

LMCC-例卷-青少年组

单选题 (20 * 3 分 = 60 分)

1. 在掩码语言模型预训练中，核心任务“掩码预测”的主要目的是什么？

- A. 学习语言的深层双向上下文表征
- B. 评估模型最终的分类准确率
- C. 专门优化模型的文本生成流畅度
- D. 减少模型训练所需的计算资源

2. 小明和小华在学习大语言模型时，尝试让它写太空探险的故事。

小明的提示词是：“写一个关于太空探险的故事。”

小华的提示词是：“请你扮演一位经验丰富的科幻小说家，为中学生写一个大约 500 字的短篇故事。故事的主角是一位名叫‘星尘’的年轻宇航员，他在探索一颗未知行星时，意外发现了一个古老外星文明的遗迹。故事风格要紧张刺激，结局要留下悬念。”

结果小华生成的故事比小明更好。从提示词工程的角度分析，为什么小华能获得远比小明更好的结果？

- A. 因为小华提问时，大语言模型的知识库恰好更新了与“外星文明”相关的数据，所以能写出更具体的内容。
- B. 因为小华的提示词更长、更复杂，根据提示工程原理，投入的文字量与生成质量成正比，只要提示词足够长，效果就会更好。
- C. 因为小华的提示词为模型设定了明确的角色、目标受众和具体约束，有效地引导了模型的思维链，使其在庞大的可能性空间中聚焦于一个高质量的输出方向。
- D. 因为大语言模型具有真正的创造力和情感理解能力，小华的提示词中“紧张刺激”和“悬念”等词语激发了它的创作灵感。

3. Transformer 架构中，使大语言模型能够有效处理长距离依赖关系的关键组件是：

- A. 前馈神经网络
- B. 层归一化
- C. 自注意力机制

D. 残差连接

4. 指令微调的主要目的是什么？

- A. 让模型从零开始学习海量无标注数据中的通用知识。
- B. 增强模型理解和遵循人类指令或意图的能力，并改善其输出风格。
- C. 大幅降低模型的基础参数量，以提高推理速度。
- D. 专门用于修复模型在预训练阶段产生的事实性错误。

5. 大语言模型中存在的社会偏见，其主要根源最可能是：

- A. 模型训练时使用的 GPU 硬件存在设计缺陷。
- B. 模型 Transformer 架构中的注意力机制算法存在固有偏差。
- C. 模型从预训练数据中学习了现实世界中存在的社会偏见。
- D. 模型在指令微调阶段，工程师有意注入了有偏见的指令。

6. 目前主流的生成式大语言模型（如 GPT 系列）的核心架构通常基于：

- A. 编码器模型
- B. 编码器-解码器模型
- C. 解码器模型
- D. 卷积神经网络模型

7. 用于指令微调的数据通常具有什么典型特征？

- A. 大规模、无标注的原始网页文本。
- B. 由（指令/问题，期望输出）配对组成的监督数据集。
- C. 纯粹的代码仓库和编程语言片段。
- D. 未经处理的原始音频和图像数据。

8. 关于“贪心搜索”和“束搜索”这两种解码策略，以下哪种说法是最准确的？

- A. 贪心搜索因为每次都选最好的，所以总能生成最完美、最富有创造力的句子。
- B. 束搜索需要同时考虑多条路径，计算起来更复杂，但它更有可能找到一个整体上更

通顺、更合理的句子。

C. 束搜索的速度通常比贪心搜索更快，因为它是并行的。

D. 这两种策略没有本质区别，无论用哪一种，AI 生成的句子质量都完全一样。

9. 大模型“人类对齐”的核心目标是使得模型的行为：

A. 在所有的数学计算上达到零误差。

B. 无限接近地在所有任务上超越人类专家水平。

C. 符合人类的价值观、意图，并做到安全、有帮助。

D. 其内部神经网络的计算过程对人类完全透明可解释。

10. 大语言模型在解决一道复杂的数学应用题时，生成了一段非常长的思考过程，其中包含了将问题分解为多个子步骤、对每个步骤进行详细解释、并逐步推导出中间结果。这种推理模式的主要优势是什么？

