

RAG 技术详解与实战应用

第9讲：微调实践：

让大模型和向量模型更懂你的领域



目录



1. 上节回顾



2. 微调你的大模型



3. 微调模型的评测及应用



4. 微调你的Embedding模型



为什么要自定义Similarity

- 检索模块的效果直接影响生成结果的相关性与准确性
- 不同场景对“相似度”的定义差异较大
- LazyLLM内置的BM25/BM25_chinese和余弦相似度可能在实际应用中不能满足要求：
 - 领域语义理解不足**：BM25 基于词频，无法处理语义近义或领域术语归一问题；
 - 余弦相似度过于粗粒度**：在多句长文档或结构复杂文本中，局部语义匹配容易被稀释；

如何自定义Similarity

- 使用register_similarity来注册
- 核心参数配置
 - mode: text (文本计算) 或 embedding (向量计算)
 - descend: 结果降序排序 (默认 True)
 - batch: 是否批量计算相似度 (默认 False)
- 实战示例
- 基于TFIDF实现相似度计算。

自定义Similarity搭建RAG应用

- 定义Prompt
- 指定数据元
- 定义RAG数据流
 - 使用注册好的TFIDF进行相似度查询，优化查询结果
- 进行问答



目录



1. 上节回顾



2. 微调你的大模型



3. 微调模型的评测及应用



4. 微调你的Embedding模型



大模型在RAG流程的位置

RAG的流程回顾：

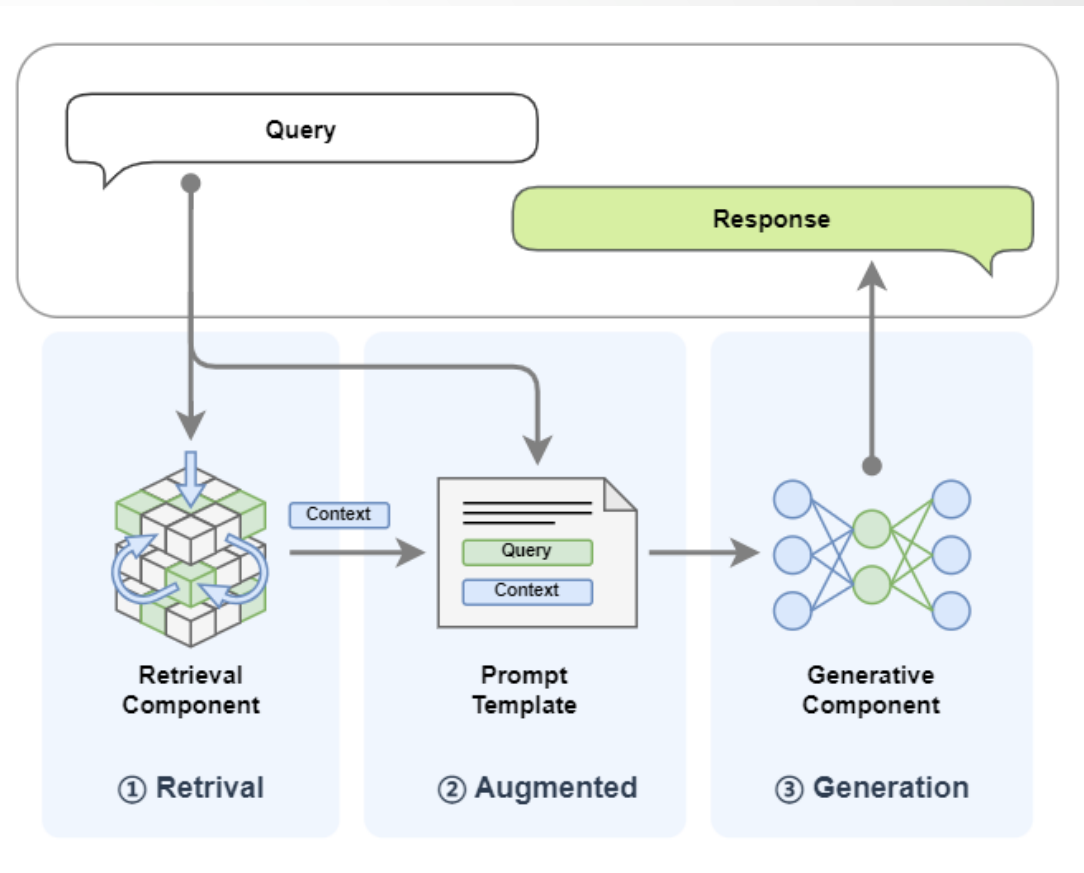
- **检索模块**会根据用户的Query去查询知识库，通过向量相似度来**召回相关的文档**；
- **生成模块**将检索结果与用户Query拼接，输入**大模型**生成最终响应Response；

从中可看出大模型在RAG系统中主要承担**内容生成**的作用，具体包括以下几个方面：

- **语义理解**：解析query的真实意图；
- **知识融合**：协调原始知识（大模型固有的知识）与新增知识（检索内容）的融合；
- **逻辑推理**：基于query的意图和上下文推理出一个合理的结果；

所以影响RAG系统精度主要有两个因素：

召回的效果 + **模型内容生成的能力**



在典型RAG架构中，大语言模型（LLM）的基准能力直接影响系统最终输出的可靠性，其性能瓶颈主要体现在以下两个维度：

1. 领域知识适配性缺陷

- **专业术语歧义**：跨领域缩写解析错误（如IC=重症监护/集成电路）
- **长尾知识缺失**：罕见病误诊/方言理解偏差/小众文化盲区
- **专业推理局限**：法律逻辑断层/复杂数学错误/复杂实验设计缺陷

2. 结构化输出控制薄弱

- **格式漂移**：JSON嵌套错误/符号缺失导致解析失败
- **幻觉干扰**：虚构医疗检查项/编造法律条文



什么是微调



➤ 定义

基于预训练大模型，通过特定领域数据二次训练，使模型适配专业场景的轻量化技术

➤ 核心方法

- 监督微调(SFT)：注入领域QA数据（如法律案例/医疗报告）
- 领域自适应：LoRA低秩适配技术，冻结原参数+训练适配层

➤ 核心优势

- **高效性**：7B模型经微调后，特定任务表现可比肩未调优的70B模型
- **低成本**：参数高效微调(PEFT)仅需更新0.1%~20%参数量
- **可控性**：强化结构化输出和内容输出约束（如JSON格式校准）

➤ 典型应用场景

- **领域知识强化**：医疗术语/法律条文定向注入
- **输出控制优化**：强制格式合规性（字段/层级/符号）
- **算力效率平衡**：小型模型经调优后替代云端大模型



