LMCC-2025 第一轮试题解析

青少年组

单选题

1. 关于模型泛化能力的理论保证，以下哪项描述最准确？

A. 训练误差越小泛化能力一定越强

B. 模型复杂度与泛化能力呈正比关系

C. 充足的数据量与合适的模型复杂度有助于提升泛化能力

D. 测试集精度完全代表模型在实际应用中的表现

答案：C

解析：泛化能力取决于多个因素的平衡。训练误差小可能导致过拟合（A错误）；模型复杂度过高反而会降低泛化能力（B错误）；测试集只是对泛化能力的一个估计，不能完全代表实际表现（D错误）。只有C选项准确地指出了数据量和模型复杂度之间的平衡关系。

2. 在以下场景中，哪项最适合应用提示学习方法？

A. 需要从零开始训练一个全新的深度学习模型

B. 希望利用现有预训练模型完成特定文本分类任务

C. 处理大规模结构化数据的统计分析

D. 进行复杂的数学公式推导计算

答案：B

解析：提示学习（Prompt Learning）的核心是利用预训练模型的知识，通过设计合适的提示来引导模型完成特定任务，无需大量标注数据和重新训练。因此最适合B选项的场景。A选项需要从零训练，C和D选项不是提示学习的典型应用场景。

3. 在筛选高质量指令数据时，以下哪项标准最能有效识别低质量数据？

A. 指令长度超过20个字符

B. 回答内容与指令语义不匹配

C. 使用英文书写的指令

D. 指令中包含数字符号

答案：B

解析：高质量指令数据的核心要求是指令与回答之间的语义一致性和相关性。指令长度、语言类型、是否包含数字符号都不是判断数据质量的关键标准。只有B选项指出的"回答内容与指令语义不匹配"才是真正的低质量数据特征。

4. 关于大语言模型的代表性模型，以下哪项描述最准确？

A. GPT、Llama 等基于 Transformer 的预训练模型通常被视为大语言模型

B. Transformer 架构模型参数量必然很小，不可能成为大语言模型

C. 卷积神经网络是大语言模型的唯一形式

D. 线性回归模型由于结构简单，是大语言模型的典型代表

答案：A

解析：大语言模型主要基于Transformer架构，GPT、Llama 系列是其代表性模型。Transformer架构恰恰能够支持大规模参数（B错误）；卷积神经网络主要用于图像处理（C错误）；线性回归是传统统计模型，不属于大语言模型（D错误）。

5. 关于模型评测中的公平性问题，以下哪项描述最准确？

A. 公平性仅涉及算法设计的数学公平

B. 评测数据集的选择不会影响公平性评估

C. 公平性要求模型在不同群体间表现一致

D. 公平性问题只存在于监督学习模型中

答案：C

解析：模型公平性的核心是确保模型对不同群体（如不同性别、种族、年龄等）提供一致的服务质量，避免歧视。公平性不仅是数学问题，还涉及社会伦理（A错误）；数据集选择会显著影响公平性（B错误）；所有类型的机器学习模型都可能存在公平性问题（D错误）。

6. 关于基于 Transformer 的三种主流模型架构（Encoder-only、Decoder-only、Encoder–Decoder），以下描述正确的是：

A. Encoder–Decoder 架构仅适用于机器翻译任务

B. Decoder-only 架构通过因果掩码使每个位置只能关注自身及之前的内容

C. Encoder-only 架构不包含自注意力机制

D. 三种架构的参数量在相同任务下始终一致

答案：B

解析：Decoder-only架构（如GPT）使用因果掩码确保自回归生成，每个位置只能看到之前的信息，这是其核心特征。Encoder-Decoder架构可用于多种序列到序列任务（A错误）；Encoder-only架构（如BERT）的核心就是自注意力机制（C错误）；不同架构的参数量差异很大（D错误）。

