

**1. 问题背****景**

**问题-1：**⼤模型通常包含数亿甚⾄数百亿个参数，对其进⾏微调需要⼤量的计算资源和存储空间。 **问题-2：**在微调过程中，直接修改预训练模型的所有参数可能会破坏模型的原始性能。

**问题-3：**存储和部署微调后的⼤模型需要⼤量存储空间，尤其是当需要在多个应⽤场景中部署不同微调版本时。

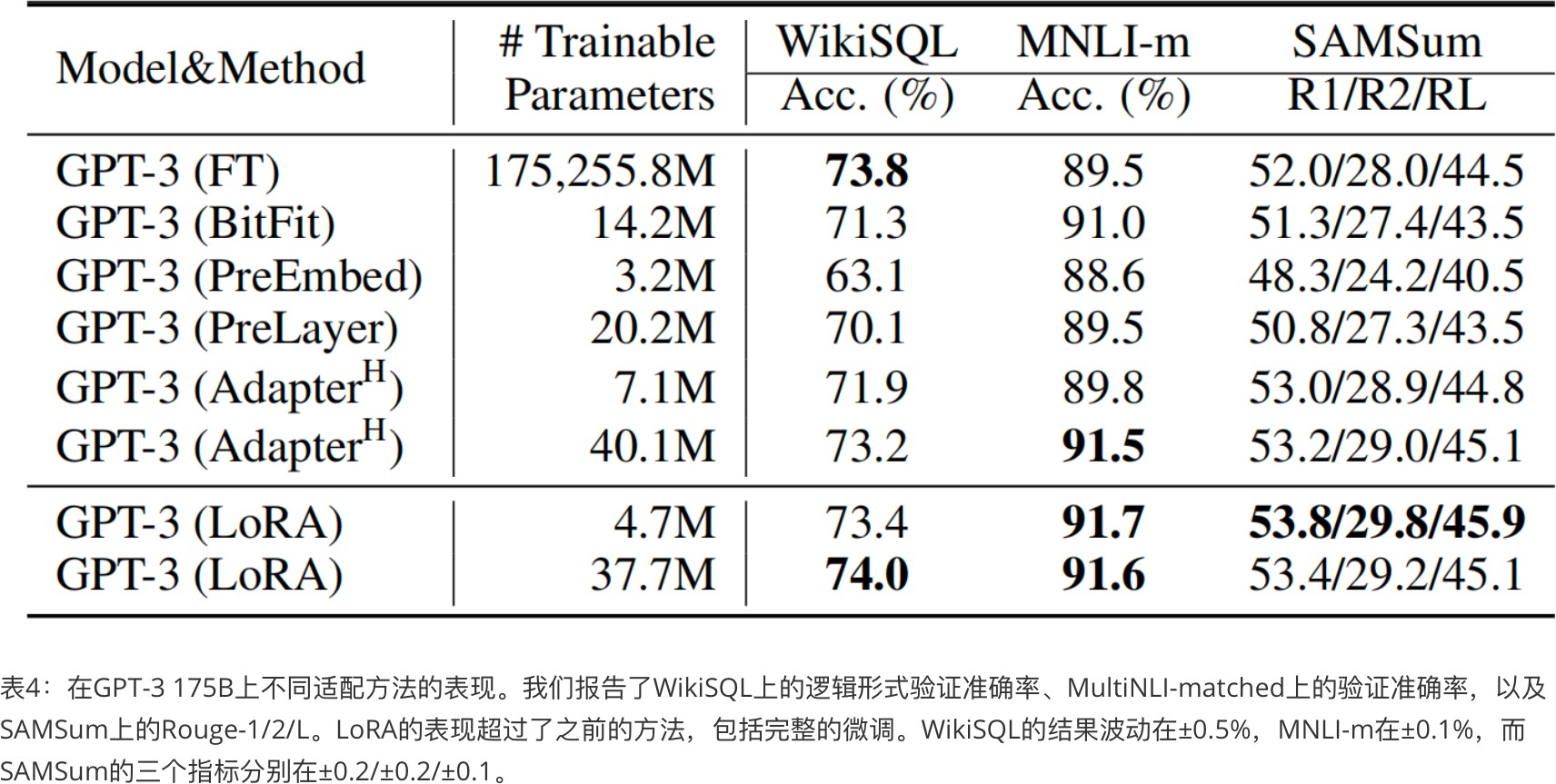
**问题-4：**许多微调⽅法会增加推理阶段的计算延迟，影响模型的实时性应⽤。

**2. 解决措****施**

LoRA（Low-Rank Adaptation) 通过引⼊低秩矩阵分解，在减少计算资源和存储需求的同时，保持了预训练模型的初始性能，稳定了微调过程，并降低了存储和部署成本。它特别适⽤于⼤规模模型的微调，在资源有限的环境中具有显 著的优势。