微调对模型有什么提升 – 示例

这里我们列举一个具体的场景：

在医疗领域，需要将医疗报告中的文本信息（例如，“患者表现出高血压症状，收缩压为150mmHg”）转换为JSON格式，包含症状、血压读数等关键信息。

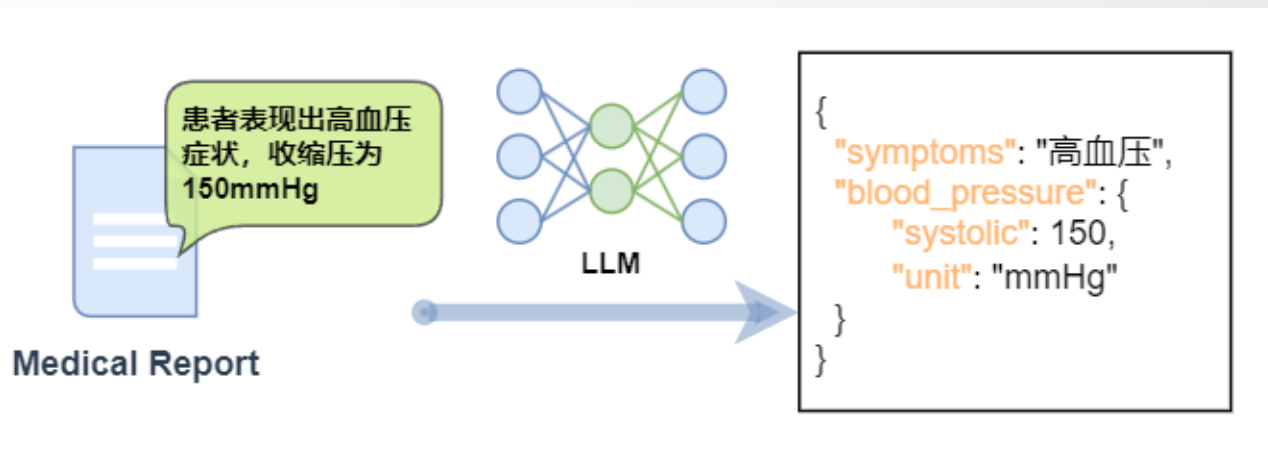
针对这类任务进行模型微调可以有以下益处：

1. 精确的信息提取：

微调可以帮助模型更准确地从自然语言文本中提取出需要结构化的信息，例如症状类型、数值、单位等。

2. 加强上下文理解：

自然语言中的信息往往依赖于上下文，微调可以让模型更好地理解这些上下文，从而更准确地转换信息。



3. 特定格式的要求：

JSON结构化文本通常有严格的格式要求，微调可以帮助模型生成符合这些要求的输出。

4. 领域特定术语的处理：

微调可以帮助模型更好地理解和处理专业术语。



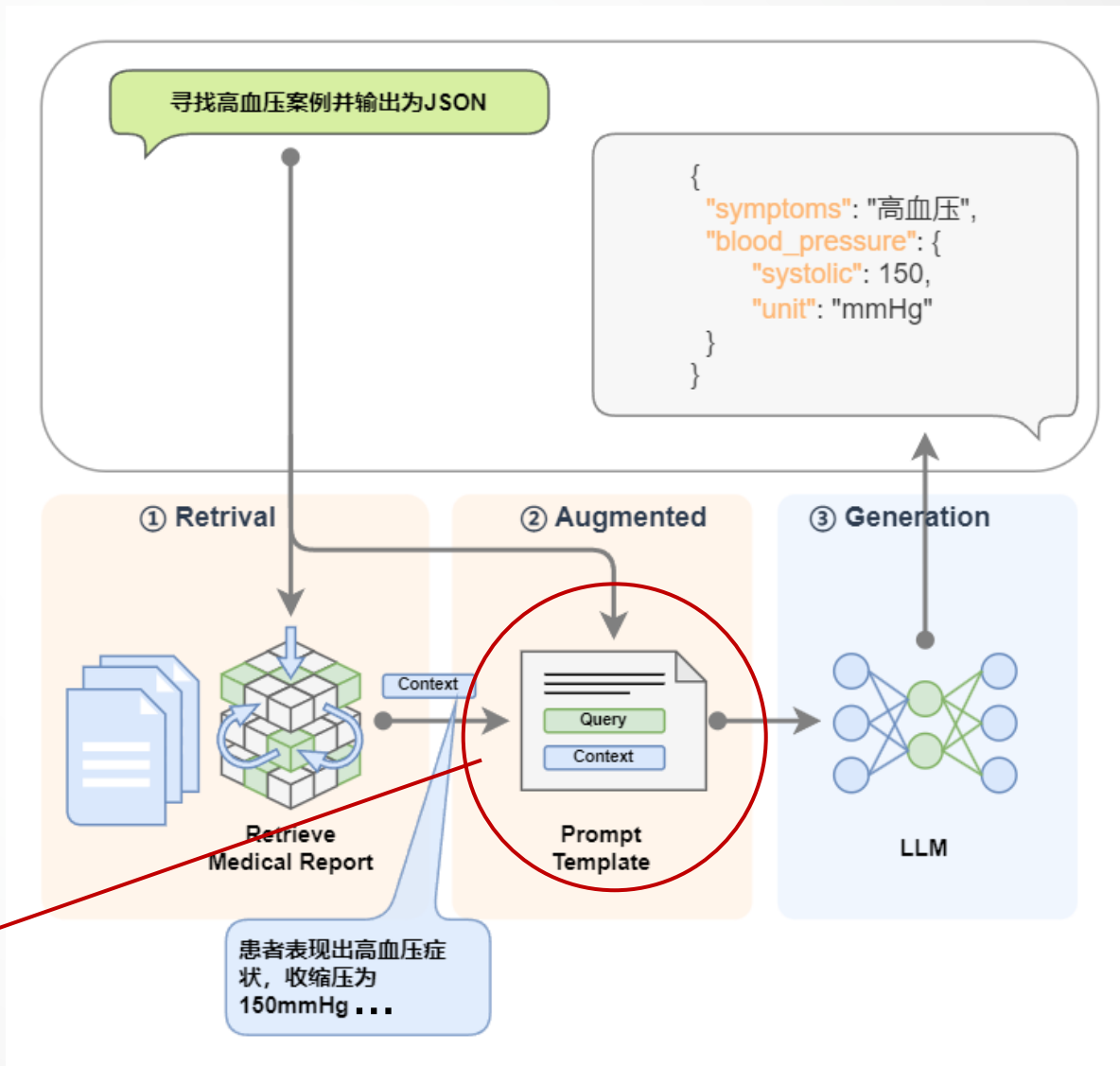
在RAG中使用微调好的模型

让我们将微调后的模型放到 RAG 中去，如右侧所示。下图是拼接好的完整prompt（即大模型的输入）

System-Prompt: 你是一个医疗报告提取器，需要基于用户的提问Query和上下文Context，从中提取出关键信息，提取的格式需要是JSON格式。

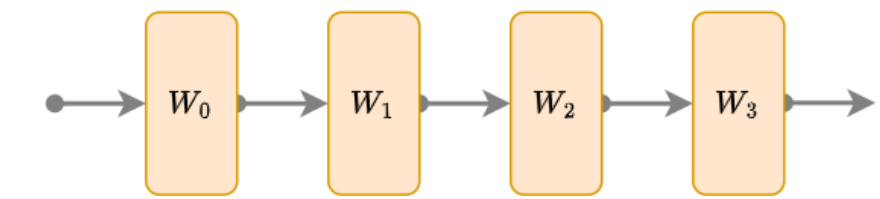
Query: 寻找高血压案例并输出为JSON

Context: 患者表现出高血压症状，收缩压为150mmHg，舒张压为100mmHg。患者自述近期经常感到头晕、头痛，并伴有间歇性心悸。体检发现患者BMI指数为28，属于超重范围。患者有长期吸烟史，每日约20支，且饮食偏咸。初步诊断为原发性高血压。建议患者立即戒烟，控制体重，低盐低脂饮食，并进行24小时动态血压监测。同时，给予药物治疗，初始方案为氨氯地平5mg，每日一次，并定期复查血压和电解质水平。

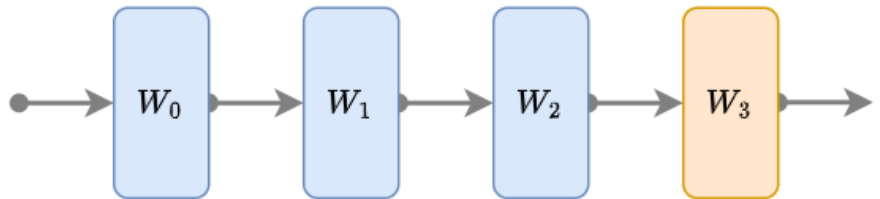


常用的微调方法对比

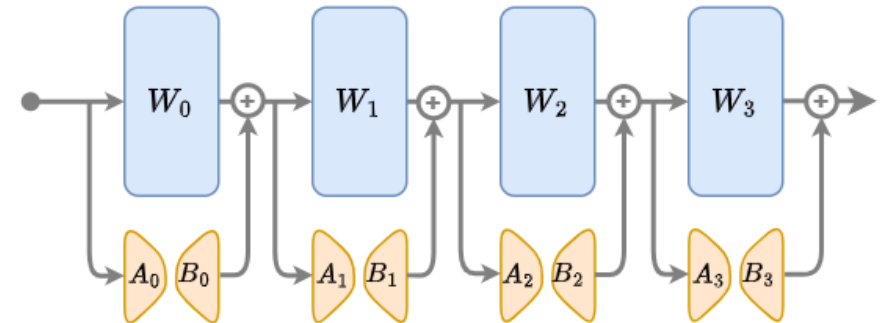
微调方法	计算资源消耗	参数更新范围	优势	劣势
全参数微调	高	所有参数	充分挖掘模型潜力, 适应性强	资源消耗大, 容易过拟合
冻结微调 (Freeze)	低	仅部分层参数	节省计算资源, 快速训练	性能可能不如全参数微调
LoRA微调	中	低秩矩阵	节省资源, 保持预训练优势	对模型结构有一定限制



