

Stream Net

吴立凡 中国科学技术大学



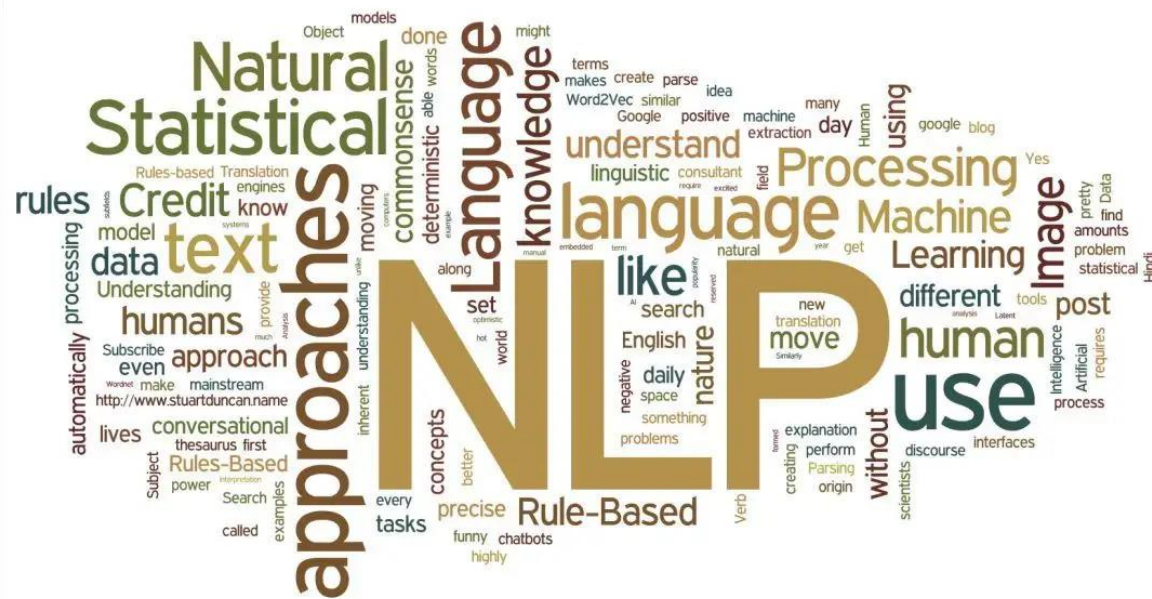


目录

相关研究
模型框架
代码实现
实验结果
总结展望

摘要

- 在这个项目中，我通过对问题进行分类，将复杂多样的自然语言理解问题简化为子问题，构建了一套解决自然语言处理问题的框架与流程。
- 我的方法是使用复合模型来处理复杂多样的自然语言处理问题，后续的实验结果也证明了这种方法的有效性。





相关研究



长文本的阅读理解问题

阅读理解问题是nlp研究中的一个很重要的领域，一种可能的提问形式是长篇文章的阅读理解问题。它给出了一个上下文，一个或多个问题，以获得问题的答案：

Context: "Still searching for their first win. In the second quarter, Dallas increased its lead as QB Tony Romo completed a 4-yard TD pass to TE Jason Witten. The Bengals would end the half with kicker Shayne Graham getting a 41-yard and a 31-yard field goal."↵
Question: "How many second quarter field goals did the Bengals get?"↵
Answer: 2↵

Figure 1: An example in the Drop dataset.↵

Drop数据集

- Drop是一个由Allen Institute for Artificial Intelligence 团队所提出来的数据集。是Discrete Reasoning Over the content of Paragraphs的缩写，Drop数据集由众筹创建，包含了96k个question。要解决数据集中问题的答案，往往需要提取文本中不同的部分的信息，并执行简单的数学运算。



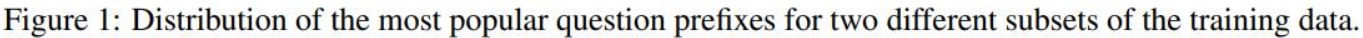
Drop数据集的构成

- 右图展示了Drop数据集中的问题的分类，问题主要由Number类问题构成，这类往往需要一定的数学运算能力。剩下的问题的答案大多是文本的一部分，我们将这类问题称为span类问题。

Answer Type	Percent	Example
NUMBER	66.1	12
PERSON	12.2	Jerry Porter
OTHER	9.4	males
OTHER ENTITIES	7.3	Seahawks
VERB PHRASE	3.5	Tom arrived at Acre
DATE	1.5	3 March 1992

Table 3: Distribution of answer types in training set, according to an automatic named entity recognition.

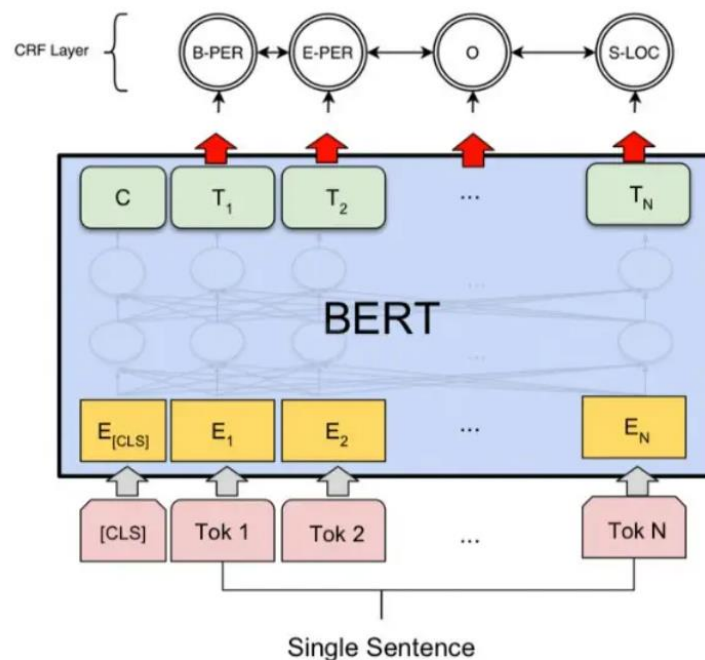
图片来源：《DROP: A Reading Comprehension Benchmark Requiring Discrete Reasoning Over Paragraphs》