**3. LoRA 优****势**

**存储与计算效率**：通过低秩适应（LoRA），可以显著减少所需存储的参数数量，并减少计算需求。



**适应性与灵活性**：LoRA⽅法允许模型通过只替换少量特定的矩阵A和B来快速适应新任务，显著提⾼任务切换的效率。

**训练与部署效率**：LoRA的简单线性设计允许在不引⼊推理延迟的情况下，与冻结的权重结合使⽤，从⽽提⾼部署 时的操作效率。

**GPT-3 175B**

**4. 深⼊理解 LoR****A**

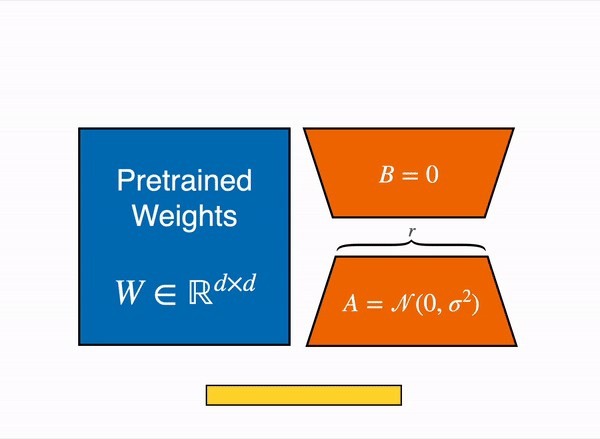
**4.1 为什么需要低秩分解？**

现代预训练模型虽然是过参数化的，但在微调时参数更新主要集中在⼀个低维⼦空间中。

2. 参数更新

可以在低维度中进⾏优化，⾼维参数空间中的⼤部分参数在微调前后⼏乎没有变化。

3. 低秩分解使参数优化更⾼效，但如果参数更新实际上在⾼维⼦空间中发⽣，可能会导致重要信息遗漏和LoRA⽅法失效。



**4.2 LoRA算法原理**

LoRA 通过冻结预训练的权重矩阵

，仅学习⼀个较⼩的偏置矩阵

，公式如下：

是⼀个随机初始化的矩阵，且服从正态分布

初始化为零矩阵，即

。

**作⽤**：这种初始化⽅法使得在训练初期，新增的部分坏预训练模型的初始性能。

对原始权重

的影响为零，从⽽不会破

参数需要训练，减少了计算梯度所需的内存和浮点运算量（FLOPS）。

秩

，只有

的 LoRA 到⼀个

的 7B 权重矩阵，仅训练不到1%的原始参数量。

例如，应⽤

假设我们有⼀个预训练的⼤模型，其中某个权重矩阵尺⼨为 4096 x 4096。

原始权重矩阵 的参数数量为：

的维度为

。假设

，即这个权重矩阵的

**使⽤ LoRA ⽅法**

选择⼀个较⼩的秩 ，例如

。在 LoRA 中，我们将权重矩阵分解为两个低秩矩阵

和

，其中：



的维度为

的维度为

使⽤ LoRA ⽅法后，需要训练的参数数量为：

与原始模型相⽐，使⽤ LoRA 后的参数数量显著减少：

也就是说，只需要训练原始参数数量的约 1.56%。

**⽆需完全微调**：LoRA⽅法允许在不累积对所有权重矩阵的全秩梯度更新的情况下，通过调整LoRA的秩 来逼近原始模型的表达能⼒。

**⽆额外推理延迟**：在⽣产中部署时，可以显式计算并存储

理。这保证了与细调的模型相⽐，不会引⼊任何额外的延迟。

，并像往常⼀样执⾏推

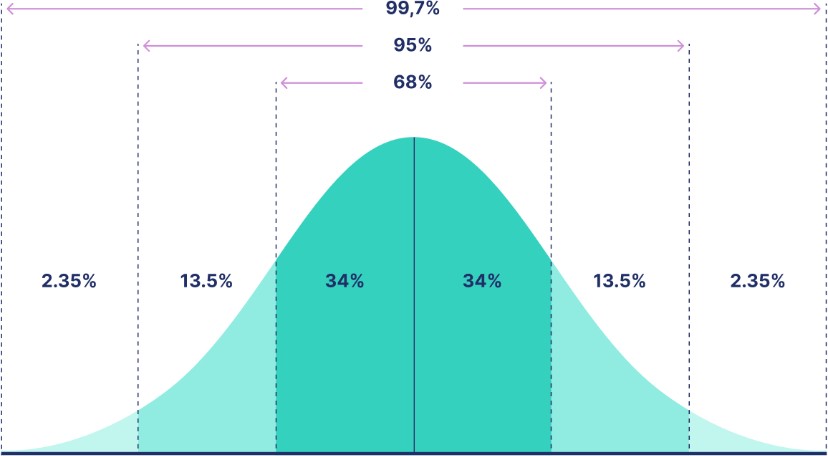
**知识补充**：

问-1：为什么初始化参数使⽤正态分布？ 答-1：这样做的原因包括：