7. 关于批量大小（Batch Size）对模型训练的影响，以下说法错误的是：

A. 较大的批量大小可以提高训练的并行效率

B. 较小的批量大小通常使梯度估计更加准确

C. 批量大小的选择需要在训练速度和模型性能之间权衡

D. 过大的批量大小可能导致模型泛化能力下降

答案：B

解析：从优化理论的角度，单次mini-batch梯度估计的期望等于真实梯度，与batch size无关。较大的批量大小能提供方差更小的梯度估计（因为平均了更多样本的梯度，统计上更稳定），而较小的批量大小梯度噪声更大（方差大）。小batch的噪声有时反而有助于逃出局部最优、提升泛化能力。因此B选项中"更加准确"这一说法不严谨，是错误的。其他选项都正确：大批量提高并行效率（A）、需要权衡（C）、过大批量可能影响泛化（D）。

8. 指令微调（Instruction Tuning）的核心目标是：

A. 从零开始训练一个全新的语言模型

B. 提高模型理解和遵循人类指令的能力

C. 减少模型的参数量以降低推理成本

D. 消除预训练数据中的所有偏见

答案：B

解析：指令微调是在预训练模型基础上进行的，通过使用指令-响应数据对进行训练，使模型更好地理解和执行人类指令。它不是从零训练（A错误）、不直接减少参数量（C错误）、也无法完全消除偏见（D错误）。

9. 在大语言模型的训练中，监督学习（Supervised Learning）的主要特征是什么？

A. 使用未标记数据进行训练

B. 使用标记数据进行训练

C. 不需要训练数据

D. 仅使用测试数据进行学习

答案：B

解析：监督学习的核心特征是使用带有标签的数据进行训练，即每个输入都有对应的正确输出（标记）。未标记数据用于无监督学习（A错误）；任何机器学习都需要训练数据（C错误）；测试数据用于评估而非训练（D错误）。

10. 关于 DeepSeek-R1 的特点，以下哪项描述最准确？

A. DeepSeek-R1 专注于强化推理能力，结合强化学习与思维链生成策略

B. DeepSeek-R1 是 2010 年发布的图像分类模型，仅支持离线部署

C. DeepSeek-R1 完全不开源，无法进行任何微调或评估

D. DeepSeek-R1 仅用于传统统计语言建模，未引入自注意力机制

答案：A

解析：DeepSeek-R1是一个现代大语言模型，其特点是通过强化学习增强推理能力，并采用思维链（Chain-of-Thought）策略来提升复杂问题的解决能力。其他选项的时间、定位、开源性和技术架构描述都不正确。

11. 以下哪一项是精确率（Precision）的计算公式？ 注：TP（True Positive，真阳性）、FP（False Positive，假阳性）、TN（True Negative，真阴性）、FN（False Negative，假阴性）；

A. TP / (TP + FP)

B. TP / (TP + FN)

C. (TP + TN) / (TP + TN + FP + FN)

D. (TP + FP) / (TP + TN + FP + FN)

答案：A

解析：精确率（Precision）衡量的是"预测为正例中真正是正例的比例"，即在所有预测为正的样本中，实际为正的比例。公式为 TP/(TP+FP)。B选项是召回率（Recall），C选项是准确率（Accuracy）。

12. 在预训练技术中，下一个词元预测任务（Next Token Prediction）主要基于以下哪种核心思想？

A. 根据前文词元序列预测下一个可能出现的词元

B. 随机遮盖输入序列中的部分词元并进行重构

C. 同时预测输入序列中所有位置的词元

D. 将输入序列转换为图像特征进行识别

答案：A

解析：下一个词元预测是自回归语言模型（如GPT）的核心训练任务，模型根据已有的前文序列预测下一个词元。B选项描述的是掩码语言模型（如BERT）的训练方式。C和D选项都不是标准的语言模型训练方式。