图片来源: 《DROP: A Reading Comprehension Benchmark Requiring Discrete Reasoning Over Paragraphs》

Bert预训练模型

- BERT的全称为Bidirectional Encoder Representation from Transformers。
- 该模型通过双向的Transformers进行预训练，以生成深层的双向语言表征。
- 预训练后，只需要添加一个额外的输出层进行 fine-tune，就可以在各种各样的下游任务中取得 state-of-the-art 的表现。在这过程中并不需要对 BERT 进行任务特定的结构修改。



Bert预训练模型的基本结构

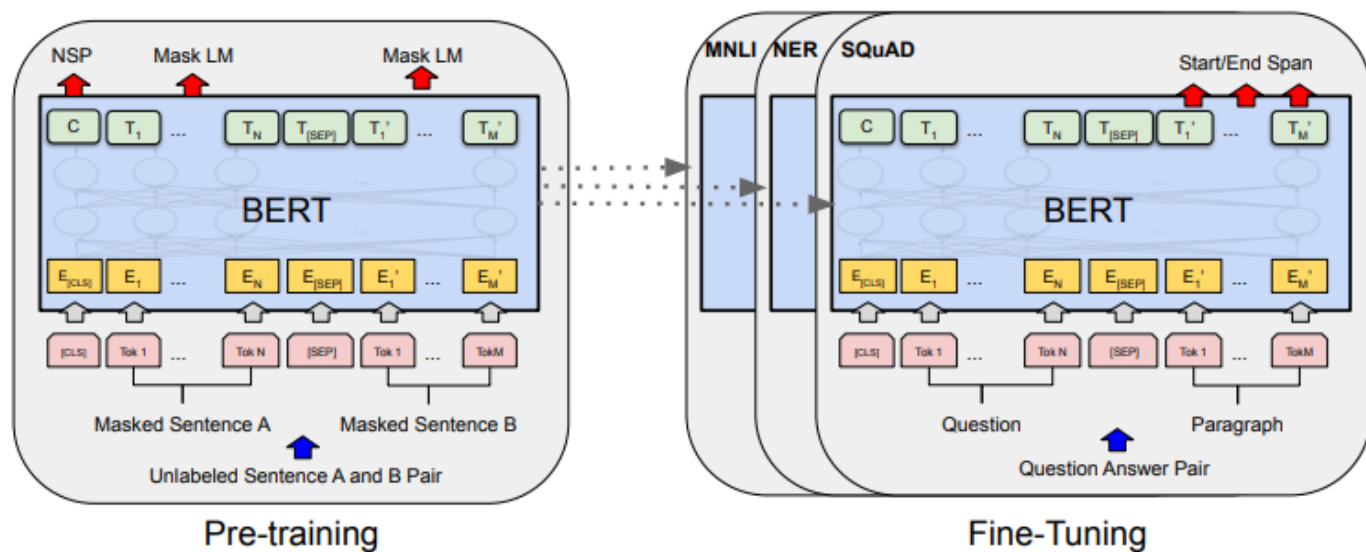


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).



模型框架



模型总体框架

- Stream net模型主体部分由一个 Bert 预训练模型和五个输出头组成。
- 输入的问题和文本经过tokenize后送入一个Bert的预训练模型，随后经过五个并联的输出层，根据head classifier模块的结果选择合适的输出层，最终解码出答案