**① 确保初始梯度的有效传播**：正态分布初始化有助于在训练初期确保梯度有效传播，避免梯度消失或爆炸的问题。

**② 提供⾜够的随机性**：正态分布的随机初始化为模型提供了⾜够的随机性，从⽽能够探索更⼴泛的参数空间，增加了模型找到最优解的可能性。

**③ 平衡训练初期的影响**：正态分布初始化的值⼀般较⼩，结合 B 初始化为零矩阵，可以在训练初期确保新增的偏置矩阵对原始预训练权重的影响为零，从⽽避免破坏预训练模型的初始性能。





问-2：为什么 A初始化服从正态分布？⽽B初始化为零矩阵？

答-2：（1）如果B和A全部初始化为零矩阵，缺点是很容易导致梯度消失；（2）如果B和A全部正态分布初始

化，那么在模型训练开始时，就会容易得到⼀个过⼤的偏移值

，从⽽引起太多噪声，导致难以收敛。

**>>> 举例讲解 <<<**

假设我们有⼀个预训练的权重矩阵

，我们想应⽤LoRA进⾏微调。

，其维度为

1. **初始化**：

我们选择⼀个较⼩的秩 ，例如

。

，

，例如：

初始化矩阵

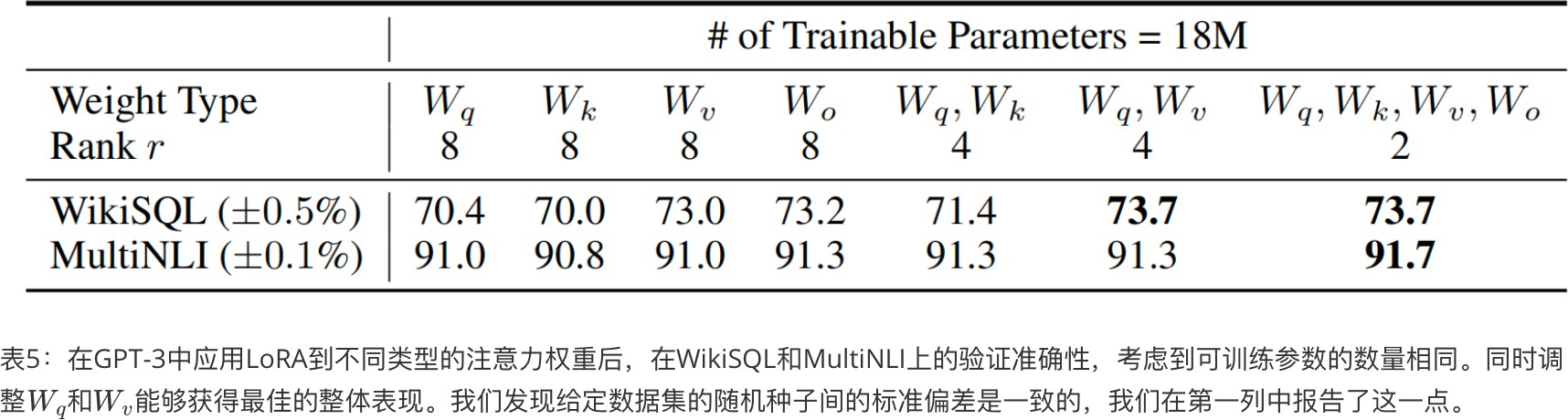
和

，其中

。

的元素服从正态分布

初始化为零矩阵：





2. **训练**：

在训练过程中，仅更新

和

的值。

随着训练的进⾏，

3. **微调权重计算**：

的值逐渐变得⾮零。

训练结束后，我们得到

和

的最终值，计算

。

将

加到

上，得到微调后的权重矩阵

。

**4.3 选择哪些权重矩阵进⾏适配？**

在有限参数预算下，应选择哪些权重矩阵进⾏适配以最⼤化下游任务性能？

**主要发现：**

同时适配

、

、

、

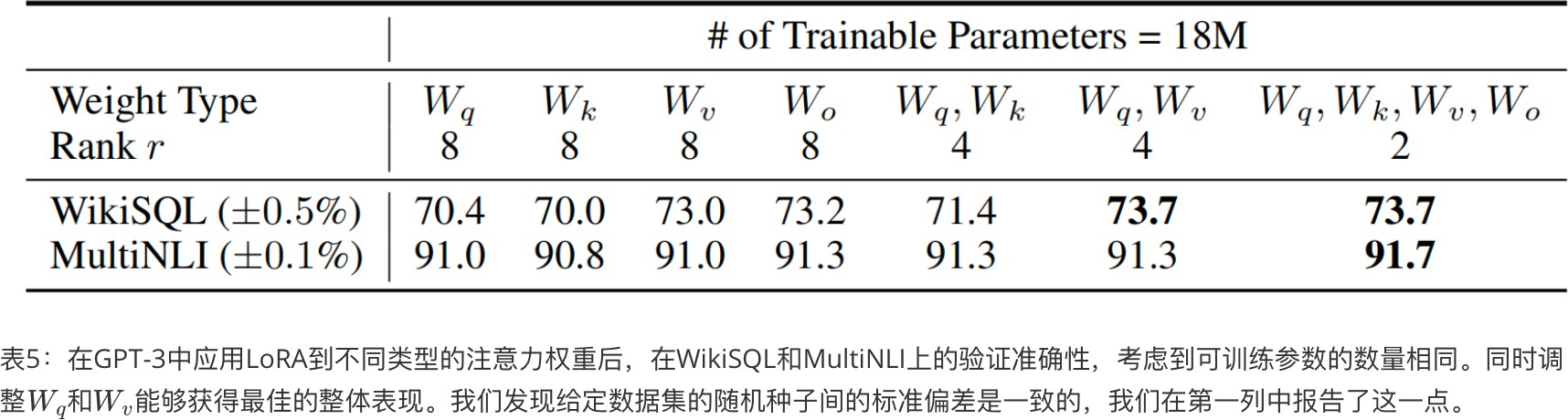
提供了最佳性能。

中捕获⾜够信息，表现优于单⼀类型权重适配但具有更⾼秩的策略。

低秩（例如rank为4）⾜以在

**4.4 为什么 LoRA 在 Q, K, V, O 上有效？**

LoRA（Low-Rank Adaptation）在 Transformer 模型的 Q（Query）、K（Key）、V（Value）和 O（Output）矩阵上有效果的原因可以归结为这些矩阵在注意⼒机制中的核⼼作⽤以及 LoRA ⽅法在降低参数数量的同时保持或提升模型性能的能⼒。具体原因如下：



**4.4.1 Self-Attention**

**Q（Query）矩阵**：⽤于⽣成查询向量，决定模型在注意⼒机制中对输⼊的关注程度。**K（Key）矩阵**：⽤于⽣成键向量，与查询向量计算相似度，帮助确定注意⼒分布。**V（Value）矩阵**：⽤于⽣成数值向量，实际传递注意⼒机制计算的输出。**O（Output）矩阵**：⽤于将多头注意⼒的输出合并并映射回原始维度。