A. 能够确保最终答案的正确性，避免计算错误。

B. 通过展示详尽的思维过程，提高了解题逻辑的可解释性和可靠性。

C. 显著减少了模型处理问题所需的总时间和计算资源。

D. 主要目的是为了生成更多的文本内容，使回答看起来更丰富。

11. 关于 AI 智能体的角色配置文件设置，下列哪个说法是最准确的？

A. 角色配置文件的主要作用是限制智能体的知识量，防止它回答超出范围的问题。

B. 角色配置文件设置一旦完成就无法改变，智能体将严格按照初始设定运行。

C. 角色配置文件是定义智能体核心身份、行为准则和对话风格的关键，通常通过初始系统指令实现。

D. 角色配置文件的功能与用户每次对话的提问内容作用相同，都是为智能体提供临时信息。

12. 在评估一个代码生成智能体时，我们常使用 **Pass@K** 指标。假设我们有一个包含 **n=100** 个编程问题的测试集。对于每个问题，让智能体独立生成 **k=5** 个不同的代码解决方案。如果对于某个问题，生成的 5 个方案中有任何一个能够通过单元测试，该问题就被视为“已解决”。最终，在 100 个问题中，有 60 个问题被成功解决。

根据上述描述，以下关于 **Pass@K** 的计算和解释，哪一项是正确的？

- A. 这里 $K=5$, $\text{Pass}@5 = (60 / 100) * 100\% = 60\%$ 。这表示随机抽取一个生成方案，其通过测试的概率是 60%。
- B. 这里 $K=5$, $\text{Pass}@5 = (60 / 100) * 100\% = 60\%$ 。这表示通过生成 5 个方案，智能体成功解决其中 60% 问题的能力。
- C. 这里 $K=100$, $\text{Pass}@100 = (60 / 100) * 100\% = 60\%$ 。这表示智能体解决了测试集中 60% 的问题。
- D. 这里 $K=5$, 但正确的计算方式应为 $\text{Pass}@5 = (60 / (100 * 5)) * 100\% = 12\%$ 。这表示所有生成方案的总通过率。

13. 下列哪一项通常不属于对大语言模型进行系统评测的核心内容？

- A. 模型训练所用 GPU 集群的实时功耗与散热效率
- B. 模型在问答、摘要、代码生成等任务上的准确率与流畅度
- C. 模型输出是否存在社会偏见、歧视性内容等安全性问题
- D. 模型在数学推理、常识推理等复杂任务上的泛化能力

14. 在 Transformer 的自注意力机制中，关于 Query、Key 和 Value 的来源，以下哪一项描述是最准确的？

- A. Query、Key 和 Value 是由三个完全不同的、随机初始化的参数矩阵生成的，与输入序列无关。
- B. Key 和 Value 是输入序列的嵌入向量本身，而 Query 是目标序列的嵌入向量本身。
- C. Query 来自编码器的最终隐藏状态，而 Key 和 Value 来自解码器的输入序列。
- D. Query、Key 和 Value 是由同一个输入向量通过三个不同的线性变换层（权重矩阵）投影得到的。

15. 下列哪一项不属于测试时间拓展（Test Time Scaling, TTS）技术？

- A. 对单个测试样本进行多种变换（如翻转、裁剪），并将所有变换版本的预测结果进行集成。
- B. 在模型推理时，仍然随机丢弃部分神经元，进行多次前向传播并将结果平均，以模拟集成效果提升健壮性。
- C. 为每个测试样本定义一个辅助任务（如旋转图片预测），利用该样本对模型进行一步梯度下降微调，再完成主任务预测。
- D. 在一个大型源数据集上预训练模型，然后在一个较小的目标数据集上对模型的所有权重进行微调。集上对模型的所有权重进行微调。