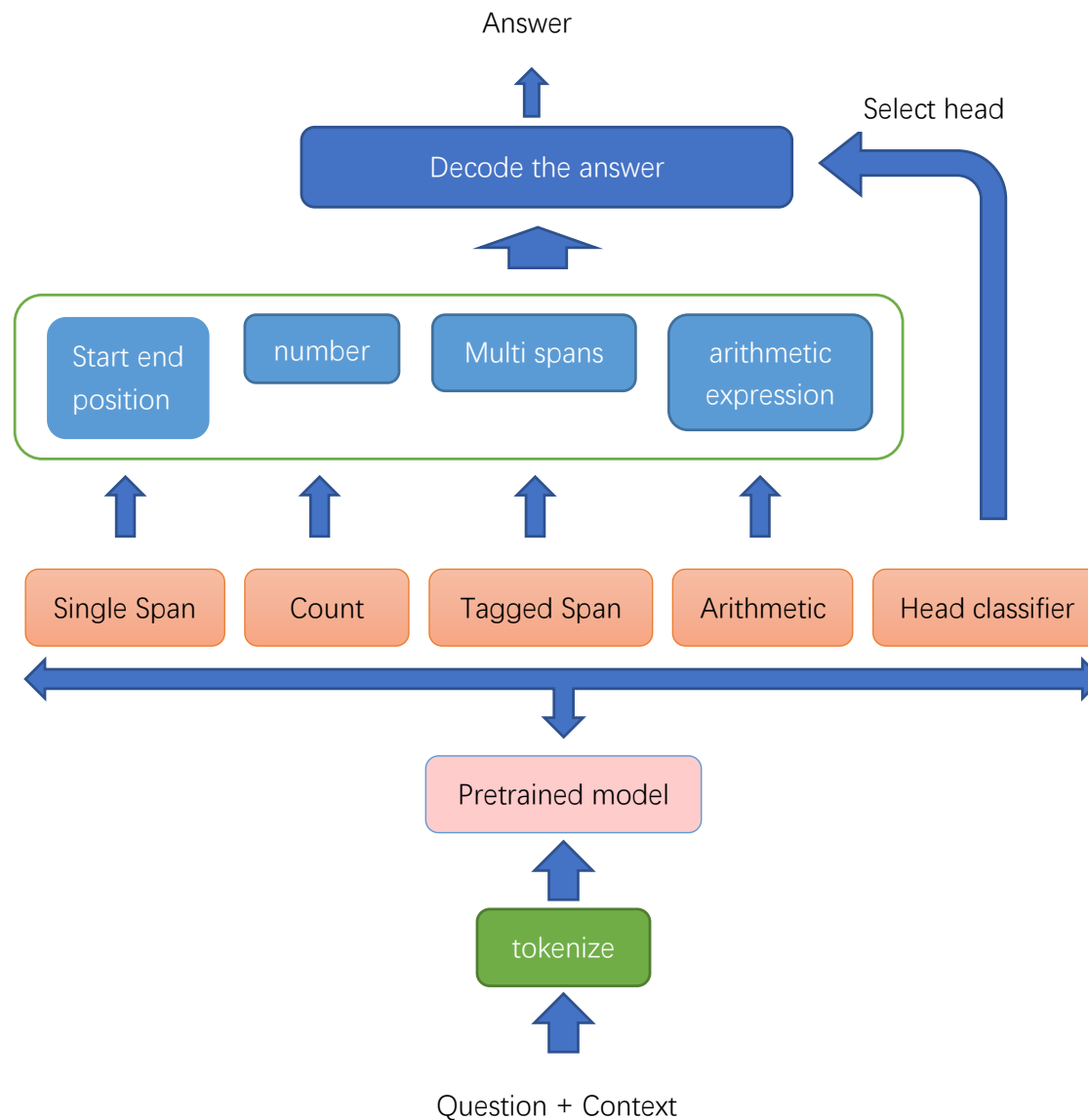


Figure 2: The framework of my model. It consists of a passage tokenizer, a Bert pretrained model and five output head.

Single Span Module

- Span类问题指的是答案能够从文本中提取出来的一类问题。
- 在Bert的工作中，给出了一种解决这类问题的方法：只需要在Bert的预训练模型后面加入两层全连接层，预测每个token是否为answer的开始或结束，取最有可能的start position和end position组合即可得到答案。

Passage:Hoping to rebound from their loss to the Patriots, the Raiders stayed at home for a Week 16 duel with the Houston Texans. Oakland would get the early lead in the first quarter as quarterback JaMarcus Russell completed a 20-yard touchdown pass to rookie wide receiver Chaz Schilens. The Texans would respond with fullback Vonta Leach getting a 1-yard touchdown run……“

question: "Who scored the first touchdown of the game?",

answer: "Chaz Schilens"

An example of span question

Tagged Spans Module

- 然而，在 Drop 数据集中，许多span类问题涉及多个分隔很远的spans，single span module 很难处理这种情况。
- 在这里，我使用了输出头的简单修改版本。我将预训练模型的输出输入到一个两层全连接网络中，然后使用 softmax函数对token进行二分类，计算token是答案的一部分的概率。

Arithmetic Module

- 算术问题占据了 Drop 数据集的大部分，也是该数据集中最困难的问题。现有的自然语言处理模型往往难以直接进行数学运算，这主要是因为现有的 tokenizer 难以有效地编码数字信息，许多数学运算的先验信息在 tokenize 阶段就已经消失了。

一个 Arithmetic 类问题的实例：
"question": "How many field goals did Kris Brown kick?",
"answer": "number": "3",

Arithmetic Module

- 首先假设一个算术问题的答案可以通过对文本中的数字进行加减来获得。我们提取文本中的所有数字，只需要预测每个数字的符号即可得到最终答案。
- 在我的实现中，符号 0 表示该数字不参与运算，1 表示应添加该数字，2 表示应减去该数字。
- 从预训练模型中获取隐藏表示后，我使用隐藏表示的池化输出作为段落摘要向量。我选择数字对应的标记表示，将summary vector和数字对应的token representation连接后输入一个两层全连接网络和一个 softmax 层，以获得这三种符号的可能性。

Arithmetic Module

- 在实践中，我发现 1 和 100 经常出现在算术表达式中。这是因为涉及百分比的计算经常发生，加一或减一也是常用的表达式。因此，我也将这两个数字作为特殊数字纳入计算，并预测它们的符号。
- 该模型仍然通过最小化预测符号和实际符号之间的交叉熵损失来训练。
- 预测时，提取文本中的数字，将文本输入模型并计算每个数字符号的概率，选择概率最高的符号进行加减运算。运算的结果就是预测的答案。

Count Module

- 但是，如果通过加减文本中的数字不能得到答案，我们需要另一种解决方案。这类不能从文中数字进行加减运算得到的数学题，就归结为count类问题。
- 计数问题的一个例子：一个篮子里有一个足球、两个乒乓球和三个排球。一共有多少种球？
- 我将预训练模型的pooler输出馈入两层前馈网络，然后使用softmax层计算count值为0、1、2、3的概率.....我只考虑count值小于10，大于 10 的部分被跳过。该模型仍然通过最小化预测数和实际数之间的交叉熵损失进行训练。

数据预处理

在预处理阶段，我提取了文本中所有数字的位置和值，并尝试按照上述方法得到答案。对于 spans 类型的问题，是寻找答案在文本中的位置，对于算术类型的问题，是遍历数字的运算组合。如果成功获得答案，则获得相应的训练数据和测试数据。如果无法生成相应输出所需的训练数据，则直接跳过该数据。

Head Type Classifier

在实际解决问题时，我们需要一种在各种输出层之间进行选择的方法。我把这个问题单独设置为分类问题，使用预训练的模型来判断问题的分类。如果答案属于上述类别之一，则可以使用相应的输出标头来预测答案。我设计了一个简单的分类头，由一个预处理模型和一个两层前馈神经网络组成。head 的输出是一个长度为 4 的向量，经过 sigmoid 函数后被裁剪在 0 和 1 之间。输出对应于每个输出头的使用概率。

Head Type Classifier

在实际的训练过程中，我使用上面的四个输出头来尝试生成一个答案。如果可以生成，则将对对应head的label设置为1。显然，同一个问题可能通过多个不同的output head来解决，比如单span head可以解决的问题，也可以通过tagged spans head解决，有些算术头能解决的问题，也可以用计数头来解决。输出标签的格式为 1 0 1 0。并且可能有一些问题不能用上面的输出头来解决。这部分数据直接跳过。

head类型分类器使用均方误差来计算预测值和实际标签之间的损失。

Evaluator

以上五个模型 (single span model、multispan model、arithmetic model、count model、head classifier model) 分别训练，预训练的模型不共享参数。