**4.4.2 信息传播的关键路径**

、 、

和

矩阵在信息传播和特征表示中起着关键作⽤：

**查询与键的交互**：

**数值的加权求和**： **多头输出的整合**：

和

的交互决定了注意⼒分布，影响模型对输⼊序列的不同部分的关注度。

矩阵通过加权求和操作，将注意⼒分布转化为具体的输出。

矩阵整合多头注意⼒的输出，提供最终的特征表示。

**4.4.3 LoRA 的低秩近似**

LoRA 通过将权重矩阵分解为两个低秩矩阵（例如持模型性能：

），减少了参数数量，降低了计算和存储成本，同时保

和 矩阵通常包含⼤量参数，LoRA 的低秩分解显著减少了需要优化的参数数量。

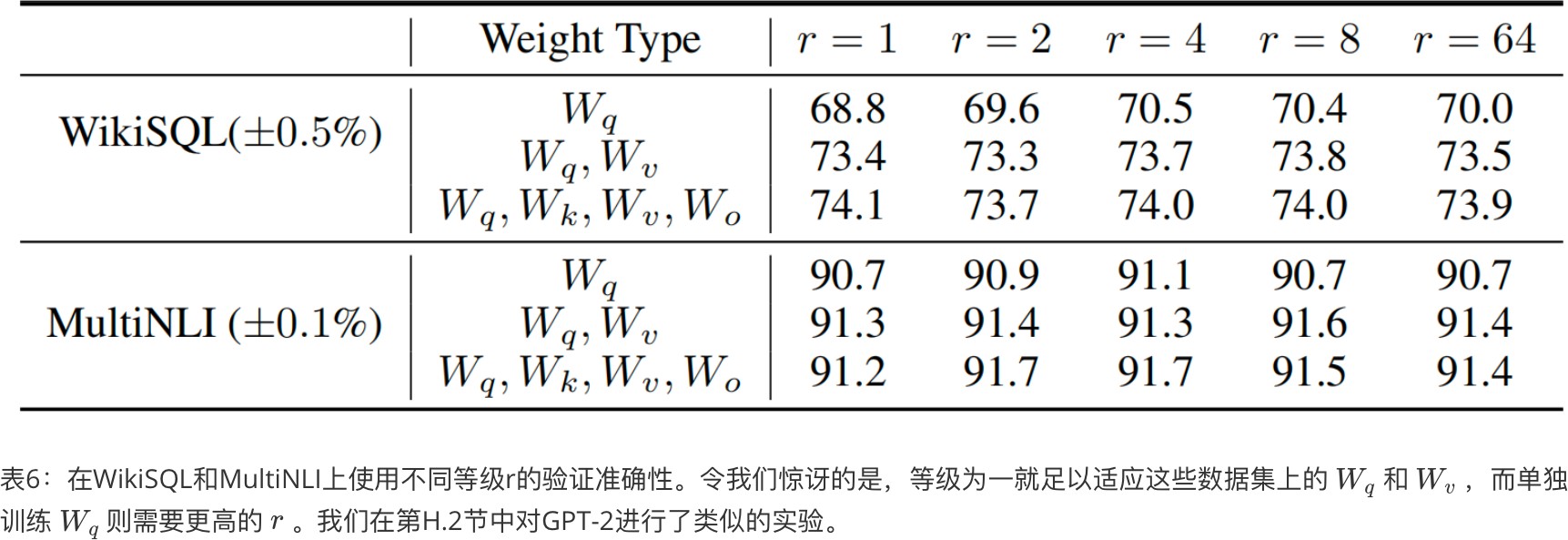
**参数压缩**： 、 、

**性能保持**：低秩矩阵能够捕捉到原始矩阵的主要信息，确保模型性能不受显著影响。

**4.5 LoRA中的最优秩**

**的选择**

作者探讨了不同秩 对模型性能的影响，并确定最⼩的有效秩，即“内在秩”。



**主要发现：**

LoRA 即使在⾮常⼩的秩 下也展现出了竞争⼒的性能。

在同时适配

和

时⽐仅适配

表现更好。

增加 并没有覆盖更多有意义的⼦空间，说明低秩适配矩阵已⾜够。

**4.6 LoRA可以应⽤到模型中的哪些层？**

LoRA（Low-Rank Adaptation）可以插⼊到模型的多个地⽅，具体取决于需要微调的模型部分和任务要求。以下是⼀些常⻅的插⼊位置及其原因：

**4.6.1 线性层（全连接层）**

**位置：**

在深度神经⽹络的全连接层（Linear Layer）中，通常会使⽤线性变换

**原因：**

**主要参数集中**：全连接层通常包含⼤量参数，通过在这些层中应⽤LoRA，可以显著减少需要微调的参数数量。 **计算密集型**：全连接层的计算量较⼤，通过低秩近似可以有效降低计算复杂度。

。

**4.6.2 注意⼒层**

**位置：**

在Transformer模型的多头⾃注意⼒（Multi-head Self-Attention）机制中，包括查询（Query）、键（Key）和值

（Value）矩阵的线性投影部分。

**原因：**

**关键功能组件**：⾃注意⼒机制是Transformer模型的核⼼组件，对模型性能影响重⼤。对这些矩阵进⾏低秩近似可以显著影响模型的表达能⼒。

**参数量⼤**：这些投影矩阵包含⼤量参数，使⽤LoRA可以减少参数数量，降低计算和存储需求。

**4.6.3 嵌⼊层**

**位置：**

在NLP任务中，嵌⼊层⽤于将离散的词汇表映射到连续的向量空间。

**原因：**

**⾼维稀疏表示**：嵌⼊层通常包含⼤量⾼维向量，通过LoRA可以有效降低维度，减少计算量和内存占⽤。

**提升训练效率**：低秩分解可以使嵌⼊层的训练更加⾼效。

**5. 案例实****战**

# 本机实验环境

1.