13. 关于自注意力机制，以下哪项描述是正确的？

A. 只能处理序列中的单个元素

B. 查询（Query）、键（Key）、值（Value）通常来自同一输入序列

C. 不需要计算注意力权重

D. 只能用于图像处理任务

答案：B

解析：自注意力机制的"自"体现在Query、Key、Value都来自同一输入序列，通过计算序列内部元素之间的关系来获取上下文信息。自注意力可以处理整个序列（A错误）、必须计算注意力权重（C错误）、主要用于NLP任务（D错误）。

14. 关于大语言模型的优势与局限性，以下描述正确的是？

A. 大语言模型在创造性任务上表现优异，但可能存在事实性错误

B. 大语言模型完全不会产生偏见内容

C. 大语言模型在所有任务上都优于人类专家

D. 大语言模型不需要大量训练数据即可达到最佳效果

答案：A

解析：大语言模型在文本生成、创造性写作等方面表现出色，但可能产生"幻觉"（hallucination），即生成看似合理但实际错误的内容。模型会继承训练数据中的偏见（B错误）、在某些专业领域不如人

类专家（C错误）、需要海量数据训练（D错误）。

15. 关于大语言模型的训练流程，以下哪项分阶段描述最准确？

A. 数据收集与预处理 → 大规模预训练 → 指令微调与对齐

B. 在线推理 → 实时部署 → 用户体验评估

C. 需求分析 → 产品设计 → 市场推广

D. 只进行一次预训练即可，无需其他阶段

答案：A

解析：大语言模型的典型训练流程分为三个主要阶段：首先收集和预处理大规模文本数据，然后进行预训练学习通用语言表示，最后通过指令微调和人类对齐使模型更好地服从人类意图。B选项描述的是部署阶段，C选项是产品开发流程，D选项过于简化。

16. 除了无害性、有用性和诚实性，人类对齐还可能包含哪些重要标准？

A. 语言表达的准确性和文化适应性

B. 符合特定社会背景下的道德规范

C. 尊重用户隐私和数据安全

D. 其它所有选项

答案：D

解析：人类对齐是一个多维度的概念，除了基本的无害性、有用性和诚实性（3H原则），还包括语言和文化适应性、道德规范、隐私保护等多个方面。不同的应用场景和文化背景可能对对齐标准有不同要求，因此A、B、C都是重要的对齐标准。

17. 下列哪个项是大语言模型产生偏见的主要来源？

A. 训练数据中特定群体的样本数量不足

B. 模型算法的计算速度过快

C. 硬件设备的存储容量限制

D. 用户界面的设计不够美观

答案：A

解析：模型偏见主要源于训练数据的不平衡和偏见。如果训练数据中某些群体的样本不足或存在刻板印象，模型会学习并放大这些偏见。计算速度（B）、硬件容量（C）和界面设计（D）都与模型偏见的产生无直接关系。

18. 按照语言模型的发展历程，以下四代模型的正确时间顺序是：

A. 统计语言模型 → 预训练语言模型 → 神经网络语言模型 → 大语言模型

B. 统计语言模型 → 神经网络语言模型 → 预训练语言模型 → 大语言模型

C. 神经网络语言模型 → 统计语言模型 → 预训练语言模型 → 大语言模型

D. 预训练语言模型 → 统计语言模型 → 神经网络语言模型 → 大语言模型

答案：B

解析：语言模型的发展经历了四个主要阶段：最早的统计语言模型（如N-gram）→ 基于神经网络的语言模型（如RNN、LSTM）→ 预训练语言模型（如BERT、GPT）→ 大规模预训练的大语言模型（如GPT-3/4、ChatGPT）。这个发展顺序反映了技术的逐步演进。

19. 关于贪心搜索（Greedy Search）和束搜索（Beam Search）这两种解码策略，以下描述正确的是：

A. 贪心搜索在每一步选择概率最高的词元，计算效率高但可能错过全局最优解

B. 束搜索的束宽（Beam Width）越大，生成结果一定越好

C. 贪心搜索等同于束宽为0的束搜索

D. 束搜索的计算复杂度与贪心搜索完全相同

答案：A

解析：贪心搜索每步选择最优词元，速度快但可能陷入局部最优。束搜索保留多个候选路径来寻找更好的全局解。束宽越大计算成本越高，且不保证结果更好（B错误）；贪心搜索相当于束宽为1的束搜索（C错误）；束搜索的计算复杂度远高于贪心搜索（D错误）。