在模型的预测阶段，我们需要选择合适的输出头。一种可能的选择方法是选择得分最高的输出头。

但考虑以下两个因素：（1）数据集中的跨度问题大部分是单跨度问题，只有少数涉及多跨度问题。（2）训练集中的大部分数学问题都是用算术方法解决的，只有少数是用计数方法解决的。

模型的决策方式

因此，我设计了一个新的决策方法：

首先，根据两个跨度相关头的分数之和和两个数学相关头的分数之和，确定问题的基本分类。

如果是span question，首先尝试通过single span head成答案。

如果答案长度大于 7 或预测的结束位置 > 开始位置，请再次尝试使用tagged spans head输出预测答案。

如果这是一个math question，就用 arithmetic head来预测数学表达式，如果不能预测数学表达式，就用count head来预测。整个预测过程如图3所示：

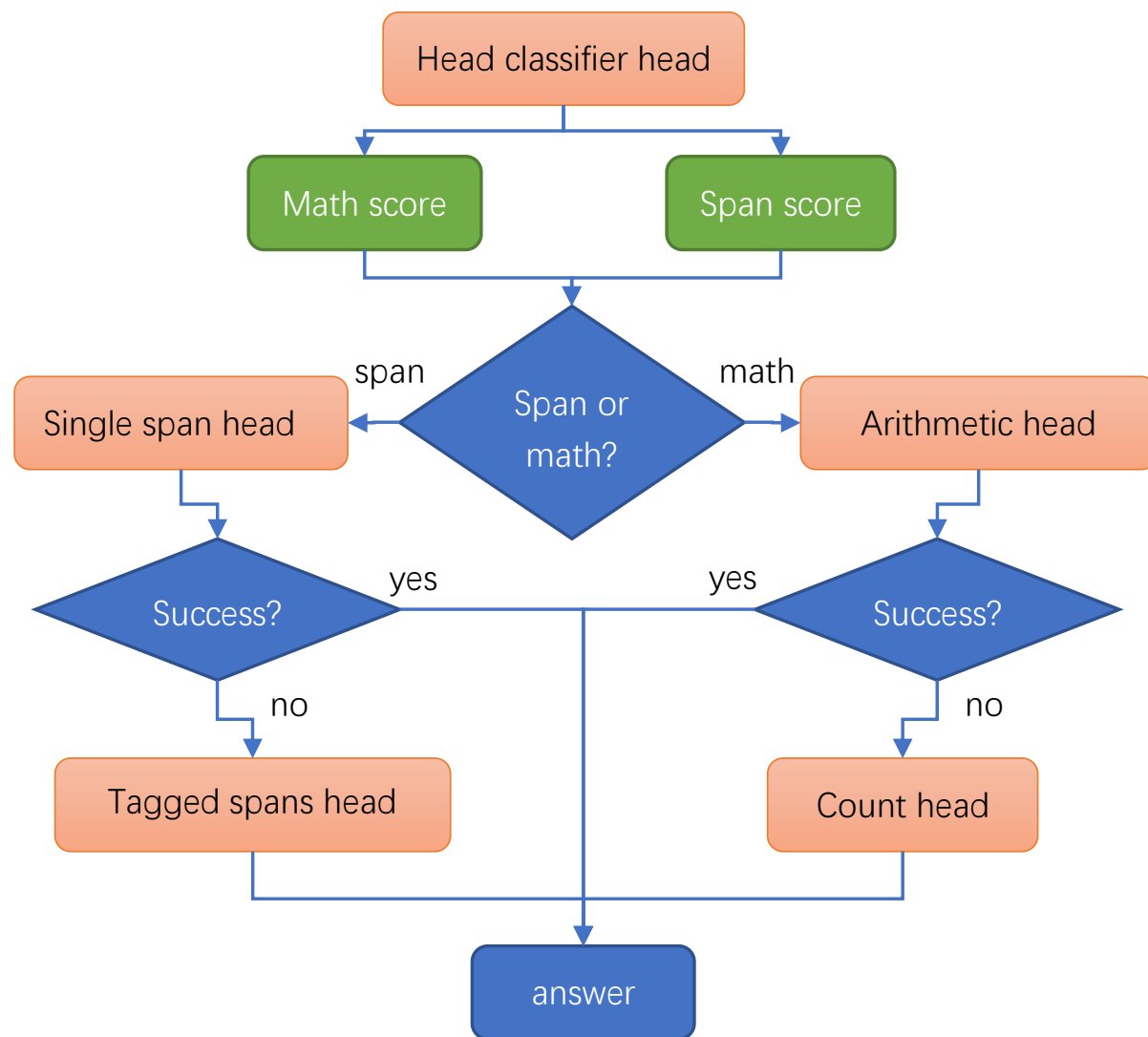


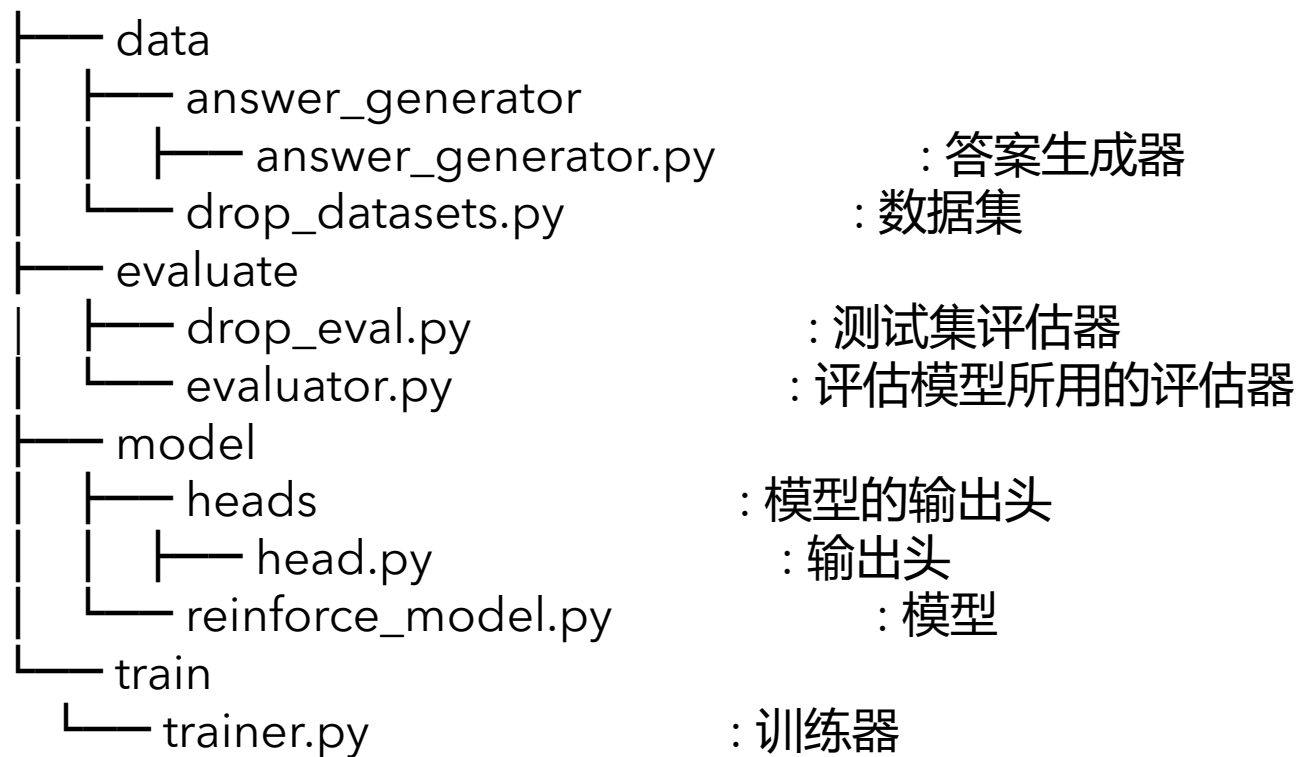
Figure3: the head choose strategy used for predicting the answer



代码实现



代码的整体框架



answer_generator

```
class AnswerGenerator:
    """
    base class for answer generator, which is used to preprocess answer for each question
    """
    def __init__(self):
        pass

    def generate_answer(self):
        raise NotImplementedError
```

在drop_datasets产生训练的实例的时候，会调用对应的AnswerGenerator，根据raw_inputs对每个instance产生对应的输出

head

```
class head(torch.nn.Module):