Full Parameter Fine-tuning



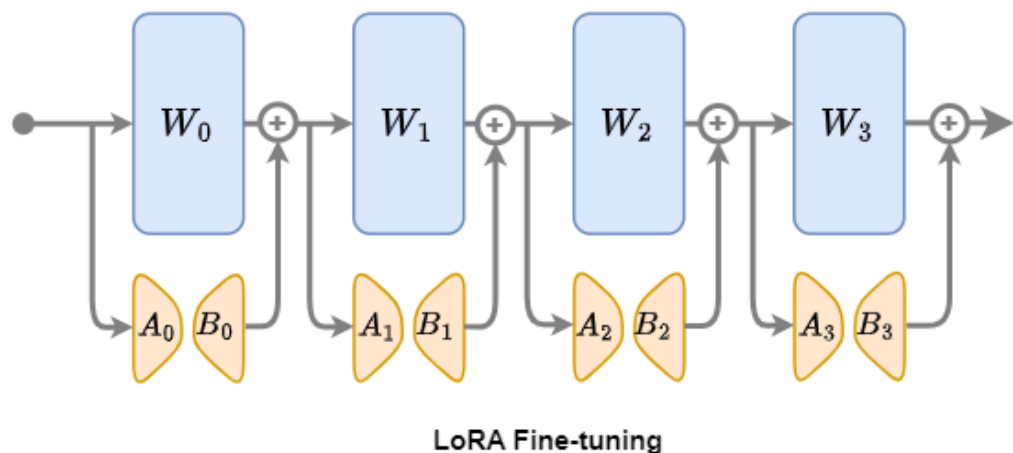
Freeze Parameter Fine-tuning



LoRA Fine-tuning



LoRA微调概述和步骤



LoRA微调是一种针对Transformer模型设计的先进微调技术。

核心思想： 在保持预训练模型参数 $W \in R^{d \times k}$ 固定不变的基础上，通过引入低秩矩阵 A 和 B 来模拟并实现微调过程。

特点：

- 不直接调整**原模型中的参数**（如 W_0 、 W_1 、 W_2 、 W_3 ），而是为每一层额外引入可训练的参数——**橙色矩阵** $B \in R^{d \times r}$ 和 $A \in R^{r \times k}$ ，仅对这些低秩矩阵进行微调。

LoRA微调步骤：

- 加载预训练模型：** 选择一个已经预训练好的Transformer模型作为基础。
- 引入低秩矩阵：** 在模型的每一层中，添加可训练的低秩矩阵 $B \in R^{d \times r}$ 和 $A \in R^{r \times k}$ 。
- 微调低秩矩阵：** 将特定任务的数据输入模型，利用反向传播算法针对低秩矩阵 A 和 B 进行参数更新。
- 评估与优化：** 在验证集上对微调后的模型进行性能评估，并根据实际需求进一步优化模型。



LoRA (Low-Rank Adaptation) 方法基于低秩矩阵近似理论，通过冻结预训练模型的参数并注入可训练的低秩矩阵，实现了参数的高效微调。在数学层面，对于预训练权重矩阵 $W \in R^{d \times k}$ ，LoRA 将其更新量分解为两个低秩矩阵的乘积：

$$\Delta W = B \cdot A$$

其中， $B \in R^{d \times r}$ ， $A \in R^{r \times k}$ ，且 $r \ll \min(d, k)$ 。这样的分解使得更新量具有更低的秩，从而减少了需要训练的参数数量。

在前向传播过程中，模型的输出变为：

$$h = Wx + \frac{\alpha}{r} B A x$$

其中：

- W 是预训练模型的原始权重矩阵
- x 是输入
- B 和 A 是低秩矩阵（秩为 r ）
- α 是缩放因子，用于灵活控制低秩矩阵对原始权重的调整强度。
- r 是秩



LoRA数学解析 - SVD



LoRA (Low-Rank Adaptation) 方法基于低秩矩阵近似理论，我们先基于奇异值分解（Singular Value Decomposition, SVD）来看低秩矩阵近似理论。SVD广泛应用于降维、特征提取、噪声过滤、推荐系统和自然语言处理（如 Latent Semantic Analysis）等领域。