20. 关于过拟合（Overfitting）现象，以下描述正确的是？

A. 模型在训练集和测试集上表现都很好

B. 模型过于简单，无法捕捉数据特征

C. 模型在训练集上表现很好，但在测试集上表现差

D. 模型参数数量过少导致性能下降

答案：C

解析：过拟合是指模型过度学习了训练数据的细节和噪声，导致在训练集上表现优异，但在未见过的测试集上泛化能力差。A选项描述的是理想情况，B和D选项描述的是欠拟合（Underfitting）现象。

编程题 1

Qwen2.5-VL 是多模态大语言模型（Vision-Language Model, VLM），能够同时处理图像与文本输入，实现图文问答与视觉描述等任务。以下代码演示了使用 Qwen2.5-VL-7B-Instruct 模型，对输入图像进行分析并生成文本回答。请阅读以下代码，并根据描述完成空缺部分。

代码块

```
import torch
from transformers import Qwen2_5_VLForConditionalGeneration, AutoProcessor
from PIL import Image

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

# 1) 加载处理器与模型
processor = ____[1]____
model = Qwen2_5_VLForConditionalGeneration.from_pretrained(
    model_name,
    device_map="auto",
    torch_dtype=torch.bfloat16,
```

```
    trust_remote_code=True
)

# 2) 读取输入图像
image = ____[2]____

# 3) 构建多模态输入提示
messages = [
    {
        "role": "user",
        "content": [
            {"type": "image", "image": image},
            {"type": "text", "text": "请描述图片中的主要内容。"}
        ]
    }
]
text = ____[3]____
inputs = processor(text=[text], images=[image], return_tensors="pt").to(model.device)

# 4) 模型推理生成回答
model = model.eval()
with ____[4]____:
    output_ids = model.generate(
        **inputs,
        max_new_tokens=128,
        temperature=0.7,
    )
    generated_ids = ____[5]____
    answer = processor.tokenizer.batch_decode(generated_ids, skip_special_tokens=True)[0]
    print("模型回答：", answer)
```

[1] Qwen2.5-VL 模型依赖自定义多模态处理逻辑，需启用 remote code。请选择一种能正确加载该模型处理器的方式：

A. `AutoProcessor.from_pretrained(model_name)`

B. `AutoProcessor.from_pretrained(model_name, trust_remote_code=True)`

C. `AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)`

D. `Qwen2_5_VLProcessor.from_pretrained(model_name)`

答案：B

解析：Qwen2.5-VL 是多模态模型，需要使用 `AutoProcessor` 来同时处理图像和文本（C选项的 `AutoTokenizer` 只能处理文本）。由于模型使用了自定义的处理逻辑，必须设置 `trust_remote_code=True` 来允许加载远程代码（A选项缺少此参数会失败）。D选项的专用处理器类名不正确。

[2] 输入图像应以 PIL.Image 对象的 RGB 格式提供给 processor。请选择正确的实现方式：

A. `Image.open("image.jpg").convert("RGB")`

B. `cv2.cvtColor(cv2.imread("image.jpg"), cv2.COLOR_BGR2RGB)`

C. `np.array(Image.open("image.jpg").convert("RGB"))`

D. `transforms.ToTensor()(Image.open("image.jpg").convert("RGB"))`

答案：A

解析：Qwen2.5-VL 的 processor 期望接收 PIL.Image 对象。A选项直接返回 PIL.Image 对象并转换为 RGB 格式，是最直接正确的方式。B选项返回 numpy 数组而非 PIL.Image 对象；C选项同样转换为了 numpy 数组；D选项返回的是 PyTorch 张量。虽然 processor 内部可能会处理这些格式，但直接使用 PIL.Image 对象是最标准的做法。