    # the loss_function is a function that takes in the output of the head and the target and return the loss
    def loss_fun(self, head_output: Dict[str, torch.Tensor], instance: Dict[str, torch.Tensor]) -> torch.Tensor:
        raise NotImplementedError
    # the predict function is used to generate the answer for the question
    def predict(self, instance: Dict[str, any]) -> Dict[str, any]:
        raise NotImplementedError

    def forward(self, head_input: Dict[str, torch.Tensor]) -> torch.Tensor:
        raise NotImplementedError
```

每一种head都是一个answer_generator所对应，完成前向传播、预测输出，计算损失的工作

Trainer

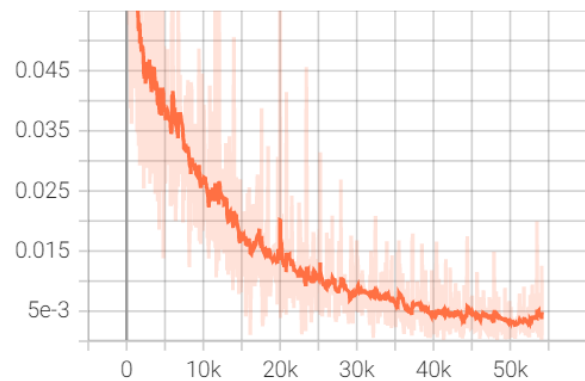
我使用 Bert-base-uncased 模型进行预训练，Bert-base-uncased 的隐藏大小为 768。学习率为 $1e-5$ ，我使用 Adam 优化器训练模型。相对较小的学习率需要更多的 epoch 学习，我为每个模型选择 30 个 epoch，batch size 为 16。模型在每个 epoch 结束时在验证集上进行测试。我在一张 rtx3090 显卡上对模型进行了总共 40 个小时的训练。

由于测试集小，测试阶段分支较多，所以我直接在CPU上进行测试阶段。

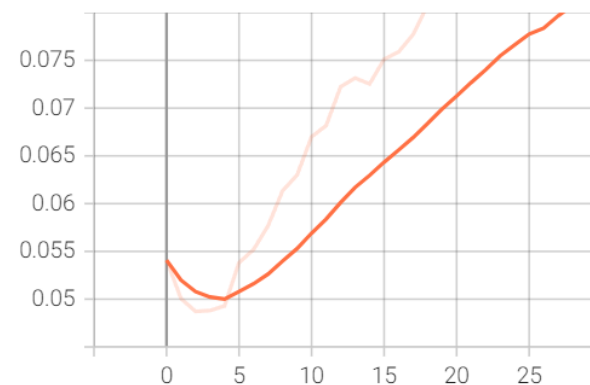
我使用 HuggingFace 的 Transformers 库 [4] 来实现 Bert 预训练模型。文本输入被截断为 512 个 token。