SVD的基本概念：

对任意一个实矩阵 $A \in \mathbb{R}^{m \times n}$,

A可以被表达为: $A = U \Sigma V^T$,

其中

$U \in \mathbb{R}^{m \times m}$,

$\Sigma \in \mathbb{R}^{m \times n}$,

$V \in \mathbb{R}^{n \times n}$,

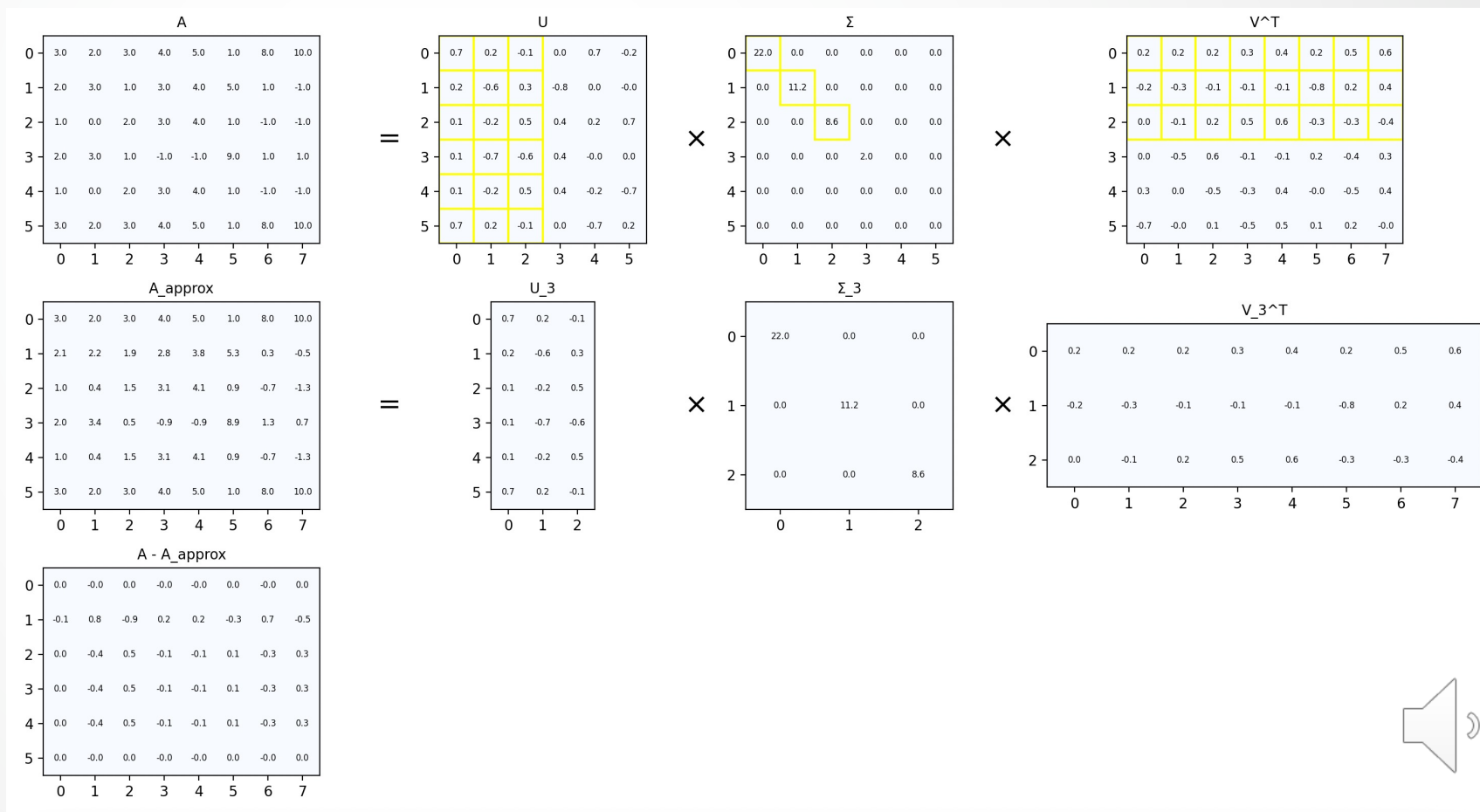
或

$U \in \mathbb{R}^{m \times k}$,

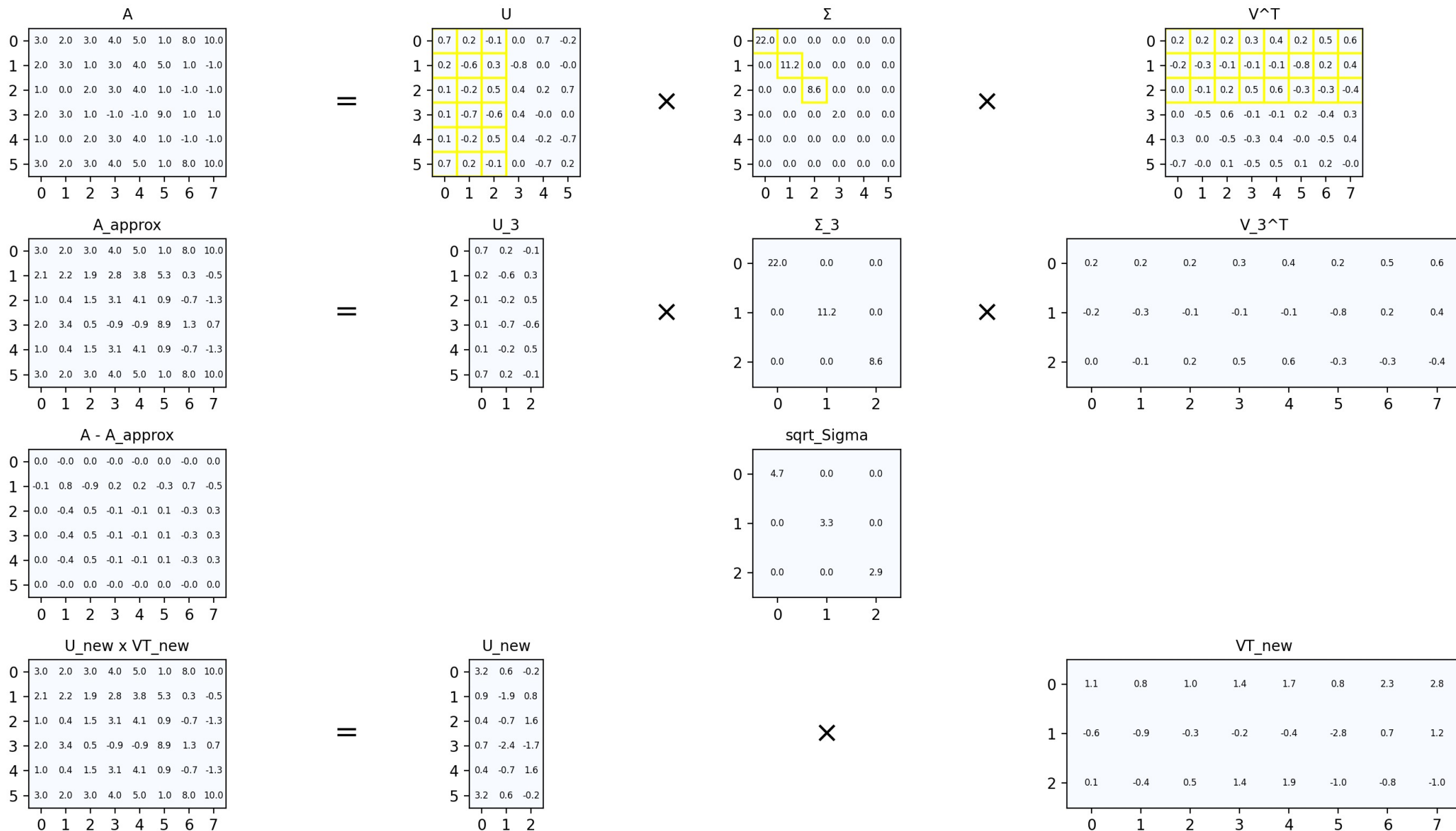
$\Sigma \in \mathbb{R}^{k \times k}$,

$V \in \mathbb{R}^{n \times k}$,

$k = \min(m, n)$



LoRA数学解析 - SVD



$$h = Wx + \frac{\alpha}{r}BAx$$

1. 秩 (r)

- **作用**：控制低秩矩阵 (BA) 的近似能力
 - 大 $r \rightarrow$ 高模型容量 (捕捉复杂更新) 但参数多 ($r \times (\dim_A + \dim_B)$)，易过拟合
 - 小 $r \rightarrow$ 参数少、训练快，但可能欠拟合
- **经验值**： $r=8$ 或 16 (平衡性能与效率)

2. 缩放因子 (α)

- **作用**：调节低秩更新项 (BAx) 的权重
 - 大 $\alpha \rightarrow$ 加速收敛但可能破坏预训练知识
 - 小 $\alpha \rightarrow$ 保留原始能力但训练慢
- **协同规则**： $\alpha = 2r$ (如 $r=8 \rightarrow \alpha=16$)

调优建议

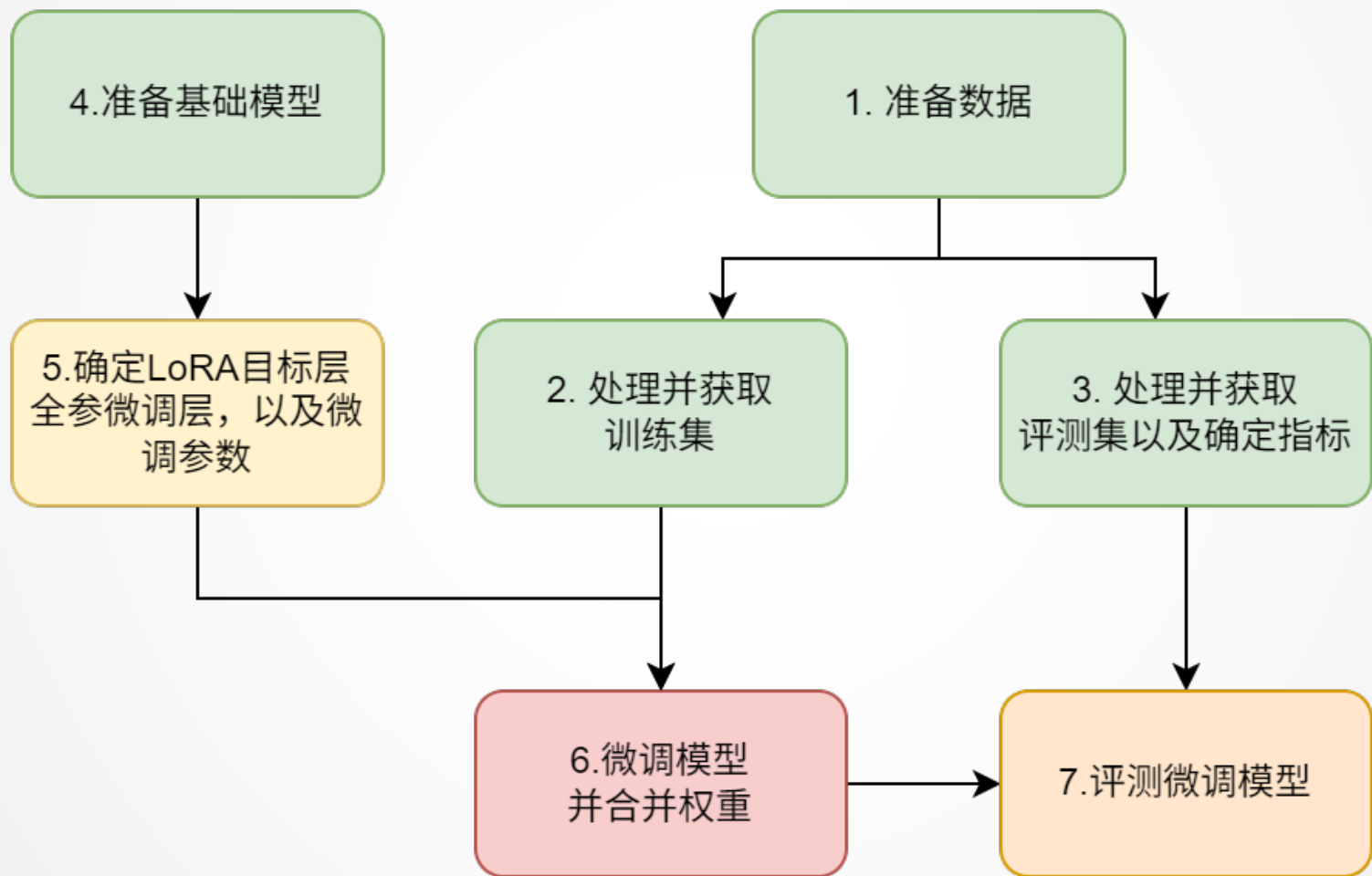
- **任务类型**：
 - 简单/小数据： $r=4$ + 中等 α
 - 复杂/大数据： $r=32$ + 高 α
- **资源限制**：显存不足时优先降 r