2.

3.

4.

ubuntu20.04 Python 3.10.14

pytorch 1.13.0

CUDA

Cuda compilation tools, release 11.8, V11.8.89

Build cuda\_11.8.r11.8/compiler.31833905\_0

**01\_LoRA\_案例实战.ipynb**

**02\_LoRA\_微软开源项⽬**

**第1步：环境配置**

conda create -n lora python=3.10

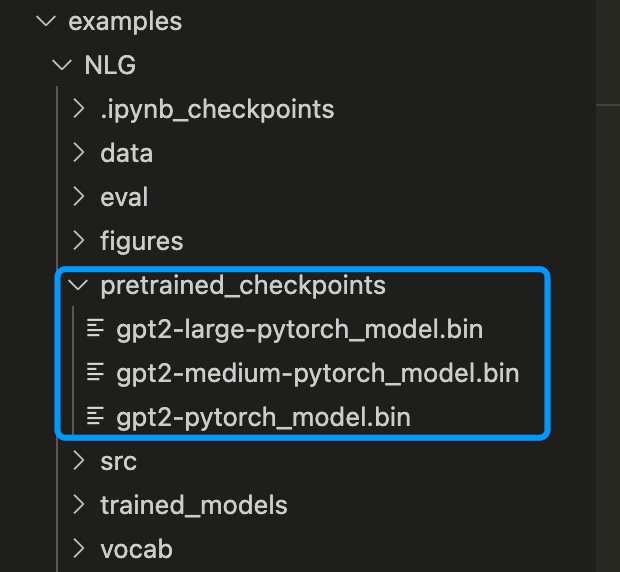
# conda init bash && source /root/.bashrc conda activate lora

# conda install ipykernel

# ipython kernel install --user --name=lora

名称

# 设置kernel，--user表示当前⽤户，lora为虚拟环境



pip pip

pip

install install

install

torch==1.13.0 progress

transformers==4.39.3 deepspeed

accelerate

datasets==2.18.0

peft

bitsandbytes

# 安装loralib # ⽅式-1

pip install loralib

## ⽅式-2（开发者模式）

pip install -e .