训练曲线

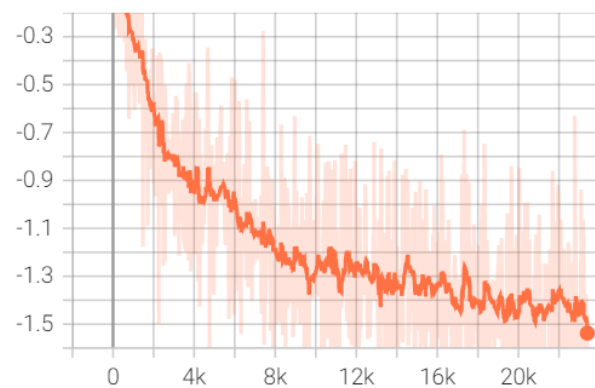
tagged_spans_train_loss



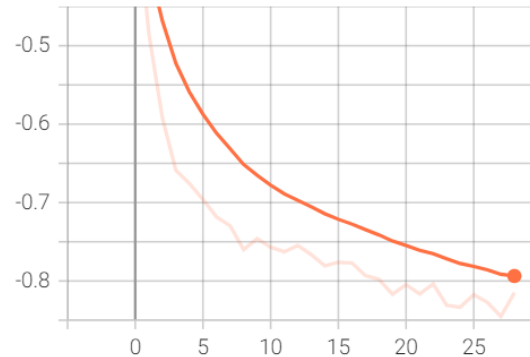
tagged_spans_average_dev_Loss:



single_span_train_loss

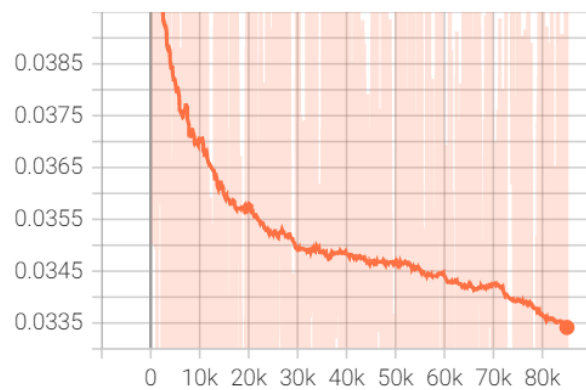


single_span_average_dev_Loss:

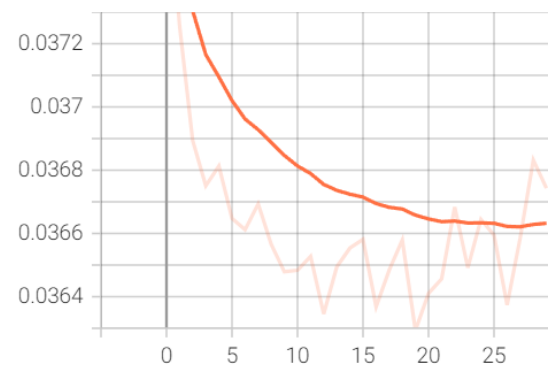


训练曲线

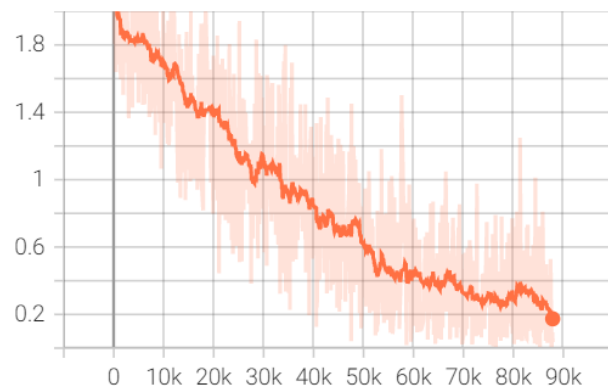
arithmetic_train_loss



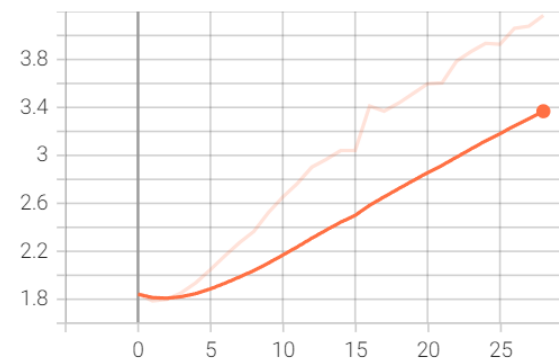
arithmetic_average_dev_Loss:



count_train_loss

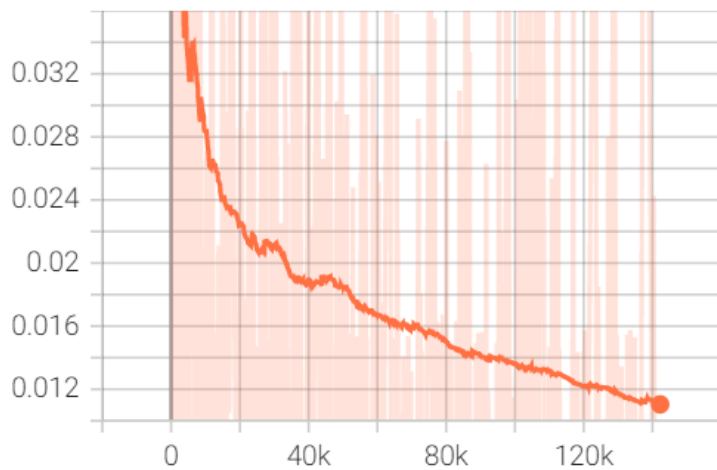


count_average_dev_Loss:

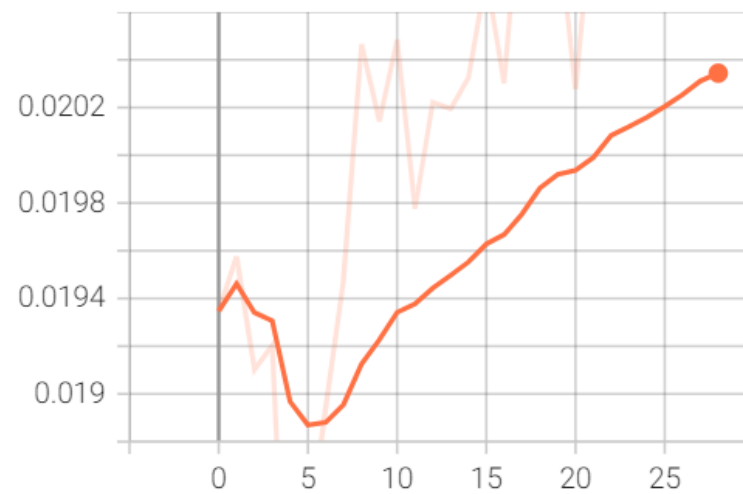


训练曲线

head_type_train_loss



head_type_average_dev_Loss:

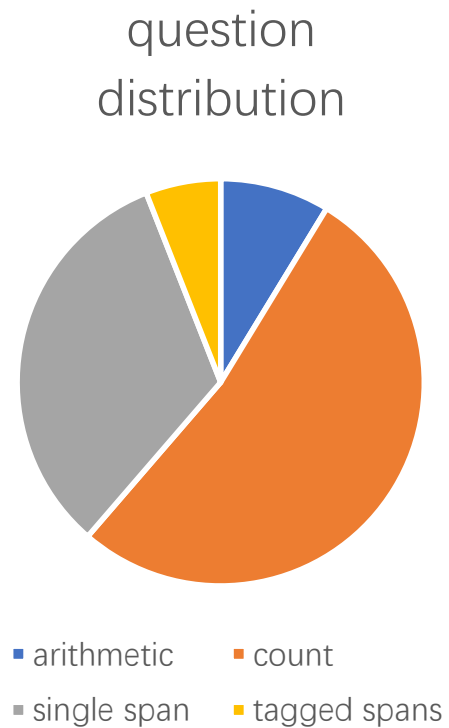
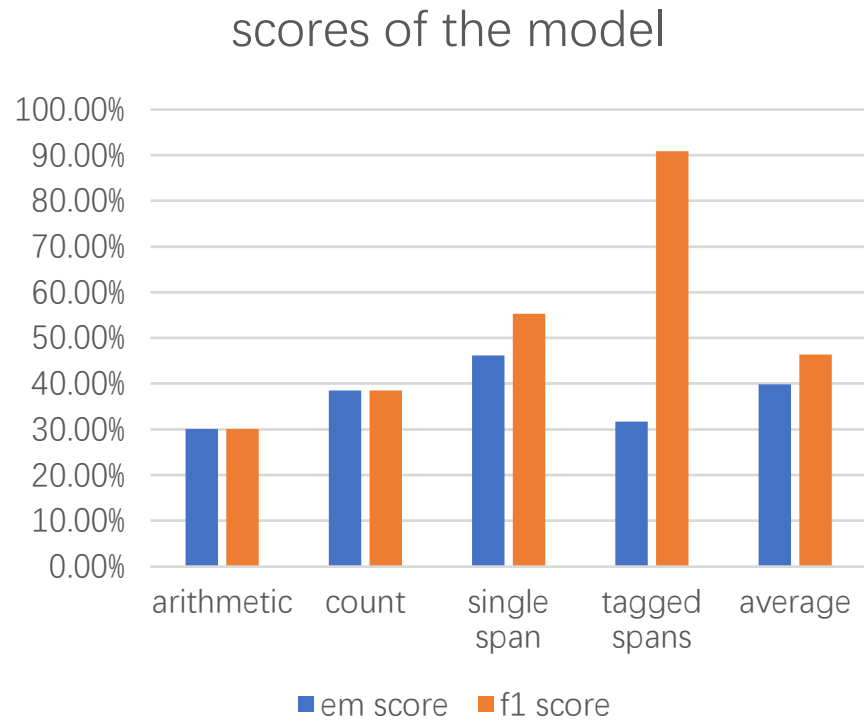




实验结果



模型在训练集上的测试结果

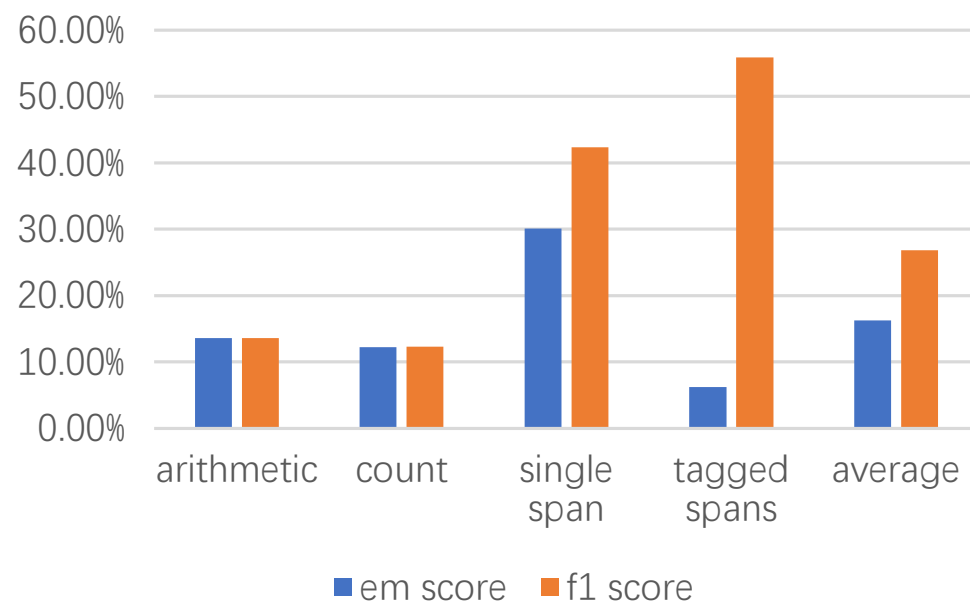


模型在训练集上的测试结果

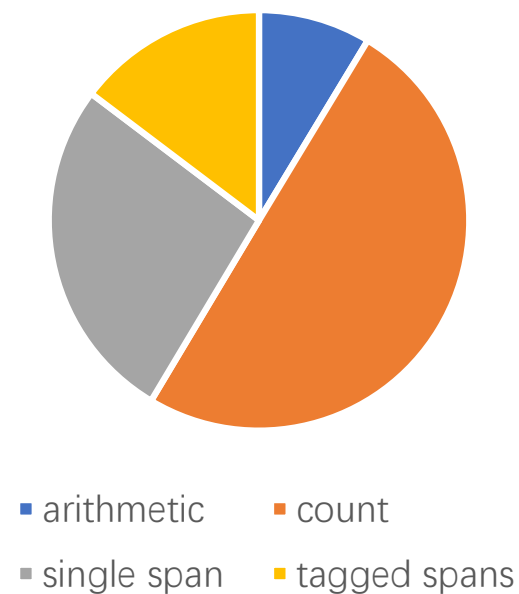
	arithmetic	count	single span	tagged spans	average
em score	30.10%	38.50%	46.20%	31.67%	39.88%
f1 score	30.10%	38.50%	55.31%	90.90%	46.39%
num	402	2426	1508	275	4611