3. α/r 比例

- **直接影响**：
 - 高比例 (大 α /小 r)：快速适应新任务，但梯度不稳定
 - 低比例 (小 α /大 r)：温和更新，需更长时间收敛
- **与学习率关系**：类似低秩项的自适应学习率



微调步骤概述



基于LazyLLM对大模型进行微调



问题:

- 原RAG中大模型回答发散, 需精简+准确遵循原文输出;

目标:

- 在7B的小模型上, 强化中文阅读理解信息抽取能力

模型选择:

- InternLM2-Chat-7B。

数据准备:

- 使用CMRC2018训练数据, 聚焦中文问答与信息抽取。

微调方法:

- LoRA策略: 冻结原始权重, 仅训练低秩矩阵 (BA), 降低显存占用。
- 控制输出: 强化遵循原文内容且精简输出的能力。



数据集简介

- **名称**: CMRC2018 (中文阅读理解数据集)
- **来源**: 维基百科短文 + 人类专家标注
- **任务类型**: 篇章片段抽取型阅读理解 (Span-Extraction Reading Comprehension)
 - 输入: 文档 + 问题
 - 输出: 文档中连续的问题答案片段

数据集用途

1. **训练数据**: CMRC2018的Train数据 (2403篇短文及对应10142个问题)
2. **评测数据**: CMRC2018的Test数据 (256篇短文及对应的1002个问题)

关键说明

- **消除检索带来的误差**: 假设召回率100%, 即短文和问题完全匹配;
- **输入格式**: 文段 + 问题 → 拼接后输入大模型
- **对比目标**: 微调前后模型在相同输入下的答案抽取效果差异

数据集划分

数据集	短文数	问题数	用途
Test	256	1,002	评测集
Train	2,403	10,142	训练集
Validation	848	3,219	未使用



数据准备 - 数据的结构



数据集中的test和train的结构是一致的，我们抽取其中的一条数据来看如下：

```
[
  {
    "id": "TRIAL_154_QUERY_0",
    "context": "尤金袋鼠 (\`Macropus eugenii\`) 是袋鼠科中细小的成员，通常都是就袋鼠及有袋类的研究对象。尤金袋鼠分布在澳洲南部岛屿及西岸地区。由于牠们每季在袋鼠岛都大量繁殖，破坏了针鼹岛上的生活环境而被认为是害虫。尤金袋鼠最初是于1628年船难的生还者在西澳发现的，是欧洲人最早有纪录的袋鼠发现，且可能是最早发现的澳洲哺乳动物。尤金袋鼠共有三个亚种：尤金袋鼠很细小，约只有8公斤重，适合饲养。尤金袋鼠的奶中有一种物质，称为AGG01，有可能是一种神奇药及青霉素的改良。AGG01是一种蛋白质，在实验中证实比青霉素有效100倍，可以杀死99%的细菌及真菌，如沙门氏菌、普通变型杆菌及金黄色葡萄球菌。",
    "question": "尤金袋鼠分布在哪些地区？",
    "answers": {
      "text": [
        "尤金袋鼠分布在澳洲南部岛屿及西岸地区"
      ],
      "answer_start": [
        52
      ]
    }
  },
  ...
]
```

在左侧数据中：

- context: 是文本段；
- question: 是针对文本段的提问；
 - answers: 给出了对应问题的答案，包括：
 - text: 答案的具体内容，源自文本段；
 - answer_start: 答案在文本段中的起始位置；



训练集的构造

- 从原train数据集中抽取出文章字段context和问题字段question，将它拼接成为用于微调的instruction字段，拼接模板是：“请用下面的文段的原文来回答问题\n\n### 已知文段：{context}\n\n### 问题：{question}\n”；
- 将原数据中的answers中的text字段作为，微调的 output 字段；
- 由于微调还需要一个input字段，而我们这个任务不需要，所以设置为空。

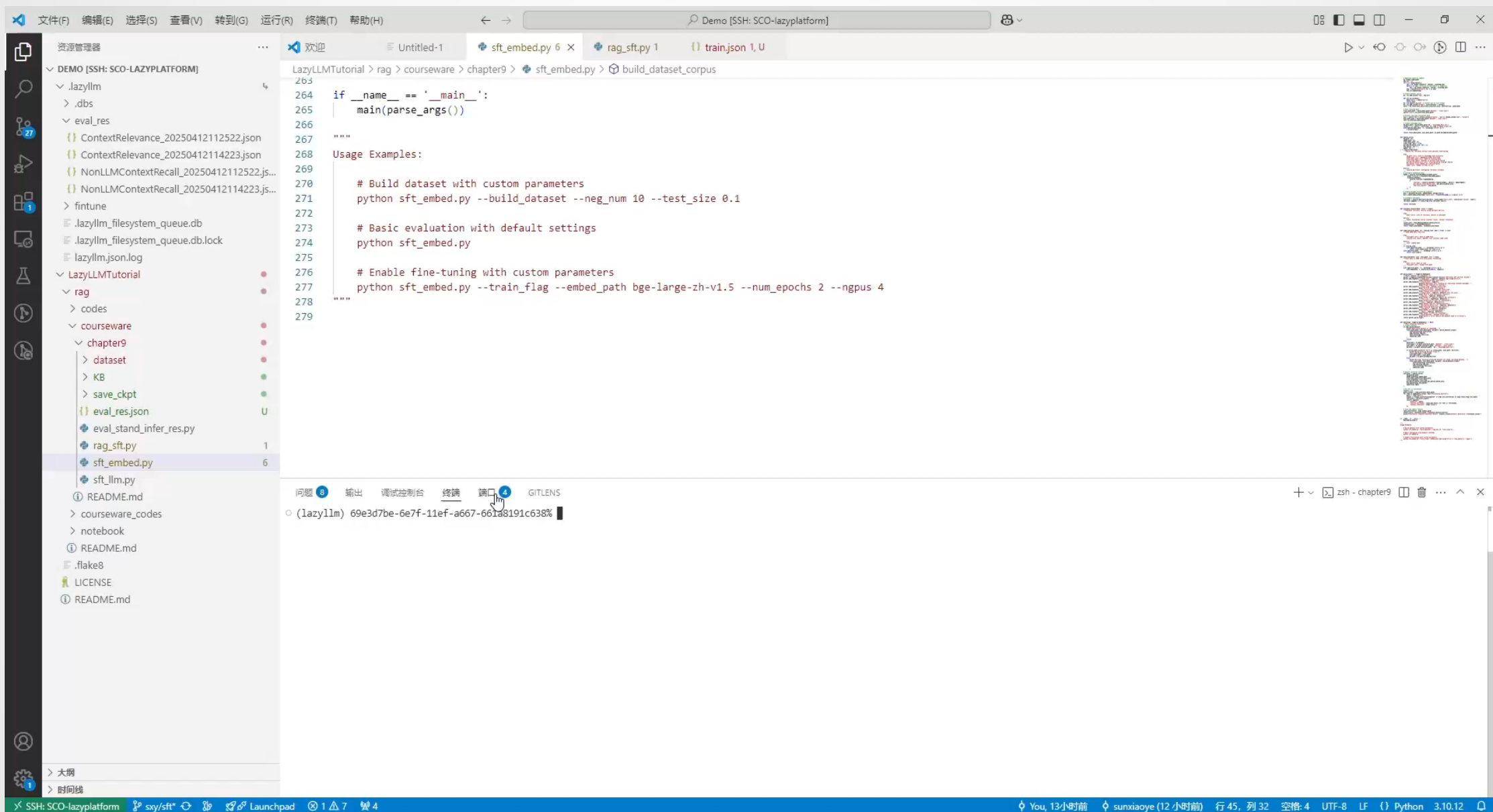
```
[  
  {  
    "instruction": "请用下面的文段的原文来回答问题\n\n### 已知文段：黄独（学名：）为薯蓣科薯蓣属的植物。多年生缠绕藤本。地下有球形或圆锥形块茎。叶腋内常生球形或卵圆形珠芽，大小不一，外皮黄褐色。心状卵形的叶子互生，先端尖锐，具有方格状小横脉，全缘，叶脉明显，7-9条，基出；叶柄基部扭曲而稍宽，与叶片等长或稍短。夏秋开花，单性，雌雄异株，穗状花序丛生。果期9-10月。分布于大洋洲、朝鲜、非洲、印度、日本、台湾、缅甸以及中国的江苏、广东、广西、安徽、江西、四川、甘肃、云南、湖南、西藏、河南、福建、浙江、贵州、湖北、陕西等地，生长于海拔300米至2,000米的地区，多生于河谷边、山谷阴沟或杂木林边缘，目前尚未由人工引种栽培。\\n\\n### 问题：黄独的外皮是什么颜色的？\\n",  
    "input": "",  
    "output": "外皮黄褐色"  
  },  
  ...  
]
```