16. 为了保障社会群体的“公平安全”，避免模型输出产生歧视性内容，最关键的缓解方法是在哪个阶段介入？

- A. 仅在模型推理部署后，通过实时过滤敏感词来屏蔽有害输出。
- B. 主要依靠扩大模型规模，期望其自动学习并消除偏见。
- C. 在模型预训练和微调阶段，系统性地清洗数据和引入公平性约束。
- D. 完全依赖第三方评测机构在发布前进行一次性的偏见评估。

17. 当前意义上的“大语言模型”最显著的特征不包括以下哪一项？

- A. 模型参数量巨大，但仅在特定领域的小规模数据上训练。
- B. 展现出强大的涌现能力，如推理、编程和创造性写作。
- C. 基于“下一个词预测”等自监督目标在海量互联网文本上训练。
- D. 具有强大的上下文学习能力，无需梯度更新即可根据提示完成新任务。

18. 基于强化学习的推理能力优化方法面临的主要挑战是什么？

- A. 模型会因此完全丧失其原有的语言生成能力。
- B. 推理路径的搜索空间巨大，导致训练不稳定和效率低下。
- C. 强化学习会使模型的词汇量急剧减少。
- D. 该方法只能应用于数学推理，无法泛化到其他领域。

19. 关于语言模型的困惑度，以下描述正确的是：

- A. 困惑度越低，说明模型对测试数据的预测越不确定，性能越差。
- B. 困惑度是模型在训练集上的准确率直接转换而来的指标。
- C. 困惑度反映模型对一组文本的平均预测不确定性，值越低表示模型性能越好。
- D. 困惑度与模型的训练步数无关，只与模型架构有关。

20. 关于监督学习和无监督学习的区别，以下说法正确的是：

- A. 监督学习不需要训练数据，无监督学习需要大量标注数据
- B. 监督学习使用标注数据学习，无监督学习从无标签数据中发现模式

C. 无监督学习只能处理数值型数据，监督学习可以处理任何类型数据

D. 无监督学习的模型性能总是优于监督学习

程序题 (10 * 4 分 = 40 分)

一、检索增强生成

这是一个基于检索增强生成 (Retrieval-Augmented Generation, RAG) 的问答系统实现。该系统包含两个主要组件：

1. 检索器 (Retriever)：使用嵌入模型将文档转换为向量，根据查询语义检索最相关的文档

2. 生成器 (Generator)：基于检索到的相关文档和原始问题，生成准确的答案

请根据代码逻辑和注释提示，补全代码中的空位。

```
Python
import torch
import torch.nn.functional as F
from transformers import AutoTokenizer, AutoModel,
AutoModelForCausalLM

class Retriever:
    def __init__(self, embedder, tokenizer, corpus):
        self.embedder = embedder
        self.tokenizer = tokenizer
        self.corpus = corpus
        self.embeddings = self.build_index(corpus)

    def _text_to_vector(self, text_batch):
        enc = [21]
        enc = {k: v.to(self.embedder.device) for k, v in
enc.items()}

        with torch.no_grad():
            outputs = [22]
            last_hidden = outputs.last_hidden_state
            attn_mask = enc["attention_mask"]
            sent_vecs = [23]

        return sent_vecs
```

```

def build_index(self, corpus):
    vecs_tensor = self._text_to_vector(corpus)
    vecs_normalized = F.normalize(vecs_tensor, p=2, dim=1)
    return vecs_normalized.cpu()

def search(self, query_vec, topk=3):
    if query_vec.dim() == 1:
        query_vec = query_vec.unsqueeze(0)
    query_normalized = F.normalize(query_vec, p=2, dim=1)
    similarities = torch.matmul(query_normalized,
self.embeddings.T)
    similarities = similarities.squeeze(0)

    topk_scores, topk_indices = torch.topk(similarities,
k=min(topk, len(self.corpus)))

    results = [
        {"text": self.corpus[idx], "score":
float(topk_scores[i])}
        for i, idx in enumerate(topk_indices)
    ]
    return results

def rag_answer(query, retriever, generator, gen_tokenizer, topk=3,
max_new_tokens=128):
    query_vec_tensor = retriever._text_to_vector([query])
    query_vec = query_vec_tensor.squeeze(0).cpu()

    docs = [24]

    context = "\n".join([d["text"] for d in docs]) if docs else ""
    prompt = f"Use the context to
answer.\nContext:\n{context}\n\nQ: {query}\nA:"

    inputs = gen_tokenizer(prompt, return_tensors="pt")
    inputs = {k: v.to(generator.device) for k, v in
inputs.items()}
    pad_id = gen_tokenizer.pad_token_id if
gen_tokenizer.pad_token_id is not None else
gen_tokenizer.eos_token_id

    with torch.no_grad():
        output_ids = generator.generate(