模型在测试集上的测试结果

scores of the model



question distribution



模型在测试集上的测试结果

	arithmetic	count	single span	tagged spans	average
em score	13.58%	12.20%	30.10%	6.18%	16.22%
f1 score	13.58%	12.27%	42.35%	55.87%	26.83%
num	324	1868	1000	550	3742

错误分析

我们可以看到，将这个问题视为一个span类问题而不是一个arithmetic类问题更合理。但是在训练的时候，我的框架对于所有number类型的answer都是采用arithmetic类的方法来处理的，导致模型更倾向于使用arithmetic header，给出了错误的答案。

Context:Detroit's Calvin Johnson caught a 1-yard pass in the third quarter.

Question: "How many yards was the shortest touchdown scoring play?"

Answer: "number": "1"

The head type model choose: arithmetic

Predict result: 31

An example of a model misjudgment

错误分析

这里模型使用 tagged spans 的方式，但输出了两个相同的正确答案，因为文本中也有两个相同的“正确”答案。我发现 tagged spans 类输出 head 倾向于输出更多可能的答案部分，导致其 f1 分数相对较高但 em 分数较低。

Context: "In South America , the Portuguese conquered from Spain most of the Rio Negro valley, and repelled a Spanish attack on Mato Grosso . Between September 1762 and April 1763, Spanish forces led by don Pedro Antonio de Cevallos,Hispano-Portuguese war of 1763-1777.

Question: "What was the latter 1/10 of a decade that Spanish forces led a campaign against the Portuguese in Uruguay and South Brazil?",

Answer: "spans": "1763"

The head type model choose: tagged_spans

Predict result: "1763 1763"

An example of a model misjudgment

错误分析

这里应该使用arithmetic head, 但是模型没有使用arithmetic head生成有效的表达式, 只能使用count head, 而count head不能输出大于10的值, 导致错误。

Context: ".....It was 17-7. Then the Texans made a 17 play drive resulting in a 2-yard Ben Tate touchdown to make it 24-7. The Texans would get a field goal to make it 27-7, and that would end the game.",

Question: "How many points were scored in total?"

Answer: "number": "31"


The head type model choose: count

Predict: 3

An example of a model misjudgment



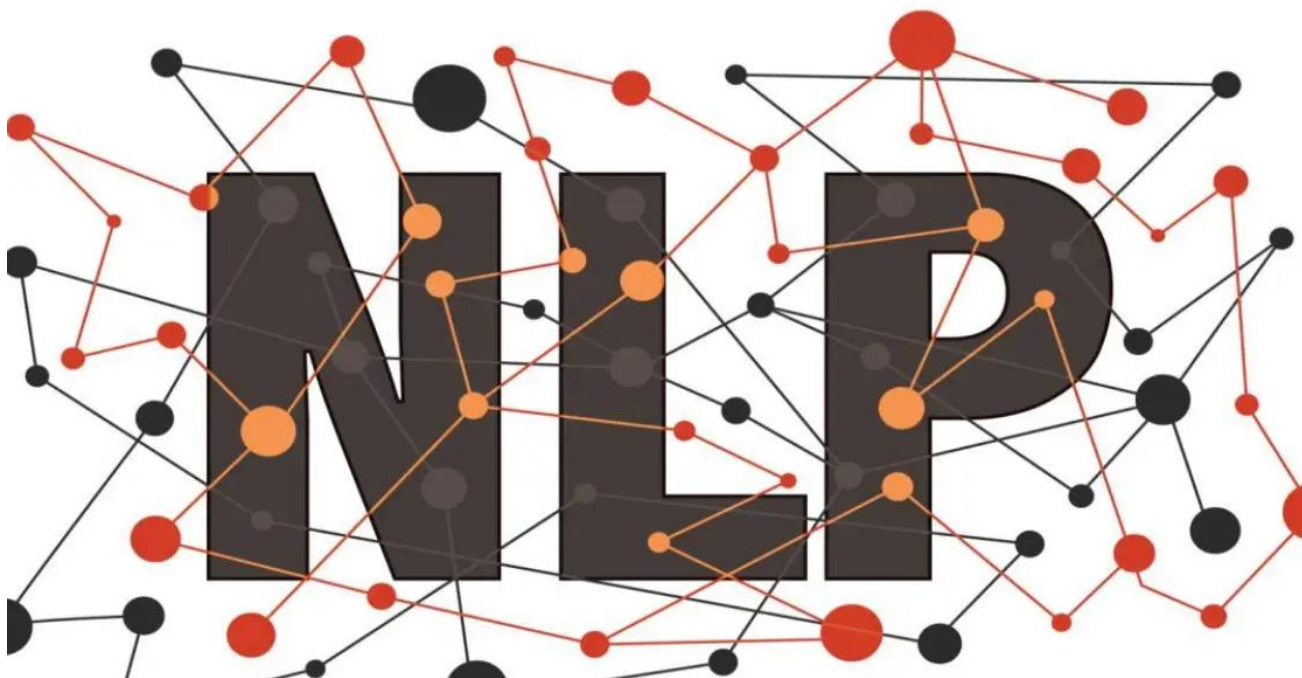
总结展望



内容

- 在这个项目中，我通过对问题进行分类，将复杂多样的自然语言理解问题简化为子问题，构建了一套解决自然语言处理问题的框架。
- 我的方法是使用混合模型来处理复杂多样的自然语言处理问题，后续的实验结果也证明了这种方法的有效性。
- 在未来的工作中，整个模型可以进一步细分和分布式训练，将多个分支模型融合到一个统一的框架中。输出头是提升模型阅读理解能力的关键。未来，我将探索更复杂的输出模型以进行后续研究，尤其是在 Drop 数据集上表现不佳的算术和计数输出标头。

NATURAL LANGUAGE PROCESSING



谢谢