评测集的字段基本都保留了下来，仅对answers字段进行了处理，将嵌套的内容提取了出来：

```
[  
  {  
    "context": "芙蓉洞位于重庆武隆县江口镇的芙蓉江畔，距武隆县城20公里。芙蓉洞于1993被发现，1994年对游人开放。2002年被列为中国国家4A级旅游景区，2007年6月作为中国南方喀斯特 - 武隆喀斯特的组成部分被列入联合国世界自然遗产，是中国首个被列入世界自然遗产的溶洞。芙蓉洞全长2846米，以竖井众多、洞穴沉积物类型齐全著称。芙蓉洞附近的天星乡境内，有世界上罕见的喀斯特竖井群。在约20平方公里的范围内，至少散布着50个超过100米的竖井，其中汽坑洞的深度为920米，为亚洲第一。芙蓉洞内有70多种沉积物，几乎包括了所有经过科学家命名的喀斯特洞穴沉积类型，其中池水沉积堪称精华。在芙蓉洞的东端有一个“石膏花支洞”，洞内的鹿角状卷曲石枝长57厘米，为世界第一。目前石膏花支洞被永久封存。在紧邻芙蓉洞洞口的芙蓉江上已经建起了江口电站大坝，水库蓄水后对地下水循环及喀斯特景观的演化造成的影响目前还难以估量。",  
    "question": "芙蓉洞位于什么地方？",  
    "answers": "重庆武隆县江口镇的芙蓉江畔"  
  },  
  ...  
]
```





The screenshot shows a VS Code editor window with the following components:

- File Explorer (Left):** Displays the project structure for 'DEMO [SSH: SCO-LAZYPLATFORM]'. The 'rag' directory is expanded, showing subdirectories 'codes' and 'courseware'. The 'chapter9' directory is also expanded, showing 'dataset', 'KB', 'save_ckpt', and 'eval_res.json'. The 'eval_res.json' file is selected.
- Editor (Center):** Displays the content of 'sft_embed.py'. The script includes a main function and usage examples for building a dataset, basic evaluation, and fine-tuning.
- Terminal (Bottom):** Shows the command prompt for the 'lazyllm' environment. The prompt is '(lazyllm) 69e3d7be-6e7f-11ef-a667-661a8191c638%'. The terminal is currently empty.
- Output (Right):** Displays the output of the 'eval_res.json' file, showing evaluation results for various models and datasets.

```
263
264 if __name__ == '__main__':
265     main(parse_args())
266
267 """
268 Usage Examples:
269
270 # Build dataset with custom parameters
271 python sft_embed.py --build_dataset --neg_num 10 --test_size 0.1
272
273 # Basic evaluation with default settings
274 python sft_embed.py
275
276 # Enable fine-tuning with custom parameters
277 python sft_embed.py --train_flag --embed_path bge-large-zh-v1.5 --num_epochs 2 --ngpus 4
278 """
279
```




```
1. import lazyllm
2. from lazyllm import finetune, deploy, launchers
3.
4. model = lazyllm.TrainableModule(model_path)\           # 指定要微调的模型
5.     .mode('finetune')\                               # 设置训练模式为：微调模式
6.     .trainset(train_data_path)\                       # 设置好刚处理的训练用的数据集路径
7.     .finetune_method((finetune.llamafactory, {        # 设置微调框架（Llama-Factory）及其参数
8.         'learning_rate': 1e-4,
9.         'cutoff_len': 5120,
10.        'max_samples': 20000,
11.        'val_size': 0.01,
12.        'per_device_train_batch_size': 2,
13.        'num_train_epochs': 2.0,
14.        'launcher': launchers.sco(ngpus=8)
15.    })))\
16.     .prompt(dict(system='You are a helpful assistant.',
17.                   drop_builtin_system=True))\         # 设置推理时候用的Prompt
18.     .deploy_method(deploy.Vllm)                      # 设置部署用的推理框架
19. model.evalset(eval_data)                             # 设置为刚处理好的评测集
20. model.update()                                       # 启动，依次完成：微调，部署，评测集推理
```



参数	作用	推荐设置	调优建议
learning_rate	控制参数更新幅度	1e-4~5e-5	大模型取较小值
cutoff_len	最大上下文长度	5120	根据GPU显存调整
max_samples	最大训练样本量	20000	注意太小导致训练用的数据太少,
per_device_train_batch_size	单卡批次大小	2	显存不足时减小
num_train_epochs	训练轮次	2.0	根据任务loss下降情况来设置

我们还可以配置一些LoRA相关的参数（LazyLLM已经默认设置好了一套经验的参数，所以上述代码中没有体现，这里我们展示如下，您可以尝试各类参数来炼丹：

参数	作用	推荐设置	说明
lora_alpha	LoRA 缩放因子	16	-
lora_dropout	LoRA 丢弃率	0.0	-
lora_rank	LoRA 秩	8	-
<u>lora_target</u>	LoRA 总用模型的目标模块	all	模型中的所有线性模块。



文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H) Demo [SSH: SCO-lazyplatform]

资源管理器

- DEMO [SSH: SCO-LAZYPLATFORM]
 - .lazyllm
 - .dbs
 - eval_res
 - fintune
 - .lazyllm_filesystem_queue.db
 - .lazyllm_filesystem_queue.db.lock
 - lazyllm.json.log
 - LazyLLMTutorial
 - rag
 - codes
 - courseware
 - chapter9
 - data
 - dataset
 - KB
 - saveckpt
 - eval_res.json
 - eval_stand_infer_res.py
 - infer_true_cp.json
 - rag_sft.py
 - sft_embed.py
 - sft_llm.py
 - README.md
 - courseware_codes
 - notebook
 - README.md
 - .flake8
 - LICENSE
 - README.md

欢迎

Untitled-1 sft_embed.py 6 sft_llm.py 5 rag_sft.py 1 train.json 1, U

LazyLLMTutorial > rag > courseware > chapter9 > sft_llm.py > ...

```
216 # Example Usage Patterns:
217 # 1. Baseline Evaluation:
218 # python sft_llm.py --mode="local_infer" --model_path="internlm2-chat-7b"
219 #
220 # 2. Fine-tuning and Evaluation:
221 # python sft_llm.py --mode="local_train"
222 #
223 # 3. Online Model Evaluation:
224 # python sft_llm.py --mode="online_infer" --model_path="DeepSeek-V3"
225 #
226 # 4. Score Calculation Only:
227 # python sft_llm.py --mode="score" --eval_res_path="path/to/results.json"
228 #
229 # 5. Build Dataset:
230 # python sft_llm.py --build_dataset"
231
```

问题 (13) 输出 调试控制台 终端 端口 (6) GITLENS

zsh - chapter9

o (lazyllm) 69e3d7be-6e7f-11ef-a667-661a8191c638%

SSH: SCO-lazyplatform sxy/sft* Launchpad 1 12 6 You, 13小时前 sunxiaoye (13 小时前) 行 223, 列 30 空格: 4 UTF-8 LF () Python 3.10.12



目录



1. 上节回顾



2. 微调你的大模型



3. 微调模型的评测及应用



4. 微调你的Embedding模型



评测目的

- **验证微调效果**：对比模型输出与标准答案，量化模型优化成果（是否超越通用大模型）
- **任务适配性检验**：通用指标（忠诚度、答案相关性）不适用于“篇章片段抽取”任务，需定制化评估
- **优化方向指导**：通过指标差异定位模型短板（如完全一致性、语义精准性、原文依赖性）

评测方法

1. 定制化指标设计

指标	定义	任务适配性
精确匹配率	预测与答案完全一致（0/1判断）	检验内容一致与格式规范性，避免表述差异干扰
语义相似度	基于BGE模型的余弦相似度（0-1连续值）	捕捉同义内容范围表达差异，衡量语义等价性
原文包含度	预测结果所有字均来自原文（0/1判断）	确保答案严格源自原文，杜绝幻觉

2. 评测流程

- **数据对齐**：测试集与推理结果逐项匹配
- **多维度计算**：遍历样本计算三指标得分（这里为了对比，也用通用指标进行计算）
- **结果聚合**：统计全局得分并输出对比报告