```

```

        **inputs,
        do_sample=True, temperature=0.7, top_p=0.9,
        max_new_tokens=max_new_tokens,
        pad_token_id=pad_id,
        eos_token_id=gen_tokenizer.eos_token_id,
    )

    new_tokens = [25]

    answer = gen_tokenizer.decode(new_tokens,
        skip_special_tokens=True).strip()
    return answer

if __name__ == "__main__":
    embedder_name = "../Qwen/Qwen3-Embedding-0.6B"
    retriever_tokenizer =
AutoTokenizer.from_pretrained(embedder_name)
    embedder = AutoModel.from_pretrained(embedder_name)
    generator_name = "../Qwen/Qwen3-0.6B"
    gen_tokenizer = AutoTokenizer.from_pretrained(generator_name)
    generator =
AutoModelForCausalLM.from_pretrained(generator_name)
    corpus = [
        "Paris is the capital of France.",
        "Tokyo is the capital of Japan.",
        "Beijing is the capital of China.",
    ]
    retriever = Retriever(embedder, retriever_tokenizer, corpus)
    query = "What is the capital of France?"
    ans = rag_answer(query, retriever, generator, gen_tokenizer,
        topk=2)
    print("Q:", query)
    print("A:", ans)

```

21. 在方法 `_text_to_vector` 中第一步是将输入的文本批次 `text_batch` 转换为模型可以理解的格式，以下代码正确的是：

- A. `self.embedder(text_batch, return_tensors="pt", truncation=True, max_length=512)`
- B. `self.tokenizer(text_batch, return_tensors="pt", truncation=True, max_length=512)`
- C. `self.tokenizer.encode(text_batch, return_tensors="pt", truncation=True, max_length=512)`
- D. `self.embedder(text_batch return_tensors="pt", truncation=True, max_length=512)`

22. 此处需要进行前向计算获取隐藏向量，以下代码正确的是：

- A. `self.embedder(**enc)`
- B. `self.embedder.forward(enc["input_ids"])`
- C. `self.embedder(enc["input_ids"])`
- D. `self.embedder.forward(enc)`

23. 此处需要池化得到句向量，以下代码正确的是：

- A. `last_hidden.mean(1)`
- B. `(last_hidden * attn_mask.unsqueeze(-1)).sum(1) / attn_mask.sum(1, keepdim=True)`
- C. `last_hidden[:, 0, :]`
- D. `(last_hidden * attn_mask).sum(1) / attn_mask.sum(1)`

24. 此处需要进行检索得到对应文档，以下代码正确的是：

- A. `retriever.index.search(query_vec, topk)`
- B. `retriever.search(query_vec, topk=topk)`
- C. `retriever.build_index(query_vec)`
- D. `retriever.generate(query_vec, topk=topk)`