[3] 对于对话式多模态模型，需要使用 chat template 格式化输入。请选择正确的实现方式：

A. `processor.format_messages(messages)`

B. `processor.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)`

C. `processor.tokenize(messages, add_special_tokens=True)`

D. `json.dumps(messages)`

答案：B

解析：`apply_chat_template` 是 Hugging Face transformers 库中用于将对话消息格式化为模型所需格式的标准方法。参数 `tokenize=False` 表示只返回格式化后的文本而不进行分词，`add_generation_prompt=True` 会添加生成提示符以引导模型开始回答。A选项的方法不存在；C选项是分词方法，不处理对话格式；D选项只是简单的JSON序列化，不会应用模型特定的对话模板。

[4] 推理时应使用专用上下文管理器以禁用梯度计算并优化性能。请选择最适合推理的方式：

A. `torch.enable_grad(False)`

B. `torch.set_grad_enabled(False)`

C. `torch.autograd.set_detect_anomaly(False)`

D. `torch.inference_mode()`

答案：D

解析：`torch.inference_mode()` 是 PyTorch 推荐用于推理的上下文管理器，它不仅禁用梯度计算，还会禁用自动求导引擎以获得更好的性能。相比 `torch.no_grad()`，它的性能优化更彻底。A选项不是有效的PyTorch API；B选项虽然可以禁用梯度，但不是上下文管理器且性能优化不如 `inference_mode()`；C选项用于调试异常检测，与推理优化无关。

[5] 模型输出包含输入和生成部分，需要提取仅新生成的 token。请选择正确的实现方式：

A. `output_ids[inputs['input_ids'].shape[0]:]`

B. `output_ids.squeeze()`

C. `output_ids[:, inputs['input_ids'].shape[1]:]`

D. `output_ids[:, -128:]`

答案：C

解析：生成式模型的输出包含完整的序列（输入 + 生成的内容）。`output_ids` 的形状是 `[batch_size, total_length]`，我们需要从第二个维度（序列长度维度）切片，去掉输入部分。`inputs['input_ids'].shape[1]` 获取输入序列的长度，`output_ids[:, inputs['input_ids'].shape[1]:]` 从该位置开始切片，得到新生成的 token。A选项在错误的维度切片；B选项只是去除维度，不提取新token；D选项假设固定生成128个token，不够灵活且可能不准确。

编程题 2

在大语言模型的实际部署中，推理阶段通常需要在保证**稳定性、时延与显存效率**之间取得平衡。常见策略包括：

- 将模型切换至推理模式以禁用 Dropout；
- 在推理时关闭梯度计算以减少显存占用；
- 通过温度采样（Temperature Sampling）或核采样（Top-p Sampling）控制生成的多样性。

以下示例基于 **Qwen2.5-7B** 模型，实现了推理函数 `deploy_generate`，演示如何完成多轮增量文本生成与结果解码。请阅读以下代码，并在 [1] 至 [5] 处选择合适选项填入。

代码块

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# === 模型路径 ===
model_name = "Qwen/Qwen2.5-7B"

# === 准备模型与分词器 ===
tokenizer = ____[1]____
model = ____[2]____

# === 推理函数（deploy_generate） ===
def deploy_generate(model, tokenizer, prompts, max_new_tokens=64,
temperature=0.8, eos_id=None, device="cuda"):
    # 0) 模型切换为推理模式
    model = ____[3]____

    # 1) Tokenize 输入
    inputs = tokenizer(prompts, return_tensors="pt", padding=True)
    tokens = inputs["input_ids"].to(device)
    batch_size = tokens.size(0)

    # 2) 预热：完整提示构建 KV 缓存并得到首个 logits
    with torch.inference_mode():
        prime_logits = model(tokens).logits[:, -1, :]  # 取最后一步的 logits
    start_pos = tokens.size(1)

    generated = []
    finished = torch.zeros(batch_size, dtype=torch.bool, device=device)