效果评测 – 评测结果对比

两个通用的评价指标并不适用于我们的“篇章片段抽取型阅读理解”任务，抽取评测例子来看：

答案相关度

模型名称	InternLM2-Chat-7B	DeepSeek-V3
上下文 (局部)	...每次泡封会减少生命数，生命数耗完即算为踢爆。重生者...	...每次泡封会减少生命数，生命数耗完即算为踢爆。重生者...
问题	生命数耗完即算为什么？	生命数耗完即算为什么？
真实答案	踢爆	踢爆
模型输出	踢爆	生命数耗完即算为踢爆。
精确匹配率	1	0
语义相似度	0.9999	0.6646
原文包含度	1	1
推测问题1	踢爆这个词语是什么意思？	在游戏术语中，“生命数耗完即算为踢爆”是什么意思？
推测问题1得分	0.3781	0.7191
推测问题2	踢爆是什么意思？这个词在什么语境下使用？	什么情况下会被视为踢爆？当生命数耗完时会怎样？
推测问题2得分	0.3825	0.6896
推测问题3	踢爆是什么意思？这个词通常在什么语境下使用？	在游戏规则中，生命数耗完后会发生什么？
推测问题3得分	0.3829	0.7918
答案相关度最终得分	0.3812	0.7335

简短下失效

在简短的answer之下，评价模型很难推测出和真实问题question相关的问题。所以表现相关度评分低。

忠诚度

模型名称	InternLM2-Chat-7B	DeepSeek-V3
上下文 (局部)	...国际象棋特级大师邀请赛，首届比赛于2008年12月11日至22日在南京市浦口区明发珍珠泉...	...国际象棋特级大师邀请赛，首届比赛于2008年12月11日至22日在南京市浦口区明发珍珠泉...
问题	首届比赛在哪里举行？	首届比赛在哪里举行？
真实答案	南京市浦口区明发珍珠泉大酒店	南京市浦口区明发珍珠泉大酒店
模型答案	南京市浦口区明发珍珠泉大酒店	首届比赛于2008年12月11日至22日在南京市浦口区明发珍珠泉大酒店举行。
精确匹配率	1	0
语义相似度	0.9999	0.5574
原文包含度	1	1
陈述1	首届比赛在南京市浦口区举行。	首届比赛于2008年12月11日至22日举行。
陈述1得分	1	1
陈述2	比赛的具体地点是明发珍珠泉大酒店。	比赛地点在南京市浦口区。
陈述2得分	1	1
陈述3	-	具体举办地是明发珍珠泉大酒店。
陈述3得分	-	1
最终得分	1.0	1.0

无区分度

两个模型都按照原文内容来回答，所以申述得分都一样（DeepSeek-V3给的多，但不准）。

精确匹配度的定义

$$EM = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(y_i = \hat{y}_i)$$

其中：

- N ：测试样本总数；
- y_i ：第 i 个样本的标准答案；
- \hat{y}_i ：模型预测结果；
- \mathbb{I} ：指示函数（完全匹配时取1，否则取0）；
- 该指标的特性在于：预测结果与标准答案需要**完全一致**

精确匹配度的实现

`exact_score = 1 if output == true_v else 0`

- `output`：模型对某个样本的预测结果。
- `true_v`：该样本的标准答案。
- `exact_score`：精确匹配得分，取值为1或0。如果预测结果与标准答案完全一致，则得分为1；否则得分为0。



语义相似度的定义

$$CS = \frac{1}{N} \sum_{i=1}^N \max \left(0, \min \left(1, \frac{emb(y_i) \cdot emb(\hat{y}_i)}{\|emb(y_i)\| \cdot \|emb(\hat{y}_i)\|} \right) \right)$$

其中：

- N ：测试样本总数；
- y_i ：第 i 个样本的标准答案；
- \hat{y}_i ：模型预测结果；
- $emb()$ ：是基于BGE模型 (bge-large-zh-v1.5) 的向量编码，它可以把自然语言编码为一个向量，即： $emb(text) = BGE_Encoder(text)$ ；

注意：该评价指标将原本 $[-1, 0)$ 的数值进行了截断，只要是负相关的语义都给0分，得分只能是正相关的。

语义相似度的实现

1. `def cosine(x, y):`
2. `product = np.dot(x, y)` # 计算两个向量的点积
3. `norm = np.linalg.norm(x) * np.linalg.norm(y)` # 计算两个向量的L2范数
4. `raw_cosine = product / norm if norm != 0 else 0.0` # 计算原始相似度
5. `return max(0.0, min(raw_cosine, 1.0))` # 截断原始相似度的负值



原文包含度的定义

$$OS = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(\forall w \in \hat{y}, w \in Context)$$

其中：

- N ：测试样本总数；
- \hat{y} ：模型预测结果；
- w ：表示预测结果中的每个字；
- $Context$ ：文章内容；
- \mathbb{I} ：指示函数（所有字在原文中出现时取1，否则有多于原文的字都取0）

原文包含度的实现

1. `def check_words_from_content(infer, content):`
2. `return 1 if all(w in content for w in infer.split()) else 0`



效果评测 – 评测指标 – 指标对比

指标维度	取值范围 (单项)	理想值 (单项)	数值特征 (单项)	优势	局限性	任务评价维度说明
精确匹配率	{0, 1}	1.0	二元判断	结果明确，无歧义	对表述差异零容忍	有多少推理能100%忠于答案
语义相似度	[0, 1]	1.0	连续数值	捕捉语义相似性	依赖编码模型质量	由于选取片段范围，或者表述有改变，所以用相似度来评价同一内容的不同表述；
原文包含度	{0, 1}	1.0	二元判断	确保答案忠实原文	忽略合理同义替换	任务要求，回答必须是用原文，此指标可反映答案是否都源自原文；



效果评测 – 综合评测



```
1. def caculate_score(eval_set, infer_set):
2.     # 部署本地 embedding 模型
3.     m = lazyllm.TrainableModule('bge-large-zh-v1.5').start()
4.     accu_exact_score = accu_cosin_score = accu_origi_score = 0
5.     res = []
6.     for eval_item, output in zip(eval_set, infer_set):
7.         output = output.strip()
8.         true_v = eval_item['answers']
9.         # 计算 精确匹配率:
10.        exact_score = 1 if output == true_v else 0
11.        accu_exact_score += exact_score
12.        # 计算 语义相似度:
13.        outputs = json.loads(m([output, true_v]))
14.        cosine_score = cosine(outputs[0], outputs[1])
15.        accu_cosin_score += cosine_score
16.        # 计算原文包含度:
17.        origin_score = check_words_from_content(output, eval_item['context'])
18.        accu_origi_score += origin_score
19.        res.append({'context':eval_item['context'], 'true': true_v, 'infer':output,
20.                    'exact_score': exact_score, 'cosine_score': cosine_score,
21.                    'origin_score': origin_score})
22.    # save_res(res, 'eval/infer_true_cp.json')
23.    return res
```



效果评测 – 评测结果对比

模型	针对任务设计的评价指标			通用评价指标	
	精确匹配率	语义相似度	原文包含度	答案相关度	忠诚度
Internlm2-Chat-7B	2.10%	74.51%	5.19%	87.14%	96.80%
DeepSeek-V3	5.29%	74.85%	15.17%	85.91%	96.90%
Internlm2-Chat-7B训 练后	39.72%	86.19%	94.91%	55.16%	96.50%