25. 此处需要对模型输出进行处理，以下代码正确的是：

- A. `output_ids[0][-max_new_tokens:]`
- B. `output_ids[0][inputs["input_ids"].shape[1]:]`
- C. `output_ids[:, -max_new_tokens:]`
- D. `output_ids[:, inputs["input_ids"].shape[1]:]`

二、参数高效微调

低秩适配方法 LoRA (Low-Rank Adaptation) 是一种参数高效微调方法，通过只训练部分低秩矩阵插入至原始模型的关键层，实现大语言模型的高效定制。与传统微调相比，LoRA 显著减少了训练参数量和显存需求，适用于资源受限场景下的大模型下游训练。一个典型训练场景如下：

- 模型: **Qwen-2.5-7B-Instruct**; LoRA 仅插入注意力机制中关键的两个投影层。
- 资源: **单卡 16GB 显存**; 输入序列长度上限 **2048**。
- 训练策略: 需将全局 batch size 控制在 8 左右, 在显存足够的情况下**尽量快**的进行训练。
- 精度: 启用半精度训练以降低显存占用。
- 产出: 训练完成后, 仅保存 LoRA adapter 权重, 供下游推理使用。

请阅读以下训练代码, 并根据需求完成空缺部分。

```
Python
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM,
TrainingArguments, Trainer
from peft import LoraConfig, get_peft_model

model_name = "Qwen/Qwen2.5-7B-Instruct"

# 1) 加载分词器
tokenizer =         [26]        
model = AutoModelForCausalLM.from_pretrained(
    model_name, device_map="auto", torch_dtype=torch.float16,
    trust_remote_code=True
)

# 2) 配置 LoRA
lora_config = LoraConfig(
            [27]        ,
    target_modules=        [28]        ,
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)

model = get_peft_model(model, lora_config)

# 3) 定义 Trainer (假设 train_dataset 已经准备好)
training_args = TrainingArguments(
    output_dir="./lora-qwen2.5-7b",
            [29]        ,
    num_train_epochs=3,
    learning_rate=2e-4,
            [30]        
```

```

        logging_steps=10,
        save_strategy="epoch"
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        tokenizer=tokenizer,
    )

    trainer.train()

# 4) 保存 LoRA 权重
model.save_pretrained("./lora-qwen2.5-7b-lora")

```

26. Qwen2.5 可能包含自定义分词器/处理逻辑，要求与模型端一致的 remote code。请选择一种能正确加载该模型分词器的方式：

- A. AutoTokenizer.from_pretrained(model_name, return_tensors="pt")
- B. AutoTokenizer.load(model_name)
- C. AutoTokenizer(model_name)
- D. AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)

27. 请参考题目描述的计算资源，在尽量保持训练稳定和速度的前提下，选择常见且相对节省显存的 LoRA 配置：

- A. r=64, lora_alpha=32,
- B. r=8, lora_alpha=64,
- C. r=8, lora_alpha=16,
- D. r=32, lora_alpha=8,

28. 为了配置 LoRA 权重的注入位置，以下代码更合理的是：

- A. ["mlp", "norm"]
- B. "all_linear_layers"
- C. ["k_proj", "o_proj"]
- D. ["q_proj", "v_proj"]

29. 显存限制和内存溢出 (Out Of Memory, OOM) 风险下, 以下配置最稳妥的是:

- A. per_device_train_batch_size=2, gradient_accumulation_steps=4,
- B. global_train_batch_size=8, gradient_accumulation_steps=4,
- C. global_train_batch_size=8, per_device_train_batch_size=1,
- D. per_device_train_batch_size=1, gradient_accumulation_steps=8,

30. 为了配置训练的数据类型, 以下代码更合理的是:

- A. fp32=True,
- B. fp16=True,
- C. training_type="fp32",
- D. training_type="bf16",

答案

单选题:

- 1-5 ACCBC
- 6-10 CBBCB
- 11-15 CBADD
- 16-20 CABCB

程序题:

- 21-25 BABBB
- 26-30 DCDDDB