    # 3) 增量生成循环
    with torch.inference_mode():
        logits = prime_logits
        for step in range(max_new_tokens):
```

```
            # 温度缩放与 softmax
            probs = ____[4]____
            next_token = torch.multinomial(probs, num_samples=1)  # 采样一个 token

            # 拼接生成序列
            tokens = torch.cat([tokens, next_token], dim=1)
            generated.append(next_token)

            # 判断是否结束
            if eos_id is not None:
                finished |= (next_token.squeeze(1) == eos_id)
                if torch.all(finished):
                    break

            # 增量步（此处使用简化形式重新计算 logits）
            logits = model(tokens).logits[:, -1, :]
            start_pos += 1

    # 4) 拼接新增 tokens 并解码
    if len(generated) > 0:
        new_tokens = torch.cat(generated, dim=1)
    else:
        new_tokens = torch.empty(batch_size, 0, dtype=torch.long, device=device)

    texts = []
    for i in range(batch_size):
        texts.append(____[5]____)
    return texts

if __name__ == "__main__":
    prompts = [
        "介绍一下西安的特色美食。",
        "请写一首关于秋天的短诗。"
    ]
    results = deploy_generate(model, tokenizer, prompts, max_new_tokens=50)
    for i, res in enumerate(results):
        print(f"\nPrompt {i+1}: {prompts[i]}")
        print("Model output:", res)
```

[1] Qwen2.5 模型需要加载支持自定义代码的分词器。请选择正确的加载方式：

A. `AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)`

B. `AutoTokenizer.load(model_name)`

C. `AutoProcessor.from_pretrained(model_name)`

D. `AutoTokenizer(model_name)`

答案：A

解析：Qwen2.5 模型使用了自定义的分词逻辑，必须通过 `AutoTokenizer.from_pretrained()` 方法加载，并设置 `trust_remote_code=True` 以允许执行远程代码。B选项的 `load()` 方法不存在；C选项的 `AutoProcessor` 用于多模态模型；D选项直接实例化类的方式不正确。

[2] 为优化推理效率，需要自动分配 GPU 设备并使用半精度格式以节省显存。请选择正确的模型加载方式：

A. `AutoModelForCausalLM.from_pretrained(model_name, device_map="auto", trust_remote_code=True)`

B. `AutoModelForSeq2SeqLM.from_pretrained(model_name, device_map="auto", torch_dtype=torch.float16, trust_remote_code=True)`

C. `AutoModelForCausalLM.load(model_name, device_map="auto", torch_dtype=torch.float16, trust_remote_code=True)`

