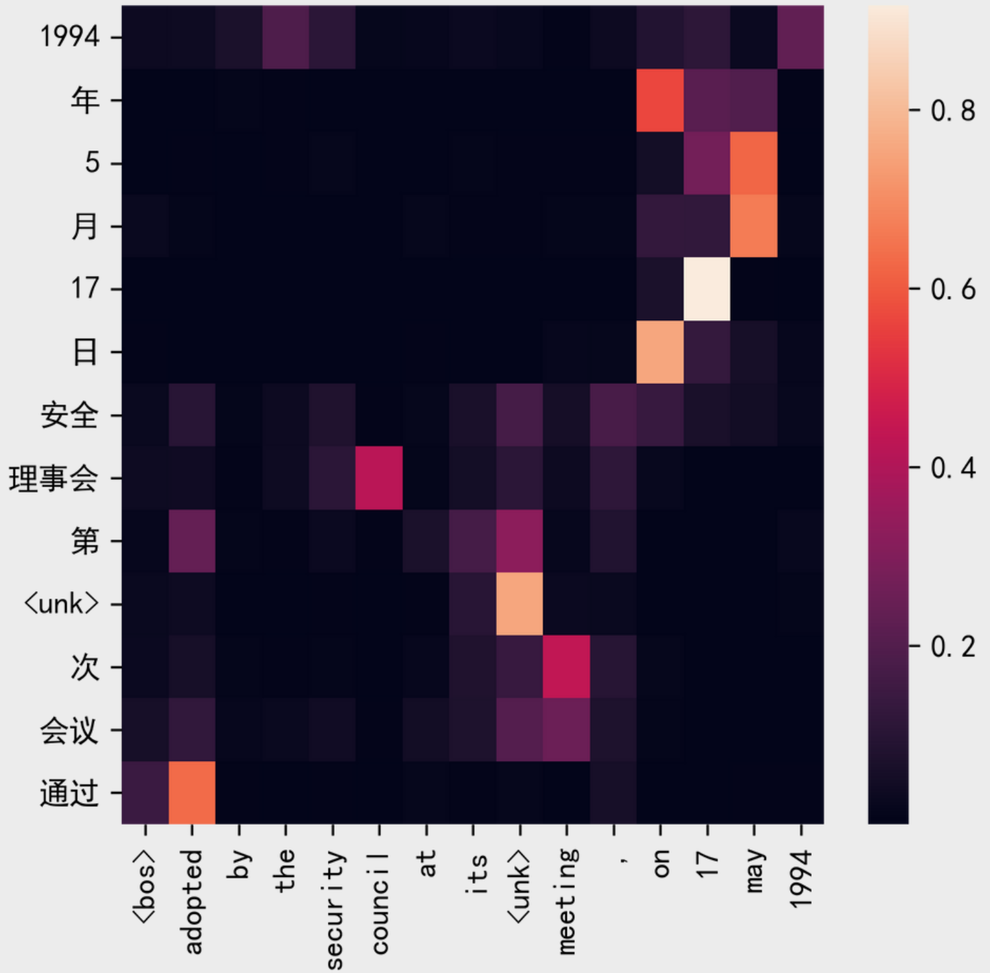
```
Input: I wonder how to improve the performance of this model.
Predicted translation: 我想知道如何改善这一模型的绩效。
```

- 翻译时出现英文结果

```
Input: Dark flame master
Predicted translation: Dark Park Pow General
```

成因：中文样本中存在大量英文

模型评测-中英文注意力对应情况



- 图中可以发现注意力的对应关系相当准确
- 注意力可以实现一对多的关系

Part 4

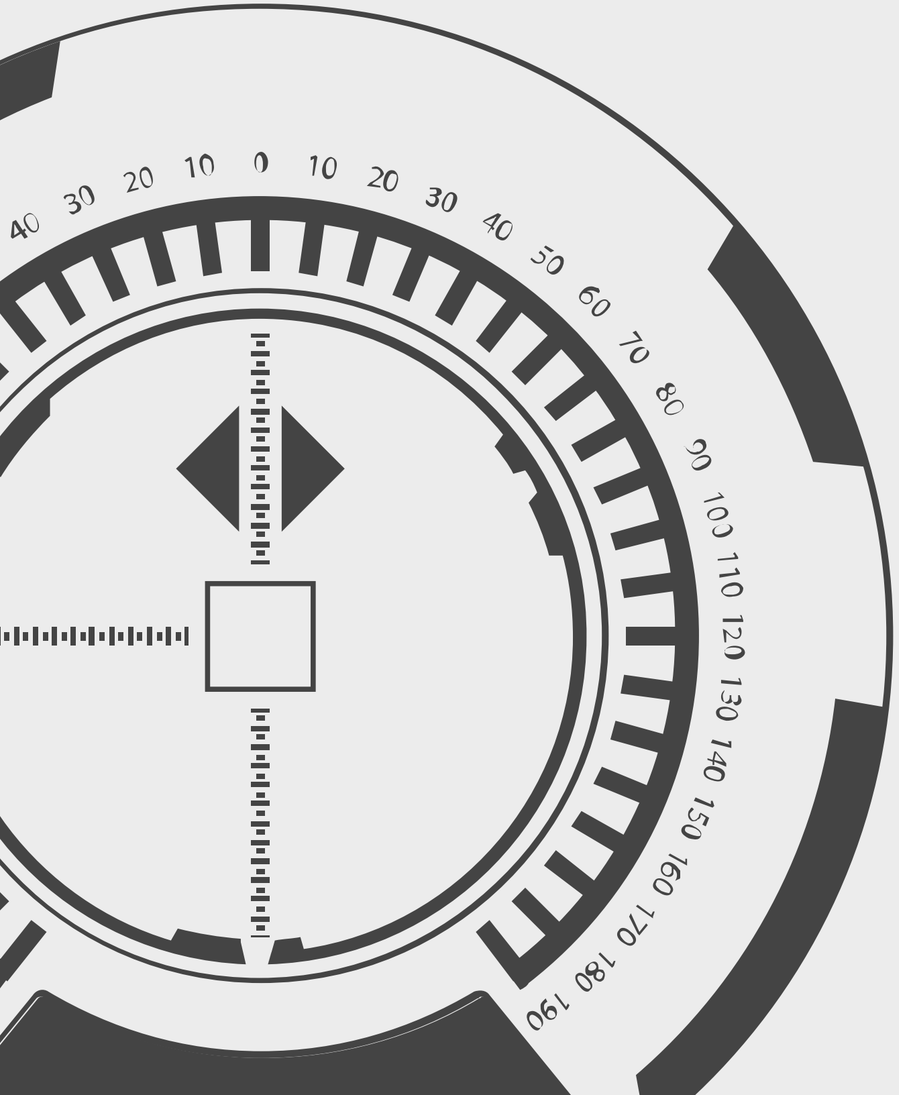
不足与后续展望

Future Work

不足与后续展望

- 当前不足：
  - 数据集的清洗工作不够完全
  - 相较于巨大的数据量，当前模型参数量可能仍然不足以充分学习数据集的信息

- 后续工作：
  - 进行数据清洗工作，考虑中文词库中的英文问题
  - 使用 DistributedDataParallel 或者 DataParallel 增大显存使用
  - 使用大规模预训练模型，比如BERT、CLIP等，并对下述方法进行对比
    - 直接在测试集上进行zero-shot
    - 在训练集用少量数据（few-shot）进行fine-tune，而后再在测试集上预测
    - 在训练集利用完整数据集进行fine-tune，而后再在测试集上预测



谢谢大家

Thank you for watching

聂宇舟 黄乘月 宋祎 唐于程