**第2步：下载预训练模型**

# 在路径：/LoRA/examples/NLG/ 下运⾏命令

bash

download\_pretrained\_checkpoints.sh

**第3步：下载训练数据集**

# 在路径：/LoRA/examples/NLG/ 下运⾏命令

bash create\_datasets.sh

**第4步：在数据集E2E上训练**

python -m torch.distributed.launch --nproc\_per\_node=1 src/gpt2\_ft.py \

--train\_data ./data/e2e/train.jsonl \

--valid\_data ./data/e2e/valid.jsonl \

--train\_batch\_size 8

--grad\_acc 1 \

--valid\_batch\_size 4

--seq\_len 512 \

--model\_card gpt2.md

\

\

\

--init\_checkpoint ./pretrained\_checkpoints/gpt2-medium-pytorch\_model.bin

--platform local \

--clip 0.0 \

--lr 0.0002 \

--weight\_decay 0.01 \

--correct\_bias \

--adam\_beta2 0.999 \

--scheduler linear \

--warmup\_step 500 \

--max\_epoch 5 \

--save\_interval 1000 \

--lora\_dim 4 \

--lora\_alpha 32 \

--lora\_dropout 0.1 \

--label\_smooth 0.1 \

--work\_dir ./trained\_models/GPT2\_M/e2e \

--random\_seed 110

\

# 参数解析：

python -m torch.distributed.launch: 使⽤PyTorch的分布式启动模块来启动脚本。

--nproc\_per\_node=1: 每个节点上的进程数为1，即不进⾏实际的分布式训练，只在单个进程上运⾏。

--grad\_acc: 梯度累积步数

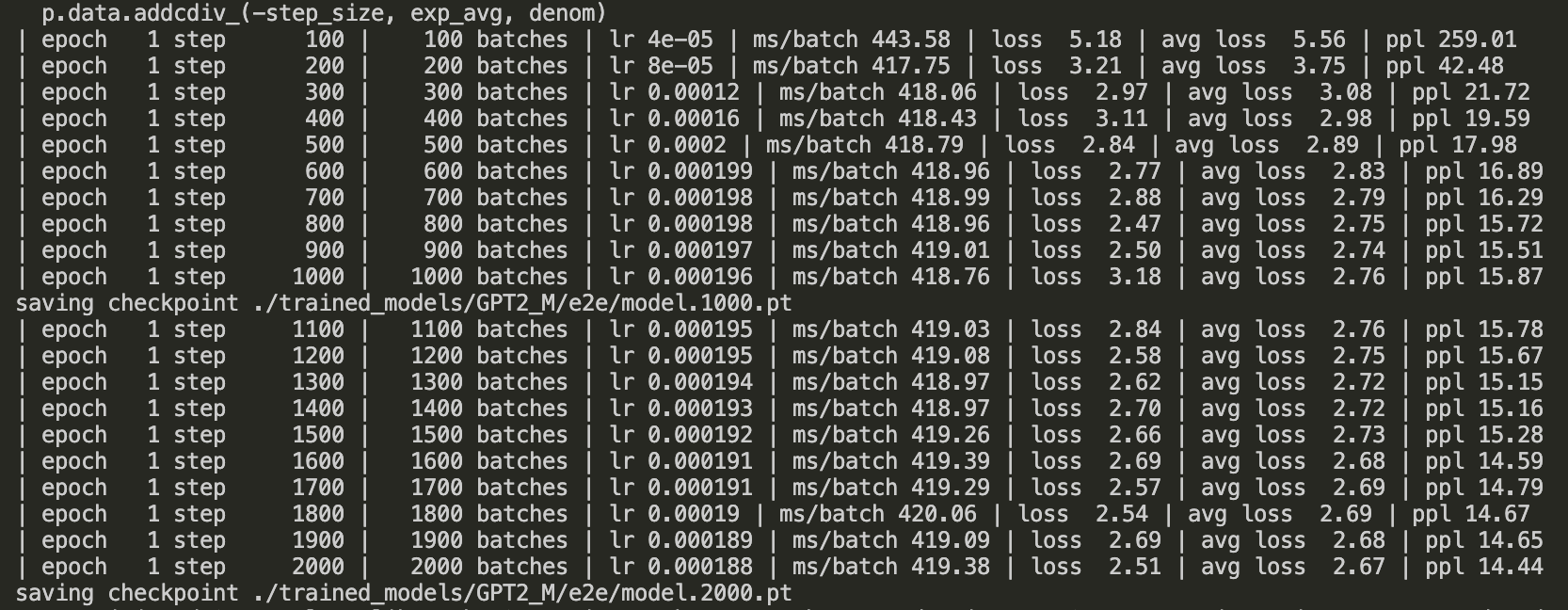
--platform: 指定平台

--clip: 梯度裁剪阈值

--correct\_bias: 是否修正偏差

--adam\_beta2: Adam优化器的beta2参数

--label\_smooth: 标签平滑系数



**跑完第1个epoch后的输出模型**

**第5步：基于上⼀步训练之后的输出模型进⾏推理**

python -m torch.distributed.launch --nproc\_per\_node=1

--data ./data/e2e/test\_200.jsonl \

--batch\_size 8 \

--seq\_len 512 \

--eval\_len 64 \

--model\_card gpt2.md \

src/gpt2\_beam.py \

--init\_checkpoint ./trained\_models/GPT2\_M/e2e/model.5258.pt \

--platform local \

--lora\_dim 4 \

--lora\_alpha 32 \



--beam 10 \

--length\_penalty 0.8 \

--no\_repeat\_ngram\_size 4 \

--repetition\_penalty 1.0 \

--eos\_token\_id 628 \

--work\_dir ./trained\_models/GPT2\_M/e2e \

--output\_file predict.5258.s200.jsonl

# 参数解析：

--beam: 束搜索的束宽，指在搜索过程中保留的候选序列数量。更⼤的束宽通常会导致更好的搜索结果，但也会增加计算成本。

--length\_penalty: ⻓度惩罚系数，避免⽣成过短或过⻓的序列。较低的值会惩罚较⻓的序列，较⾼的值会⿎励⽣成更

⻓的序列。

# 示例：

--length\_penalty 0.8：适度惩罚较⻓的序列，⽣成结果不会过⻓。

--length\_penalty 1.0：不进⾏⻓度惩罚，⽣成结果的⻓度将仅由概率决定。

--length\_penalty 1.2：⿎励⽣成更⻓的序列，减少⽣成短序列的可能性。

--no\_repeat\_ngram\_size: 禁⽌重复的n-gram⼤⼩，⽤于避免⽣成结果中出现重复的n-gram。n-gram是指连续的n 个词，设置这个参数可以提⾼⽣成⽂本的多样性。