D. `AutoModelForCausalLM.from_pretrained(model_name, device_map="auto", torch_dtype=torch.float16, trust_remote_code=True)`

答案：D

解析：Qwen2.5 是因果语言模型（自回归生成），需要使用 `AutoModelForCausalLM`（B选项的 `AutoModelForSeq2SeqLM` 用于编码器-解码器架构如T5）。完整配置包括：`device_map="auto"` 自动分配GPU设备，`torch_dtype=torch.float16` 使用半精度节省显存，`trust_remote_code=True` 允许自定义代码。C选项的 `load()` 方法不存在；A选项缺少半精度配置。

[3] 推理前需要将模型移至目标设备并切换到评估模式以禁用 Dropout 等训练行为。请选择正确的实现方式：

A. `model.train()`

B. `model.to(device).eval()`

C. `model.cuda()`

D. `model.bfloat16()`

答案：B

解析：推理时需要两步操作：`model.to(device)` 将模型移至指定设备（如GPU），`model.eval()` 切换到评估模式，禁用Dropout、BatchNorm等训练特有的行为。这两个操作可以链式调用。A选项的 `train()` 是训练模式，与需求相反；C选项只移动到CUDA设备但未切换评估模式；D选项是数据类型转换，不是模式切换。

[4] 需要对 logits 应用温度缩放并通过 softmax 转换为概率分布以控制生成多样性。请选择正确的实现方式：

A. `torch.softmax(logits / temperature, dim=-1)`

B. `torch.softmax(logits * temperature, dim=0)`

C. `torch.sigmoid(logits)`

D. `torch.softmax(logits, dim=-2)`

答案：A

解析：温度采样通过 `logits / temperature` 进行缩放来控制生成的多样性：temperature > 1 使分布更平滑（更随机），temperature < 1 使分布更尖锐（更确定）。然后在最后一个维度（词表维度）上应用 softmax 得到概率分布。B选项使用乘法且维度错误；C选项的 sigmoid 用于二分类，不适合多分类；D选项的维度-2不是词表维度。

[5] 生成完成后需要将新增的 token 序列解码为可读文本。请选择正确的解码方式：

A. `tokenizer.decode(new_tokens[i])`

B. `tokenizer.batch_decode(new_tokens, skip_special_tokens=True)`

C. `tokenizer.decode(new_tokens[i].tolist(), skip_special_tokens=True)`

D. `tokenizer.decode(tokens, skip_special_tokens=False)`

答案：C

解析：在循环中逐个处理每个样本（通过索引 `i`），需要使用 `tokenizer.decode()` 而非批量解码。输入必须是Python列表或一维张量，因此需要 `.tolist()` 转换；`skip_special_tokens=True` 移除特殊token（如 `<pad>`，`</s>` 等）使输出更清晰。A选项缺少 `tolist()` 和参数；B选项是批量解码但代码在循环中；D选项解码了完整序列（包括输入）且保留特殊token。

成人组

单选题

1-10 题同青少年组 11-20 题

11. 直接偏好优化（Direct Preference Optimization, DPO）的核心思想是什么？

A. 通过强化学习从人类反馈中直接优化策略

B. 将偏好学习问题转化为二元分类任务进行优化

C. 使用隐式奖励建模，避免显式的奖励函数学习

D. 通过策略概率比的对数构建损失函数，直接优化偏好数据

答案：D

解析：DPO是一种新型的对齐方法，其核心创新在于跳过传统RLHF中的奖励模型训练阶段，直接利用偏好数据优化策略模型。它通过构建基于策略概率比对数的损失函数，将偏好优化问题转化为监督学习问题，既保持了RLHF的效果，又大大简化了训练流程。在四个选项中，D最准确地描述了DPO的技术实现（基于log策略比构建显式损失函数）。选项C虽然在直觉上也反映了DPO"隐式奖励"的一个侧面（将奖励隐式编码在损失函数中而非训练独立的reward model），但这一特征也适用于其他类似方法（如IPO、KTO、ORPO等），且C没有体现DPO核心的"策略概率比对数损失"这一具体形式，因此不如D准确和具有区分度。

