

LMCC-例卷-成人组

单选题 (20 * 3 分 = 60 分)

1. 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
2. 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。
3. 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。
4. 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。

5. 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。

6. 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。

7. 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。

8. 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

9. 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。

10. 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。

11. 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。

12. 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。

13. 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。

14. 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。

15. 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

16. 在使用低秩自适应法(Low-Rank Adaptation, LoRA)对一个大语言模型进行微调时, 以下哪个操作是其典型的流程?

- A. 首先对原始模型进行奇异值分解, 保留主要成分以降低其秩, 然后在这个低秩模型上进行全参数微调。
- B. 在模型的注意力机制模块和前馈网络的线性层旁, 并联一对低秩矩阵 (A 和 B), 在微调时只训练这对矩阵, 并将它们的乘积与冻结的原始权重相加作为该层的实际输出。
- C. 准备两套模型参数, 一套保持冻结, 另一套用于训练。通过比较两套参数的输出来动态决定更新哪一套参数。
- D. 在训练开始时, 将原始模型权重复制一份作为可训练参数, 然后通过正则化手段强制让这些可训练参数与冻结的原始参数之间的差异矩阵保持低秩。

17. 一个 AI 团队在完成其大型模型的监督微调后, 准备进行偏好对齐。他们收集了大量成对的人类偏好数据, 每个提示词对应一个更受偏好的响应和一个不太理想的响应。团队在传统从人类反馈中强化学习 (Reinforcement Learning from Human Feedback, RLHF) 和新兴的直接偏好优化 (Direct Preference Optimization, DPO) 两种技术路线间进行抉择。如果他们选择直接偏好优化方案, 将意味着他们在流程中可以省去以下哪个关键步骤?

- A. 利用偏好数据对监督微调后的模型进行进一步的优化。
- B. 训练一个独立的模型来预测人类对不同回答的偏好程度, 并将其量化为奖励分数。
- C. 在优化过程中, 引入 KL 散度惩罚项以防止模型策略与初始监督微调后的模型偏离过远。
- D. 使用 GPU 集群进行模型参数的梯度更新。

18. 下列哪一项是混合专家模型 (Mixture of Experts, MoE) 面临的主要挑战?

- A. 模型训练过程极其不稳定, 难以收敛。
- B. 训练和推理速度相比同等参数量的稠密模型会变慢。
- C. 专家负载不均衡, 可能导致某些专家被过度使用, 而其他专家训练不足。
- D. 模型无法处理长序列输入, 存在上下文长度的限制。

19. 大模型推理加速工具 vLLM 之所以能够极大地提升大型语言模型的高吞吐量服务性能, 其最核心的架构创新在于什么?

- A. 采用更快的 GPU 计算库 (如 FlashAttention) 来优化注意力计算。

- B. 引入了名为 **PagedAttention** 的内存管理机制，有效解决了 **KV Cache** 的内存碎片化问题。
- C. 使用模型量化技术，降低模型权重占用的显存空间。
- D. 实现了更高效的调度算法，总是优先处理序列长度最短的请求。

20. 学习率预热是大模型训练中一项非常重要的技术。它的主要目的是什么？

- A. 在训练初期快速降低损失，以节省训练时间。
- B. 防止在训练初期，由于模型权重随机初始化且训练数据分布不稳定，过大的梯度更新导致模型不稳定或梯度爆炸。
- C. 为了让模型在训练初期有更强的探索能力，避免过早陷入局部最优。
- D. 主要是为了在预热阶段找到一个最优的初始学习率。

程序题（10 * 4 分 = 40 分）

一、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 = [21]
model = AutoModelForCausalLM.from_pretrained(
    model_name, device_map="auto", torch_dtype=torch.float16,
    trust_remote_code=True
)

# 2) 配置 LoRA
lora_config = LoraConfig(
    [22],
    target_modules=[23],
    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",
    [24],
    num_train_epochs=3,
    learning_rate=2e-4,
    [25],
    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")
```

21. 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)

22. 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,

23. 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"]

24. 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,

25. 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",

二、分组查询注意力

本题旨在考察你对分组查询注意力 (Grouped-Query Attention, GQA) 单步推理的理解与实现能力。题目提供了一个带有 5 个空缺的函数 `attn_step_gqa`，你需要从给定的选项中选择正确答案填入。