# 示例：

--no\_repeat\_ngram\_size 2：禁⽌⽣成结果中出现重复的⼆元组（bigram），例如“the cat the cat”。

--no\_repeat\_ngram\_size 3：禁⽌⽣成结果中出现重复的三元组（trigram），例如“the cat is the cat is”。

--repetition\_penalty: 重复惩罚系数，⽤于降低⽣成过程中重复词或短语的概率。较⾼的值会强烈惩罚重复，⿎励⽣成更多样化的⽂本。

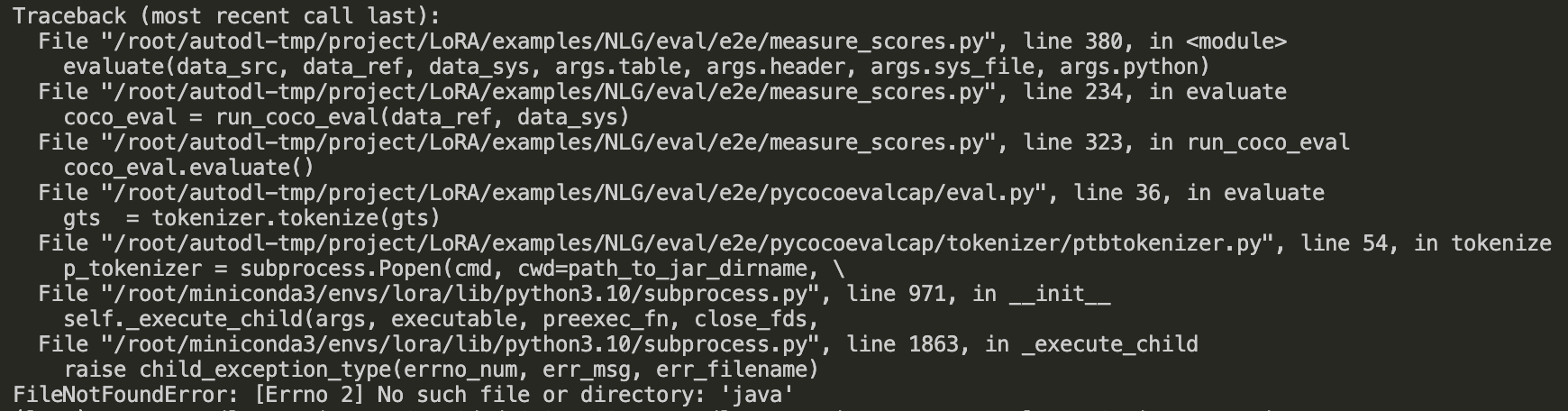
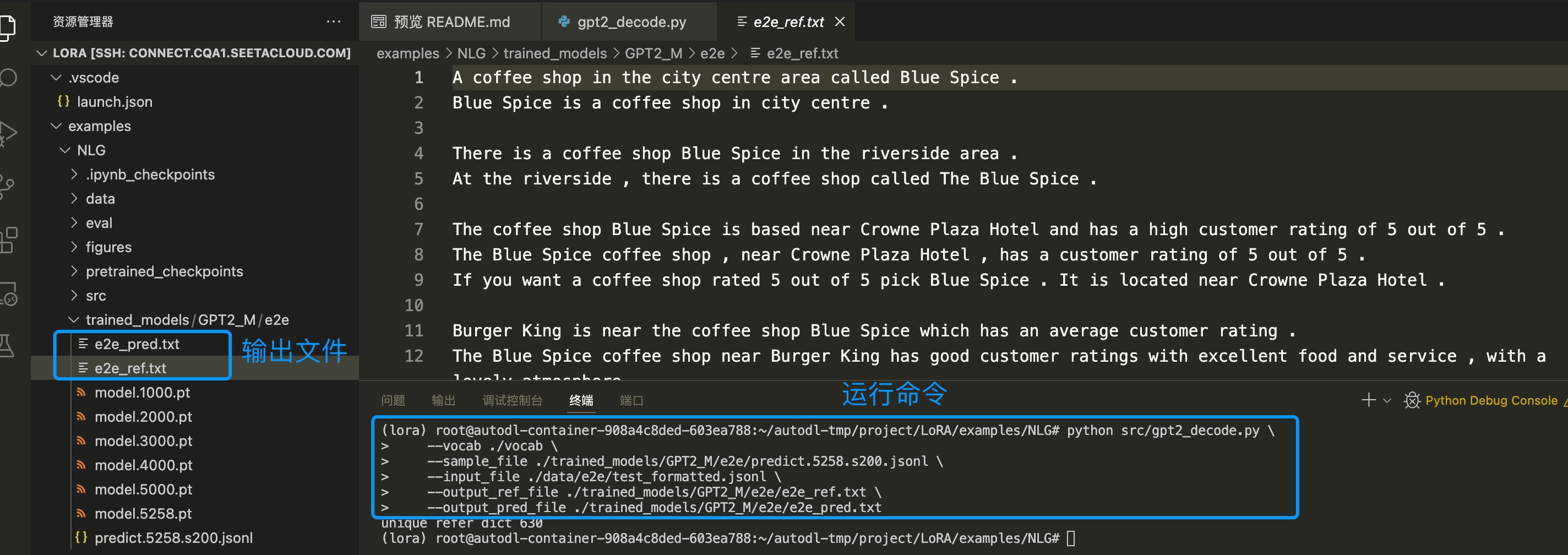
# 示例：