目录



1. 上节回顾



2. 微调你的大模型



3. 微调模型的评测及应用



4. 微调你的Embedding模型



Embedding模型在RAG系统中的作用

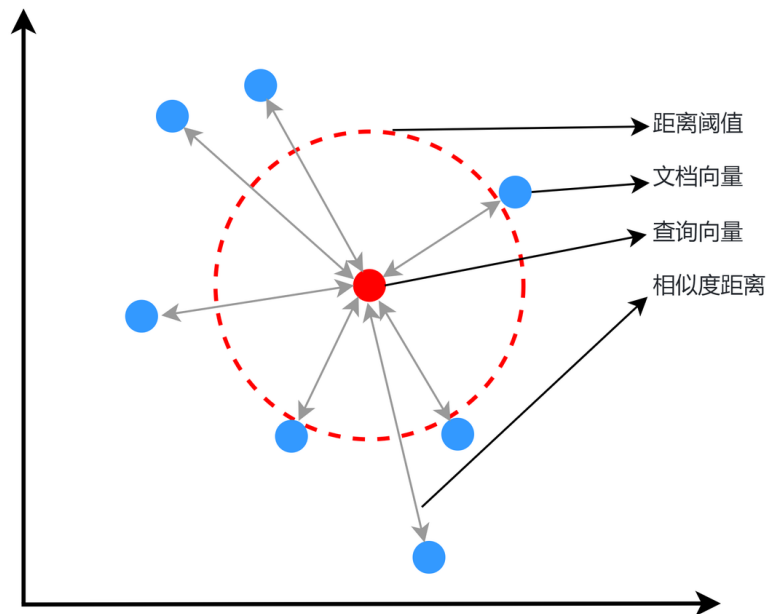
主要作用

1. 语义编码

- **功能**：将文本数据转换为高维向量表示，保留语义信息
- **重要性**：为后续相似度计算和检索提供基础，确保语义的准确表达

2. 相似度计算

- **方法**：通过向量余弦相似度实现高效的相关性检索
- **优势**：快速、准确地找到与用户查询最相关的知识片段



实践应用

- **基础模型**：BAAI的 bge-large-zh-v1.5
- **优化策略**：通过金融领域数据微调（SFT），提升垂直领域效果



Embedding模型微调 – 数据准备

数据集说明

- 来源：HuggingFace [virattt/financial-qa-10K]
- 字段：Question（用户提问）、Context（背景文本）




处理流程

- 加载原始数据
- 生成负样本（10负例/样本）
- 拆分训练集/评测集（9:1）
- 构建知识库（基于评测集）

Datasets: virattt/financial-qa-10K like 82

Split (1)
train · 7k rows

Search this dataset

question string · lengths	answer string · lengths	context string · lengths
 69→92 36.4%	 74→148 25.8%	 0→559 96.7%
What area did NVIDIA initially focus on before...	NVIDIA initially focused on PC graphics.	Since our original focus on PC graphics, we have expanded to...
What are some of the recent applications of...	Recent applications of GPU-powered deep learning...	Some of the most recent applications of GPU-powered...
What significant invention did NVIDIA create in 1999?	NVIDIA invented the GPU in 1999.	Our invention of the GPU in 1999 defined modern computer...
How does NVIDIA's platform strategy contribute to th...	NVIDIA's platform strategy brings together hardware,...	NVIDIA has a platform strategy, bringing together...
What does NVIDIA's CUDA programming model enable?	NVIDIA's CUDA programming model opened the parallel...	With our introduction of the CUDA programming model in...
What industries use NVIDIA's GPUs and softwar...	NVIDIA's GPUs and software are used for automation i...	A rapidly growing number of enterprises and startups...
Why did NVIDIA and SoftBank terminate their...	NVIDIA and SoftBank terminated their Share...	Termination of the Arm Share Purchase Agreement In Februar...



Embedding模型微调 – 数据准备



在 embedding 学习中，我们的目标是让：

- 语义相似的样本（**正样本对, positive pair**）在向量空间中更靠近。
- 语义无关或相反的样本（**负样本对, negative pair**）距离更远。

负样本的作用是提供一个**对比参照**，让模型知道“哪些是**不该靠近**”。

正样本对：

- query: “ChatGPT 是什么？”
- doc: “ChatGPT 是由 OpenAI 开发的语言模型，基于 Transformer 架构.....”

负样本对：

- query: “ChatGPT 是什么？”
- doc: “Midjourney 是一个 AI 图像生成模型.....”

处理后的训练集

```
{  
  "query": "What was the total...",  
  "pos": ["consolidated statements reflected"],  
  "neg": ["In June 202...", "repurchase $2.0 billion ..."],  
  "prompt": "Represent this sentence for searching  
relevant passages: "  
}
```

处理后的评测集

```
{  
  "query": "How have certain vendors...",  
  "corpus": ["Certain vendors have been impacted..."]  
}
```

处理后的知识库

(TXT File)

Certain vendors have been impacted by volatility in the supply chain financing market.
Recruitment As the demand for global technical ta



Embedding模型微调 – 数据准备



通过LazyLLM框架进行分布式微调

```
1. embed = lazyllm.TrainableModule(embed_path)\
2.     .mode('finetune').trainset(train_data_path)\
3.     .finetune_method((
4.         lazyllm.finetune.flagembedding,
5.         {
6.             'launcher': lazyllm.launchers.remote(
7.                 nnode=1, nproc=1, ngpus=4),
8.             'per_device_train_batch_size': 16,
9.             'num_train_epochs': 2,
10.        })
11.    )
12.)
13. docs = Document(kb_path, embed=embed, manager=False)
14. docs.create_node_group(name='split_sent', transform=lambda s: s.split('\n'))
15. retriever = lazyllm.Retriever(doc=docs, group_name="split_sent", similarity="cosine", topk=1)
16. retriever.update()
```

关键参数说明：

- **embed_path**: 指定微调的模型路径。
- **train_data_path**: 指定训练数据集的路径。
- **lazyllm.finetune.flagembedding**: 指定微调框架为flagembedding。
- **ngpus=4**: 使用4张GPU进行并行训练。
- **per_device_train_batch_size=16**: 每张GPU的批处理大小为16。
- **num_train_epochs=2**: 训练2个epoch。

- **文档切分策略**: 按照换行符分割知识库文档。
- **Retriever配置**: 作用于文档及其切分方式, 使用余弦相似度作为度量工具, 返回最相关的1个文本段 (topk=1) 。
- 执行update()后, 先对embed模型进行微调, 然后部署微调后的模型, 为Documet和Retriever提供向量化。



Embedding模型微调



File Editor Interface (VS Code) showing a file explorer on the left and a code editor on the right. The file explorer shows a project structure for 'DEMO [SSH: SCO-LAZYPLATFORM]' with folders like '.lazyllm', 'eval_res', 'fintune', 'LazyLLMTutorial', 'rag', 'courseware', 'chapter9', 'dataset', 'KB', and files like 'eval.json', 'train.json', 'knowledge_base.txt', 'eval_res.json', 'eval_stand_infer_res.py', 'sft_embed.py', 'sft_llm.py', 'README.md', 'courseware_codes', 'notebook', '.flake8', 'LICENSE', and 'README.md'.

The code editor displays the file 'sft_embed.py' with the following content:

```
280 # Basic evaluation with default settings
281 python sft_embed.py
282
283 # Enable fine-tuning with custom parameters
284 python sft_embed.py --train_flag --embed_path bge-large-zh-v1.5 --num_epochs 2 --ngpus 4
285 """
```

The bottom status bar shows the current file is 'sft_embed.py' (6, M) and the editor is running 'Python 3.10.12'.



Embedding模型微调 – 效果评测



评估结果

	微调前 bge-large-zh-v1.5	微调后 bge-large-zh-v1.5
上下文召回率	78.28	88.57
上下文相关度	75.71	86.57

评测指标体系（往期已介绍过）

- 上下文召回率
- 上下文相关度

评估流程

- 1.加载评测集
- 2.使用检索服务（微调后部署起来的服务）
- 3.执行批量推理
- 4.计算双指标

```
1. def evaluate_results(data: list) -> tuple:  
2.     recall_eval = NonLLMContextRecall(binary=False)  
3.     relevance_eval = ContextRelevance()  
4.     return recall_eval(data), relevance_eval(data)
```



在RAG中使用微调好的模型



```
1. import os
2. import lazyllm
3.
4. prompt = ('You will act as an AI question-answering assistant and complete a dialogue task.'
5.          'In this task, you need to provide your answers based on the given context and questions.')
6.
7. embed = lazyllm.TrainableModule('bge-large-zh-v1.5', 'path/to/sft/bge' )
8. llm = lazyllm.TrainableModule('internlm2-chat-7b', 'path/to/sft/llm' )
9.
10. documents = lazyllm.Document(dataset_path=os.path.join(os.getcwd(), "KB"), embed=embed, manager=False)
11. documents.create_node_group(name='split_sent', transform=lambda s: s.split('\n'))
12. with lazyllm.pipeline() as ppl:
13.     ppl.retriever = lazyllm.Retriever(
14.         doc=documents, group_name="split_sent", similarity="cosine", topk=1, output_format='content', join='')
15.     ppl.formatter = (lambda nodes, query: dict(context_str=nodes, query=query)) | lazyllm.bind(query=ppl.input)
16.     ppl.llm = llm.prompt(lazyllm.ChatPrompter(instruction=prompt, extra_keys=['context_str']))
17.
18. ppl.start()
```



感谢聆听
Thanks for Listening