```
Python
import math
import torch
from typing import Tuple

def expand_kv_for_gqa(x: torch.Tensor, n_rep: int) ->
torch.Tensor:
    B, Kv, L, D = x.shape
    H = Kv * n_rep
    x = x.unsqueeze(1).repeat(1, n_rep, 1, 1, 1)  # (B, n_rep,
Kv, L, D)
    return x.reshape(B, H, L, D)

def attn_step_gqa(
    q: torch.Tensor,                # (B, H, 1, D)
    k_new: torch.Tensor,            # (B, Kv, 1, D)
    v_new: torch.Tensor,            # (B, Kv, 1, D)
    k_cache: torch.Tensor,          # (B, Kv, Lk, D)
    v_cache: torch.Tensor,          # (B, Kv, Lk, D)
    *, head_rep: int,              # H = Kv * head_rep
    causal_mask: torch.Tensor       # (1, 1, 1, Lk+1); 可见位置=1,
不可见=0
) -> Tuple[torch.Tensor, torch.Tensor, torch.Tensor]:
    # 1) 追加 KV 缓存 (在 Kv 头域)
    k_all = torch.cat([k_cache, k_new], dim=2)      # (B, Kv,
Lk+1, D)
    v_all = torch.cat([v_cache, v_new], dim=2)      # (B, Kv,
Lk+1, D)

    # 2) 将 KV 扩成 H 头, 以与 Q 对齐 (GQA/MQA 共享)
    kH = [26]
```

```

vH = expand_kv_for_gqa(v_all, head_rep)

# 3) 注意力打分:  $(B, H, 1, D) @ (B, H, D, L) \rightarrow (B, H, 1, L)$ 
scale = [27]
logits = [28]

# 4) 掩码与归一化
logits = [29]
attn = torch.softmax(logits, dim=-1)

# 5) 聚合得到输出
out = [30]

# 6) 返回输出与更新后的缓存 (供下一步继续用)
return out, k_all, v_all

def example_usage():
    """演示 GQA 的使用方法"""
    # 配置参数
    B = 2      # batch size
    H = 8      # 查询头数
    Kv = 2     # 键值头数 (GQA 中  $Kv < H$ )
    D = 64     # 头维度
    Lk = 10    # 缓存序列长度

    head_rep = H // Kv # 每个 KV 头的复制次数

    # 创建示例张量
    q = torch.randn(B, H, 1, D)
    k_new = torch.randn(B, Kv, 1, D)
    v_new = torch.randn(B, Kv, 1, D)
    k_cache = torch.randn(B, Kv, Lk, D)
    v_cache = torch.randn(B, Kv, Lk, D)

    # 创建因果掩码 (允许查看所有之前的位置)
    causal_mask = torch.ones(1, 1, 1, Lk + 1)

    # 执行 GQA 步骤
    out, k_all, v_all = attn_step_gqa(
        q, k_new, v_new, k_cache, v_cache,
        head_rep=head_rep,
        causal_mask=causal_mask
    )

```

```

print(f"输入形状:")
print(f"  q: {q.shape}")
print(f"  k_new: {k_new.shape}")
print(f"  v_new: {v_new.shape}")
print(f"  k_cache: {k_cache.shape}")
print(f"  v_cache: {v_cache.shape}")
print(f"\n 输出形状:")
print(f"  out: {out.shape}")
print(f"  k_all (updated): {k_all.shape}")
print(f"  v_all (updated): {v_all.shape}")

if __name__ == "__main__":
    example_usage()

```

26. 在 GQA 机制中，键值头数少于查询头数，需要将 KV 张量扩展以匹配 Q 的头数，以下代码更合理的是：

- A. `k_all.view(B, head_rep, Kv, Lk+1, D)`
- B. `expand_kv_for_gqa(q, head_rep)`
- C. `expand_kv_for_gqa(k_all, head_rep)`
- D. `k_all.transpose(1, 2)`

27. 在 Scaled Dot-Product Attention 中，需要对 QK^T 的结果进行缩放以稳定梯度和数值范围，以下代码更合理的是：

- A. `1.0 / math.sqrt(H)`
- B. `1.0 / math.sqrt(D)`
- C. `1.0 / D`
- D. `1.0 / H`

28. 此处需要计算查询与键的点积，得到注意力分数 (logits)，以下代码更合理的是：

- A. `torch.matmul(q, kH.transpose(-2, -1)) * scale`
- B. `torch.matmul(q, kH) * scale`
- C. `torch.matmul(kH, q) * scale`
- D. `torch.einsum('bh1d,bhld->bh1d', q, kH) * scale`

29. 在自回归生成任务中，需要应用掩码来防止模型在计算注意力时“看到”未来的内容，以下代码更合理的是：

- A. `logits * (causal_mask == 0)`
- B. `logits * causal_mask`
- C. `logits.masked_fill(causal_mask == 0, float("-inf"))`
- D. `torch.where(causal_mask == 0, logits, float("-inf"))`

30. 在 Attention 机制的最后一步，需要使用注意力权重对值向量进行加权求和，得到最终输出，以下代码更合理的是：

- A. `torch.matmul(attn, vH)`
- B. `torch.matmul(vH, attn)`
- C. `torch.einsum('bh1l,bhld->bhld', attn, vH)`
- D. `torch.einsum(attn, vH, 'bh1l,bhld->bhld')`

答案

单选题：

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

程序题：

- 21-25 DCDDDB
- 26-30 CBACA