--repetition\_penalty 1.0：不进⾏重复惩罚。

--repetition\_penalty 1.2：轻微惩罚重复的词或短语，增加⽣成⽂本的多样性。

--repetition\_penalty 1.5：强烈惩罚重复的词或短语，显著增加⽣成⽂本的多样性。

**第6步：基于第5步的输出进⾏解码**



python src/gpt2\_decode.py \

--vocab ./vocab \

--sample\_file ./trained\_models/GPT2\_M/e2e/predict.5258.s200.jsonl

--input\_file ./data/e2e/test\_formatted.jsonl \

--output\_ref\_file ./trained\_models/GPT2\_M/e2e/e2e\_ref.txt \

--output\_pred\_file ./trained\_models/GPT2\_M/e2e/e2e\_pred.txt

\

**第7步：在E2E测试集上进⾏评估**

python eval/e2e/measure\_scores.py ./trained\_models/GPT2\_M/e2e/e2e\_ref.txt

./trained\_models/GPT2\_M/e2e/e2e\_pred.txt

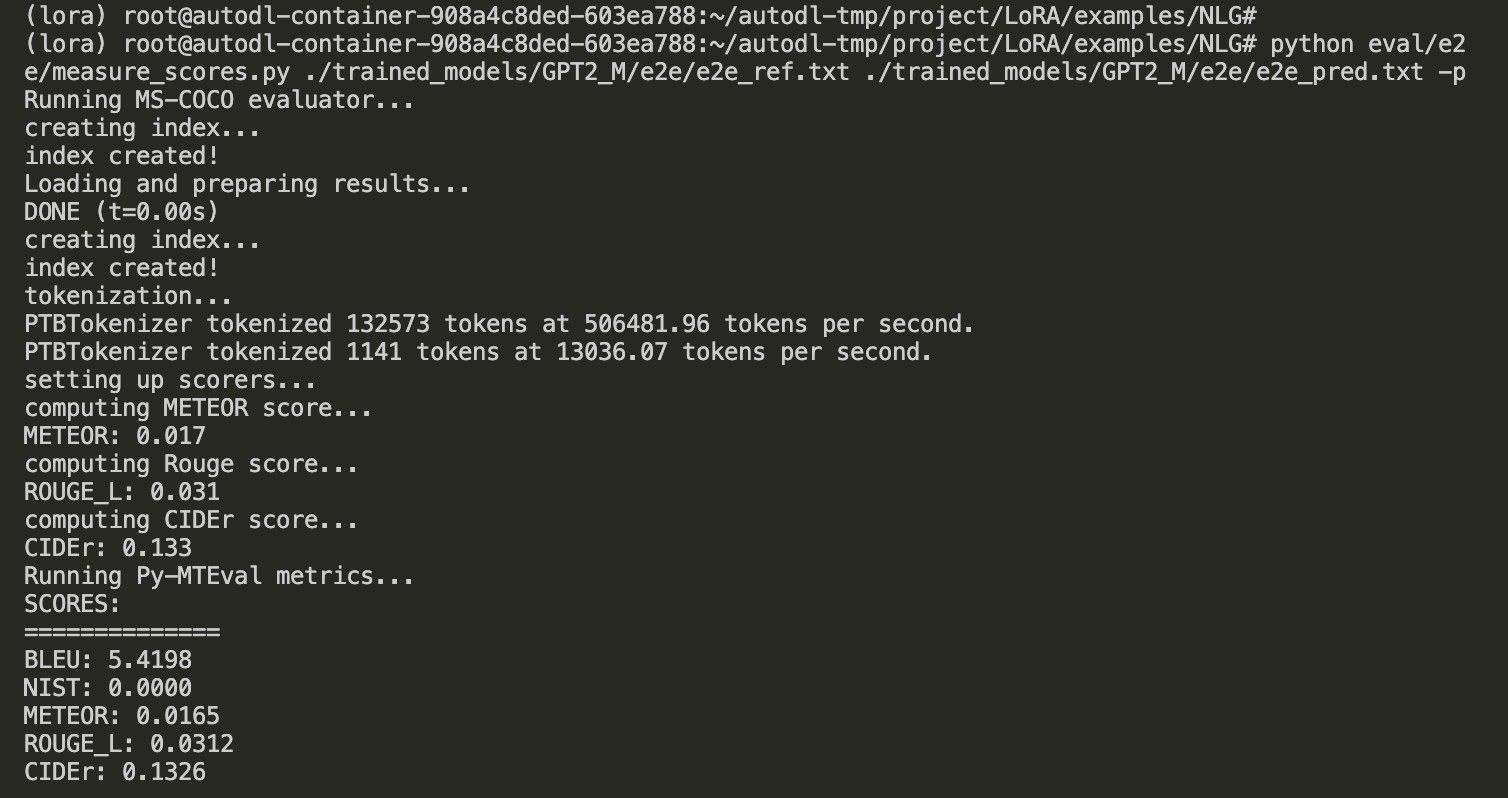
-p

**踩坑记录**

运⾏上⾯的命令后，抛出Error。如下图：

**解决措施：**

在系统上安装JAVA，



sudo apt update

sudo apt install default-jre

**解决Error后，再次运⾏命令，输出如下：**

**指标含义：**

**1. BLEU**

**含义**：

BLEU(Bilingual Evaluation Understudy) 是⼀种⽤于评估机器翻译和⾃然语⾔⽣成模型的精确度指标。它通过计算⽣成的⽂本与参考⽂本之间的 n-gram 匹配程度来评估⽣成⽂本的质量。

**计算⽅法**：

计算⽣成⽂本和参考⽂本中 n-gram 的精确匹配数。

计算不同 n-gram（通常是 1-gram 到 4-gram）的加权⼏何平均值。结合⻓度惩罚项，惩罚⽣成的⽂本⻓度与参考⽂本⻓度不⼀致的情况。

**公式**：

其中：

是⻓度惩罚（Brevity Penalty）。

是 n-gram 的权重（通常等权重）。



是⽣成⽂本和参考⽂本之间 n-gram 的精确匹配⽐例。

**2. NIST**

**含义**：

NIST(NIST Machine Translation Evaluation) 是 BLEU 的变种，它不仅考虑 n-gram 的精确匹配，还考虑匹配 n-gram

的信息量（即罕⻅ n-gram 的匹配会得到更⾼的得分）。

**计算⽅法**：

计算⽣成⽂本和参考⽂本中 n-gram 的精确匹配数。计算每个匹配 n-gram 的信息量。

结合⻓度惩罚项。

**公式**：

与 BLEU 类似，但对 n-gram 匹配的权重进⾏了加权处理，强调罕⻅ n-gram 的匹配。

**3. METEOR**

**含义**：

METEOR 通过考虑词形变化、同义词匹配和词序来评估⽣成⽂本的质量。

**计算⽅法**：

计算⽣成⽂本和参考⽂本之间的 unigram 精确匹配。计算词形变化、同义词匹配的匹配数。

结合词序惩罚项。

**公式**：

其中：

是精确度和召回率的调和平均。

是词序惩罚项。

**4. ROUGE\_L**

**含义**：

ROUGE\_L 基于最⻓公共⼦序列（LCS）来评估⽣成⽂本的覆盖度，强调⽣成⽂本中有多少部分与参考⽂本的顺序匹配。

**计算⽅法**：

计算⽣成⽂本和参考⽂本之间的最⻓公共⼦序列。



结合精确度和召回率。

**公式**：

其中：

精确度 (Precision) 和召回率 (Recall) 基于 LCS 计算。

是精确度和召回率的权重系数。

**5. CIDEr**

**含义**：

CIDEr (Consensus-based Image Description Evaluation) 主要⽤于图像描述⽣成任务，通过评估⽣成⽂本与参考⽂本的共识度来评估⽣成⽂本的质量。

**计算⽅法**：

计算⽣成⽂本和参考⽂本之间的 TF-IDF 加权 n-gram 相似度。

结合多个参考⽂本的共识度。

**公式**：

其中：

是参考⽂本数量。

是第 i 个参考⽂本的第 n-gram 相似度。