12. 关于前缀微调（Prefix Tuning）与提示微调（Prompt Tuning）的异同点，以下哪种说法最准确？

A. 两者都只在输入层添加可训练参数，不修改模型内部结构

B. 前缀微调在每一层都添加可训练前缀，而提示微调仅在输入层添加提示

C. 提示微调通过梯度更新优化提示向量，前缀微调仅使用人工设计的模板

D. 前缀微调需要修改模型架构，提示微调完全不需要训练任何参数

答案：B

解析：Prefix Tuning和Prompt Tuning都是参数高效微调方法，但机制不同。Prefix Tuning在Transformer的每一层都添加可训练的前缀向量，影响所有层的表示；而Prompt Tuning只在输入层添加软提示（soft prompt）。两者都通过梯度更新优化（C错误），都不需要修改模型架构且都需要训练参数（D错误）。

13. 关于AdamW优化器的改进，以下说法正确的是？

A. 移除了权重衰减与梯度更新的耦合关系

B. 增加了动量因子的计算复杂度

C. 取消了学习率自适应机制

D. 仅适用于计算机视觉任务

答案：A

解析：AdamW是对Adam优化器的改进，其核心改进是将权重衰减（weight decay）从梯度更新中解耦出来，直接作用于参数本身，这使得正则化效果更加稳定有效。AdamW保留了Adam的自适应学习率机制（C错误），计算复杂度没有明显增加（B错误），且广泛应用于各类深度学习任务（D错误）。

14. 在深度学习模型训练过程中，梯度裁剪（Gradient Clipping）技术的主要作用是什么？

A. 加速模型收敛速度，提高训练效率

B. 防止梯度爆炸，维持训练稳定性

C. 减少模型参数量，降低计算复杂度

D. 增强模型泛化能力，防止过拟合

答案：B

解析：梯度裁剪通过限制梯度的范数或值，防止在训练过程中出现梯度爆炸问题，这在训练循环神经网络和深层网络时尤为重要。它不直接加速收敛（A错误）、不减少参数量（C错误）、主要目的也不是防止过拟合（D错误），而是确保训练过程的数值稳定性。

15. 关于知识利用能力评测中时效性要求的理解，以下哪项描述最准确？

A. 仅开卷问答需要考虑知识时效性

B. 所有任务类型都需要考虑知识时效性

C. 时效性只影响知识检索任务

D. 时效性仅与知识更新机制相关

答案：B

解析：知识时效性是评测大语言模型的重要维度，因为现实世界的知识不断更新变化。无论是闭卷问答、开卷问答、知识检索还是推理任务，都需要考虑知识的时效性，以确保模型能够提供准确且最新的信息。时效性不仅影响单一任务类型（A、C错误），也不仅限于更新机制（D错误）。

16. 在评估大语言模型的逐步推理能力时，以下哪种情况最能体现模型的真实推理水平？

A. 模型直接给出最终答案，不展示中间步骤

B. 模型生成的推理步骤逻辑连贯且可验证

C. 模型引用大量外部知识但不解释推理过程

D. 模型重复问题描述后直接给出结论

答案：B

解析：逐步推理能力的核心在于模型能否展示清晰、连贯、可验证的推理过程，这类似于人类解决问题时的思考步骤。只有当推理步骤逻辑严密且每一步都可以被验证时，才能真正体现模型的推理能力。直接给答案（A）、不解释过程（C）或简单重复（D）都无法有效评估推理水平。

17. 已知标准注意力的计算复杂度为 $O(L^2d)$ ，其中序列长度为 L ，特征维度为 d 。设 $Q,K,V \in \mathbb{R}^{L\times d}$ ，某研究提出使用 $\phi(Q)\phi(K)^\top$ 近似 $QK^\top$ ，其中 $\phi:\mathbb{R}^d\to\mathbb{R}^d$ ，计算复杂度为 $O(d)$ 。通过调整计算顺序为 $\phi(Q)[\phi(K)^\top V]$ ，则该算法的总体时间复杂度为：

A. $O(Ld+d^2)$

B. $O(Ld^2)$

C. $O(L^2+Ld)$

D. $O(Ld)$

答案：B

解析：这是一类借助核函数分解实现的、相对于序列长度线性的注意力近似方法的复杂度分析。计算过程分解为：(1) 计算 $\phi(Q)$ 和 $\phi(K)$ ： $O(Ld)$ ；(2) 计算 $\phi(K)^\top V$ ： $(d\times L)\times(L\times d)=O(Ld^2)$ ；(3) 计算 $\phi(Q)\times[\phi(K)^\top V]$ ： $(L\times d)\times(d\times d)=O(Ld^2)$ 。总体复杂度为 $O(Ld^2)$ 。相比标准注意力的 $O(L^2d)$ ，该方法消除了对序列长度的二次依赖，在长序列（ $L\gg d$ ）时有显著优势。注意：在实际的线性注意力实现中，通常会将 ϕ 映射到更低维度 $m\ll d$，从而进一步将复杂度降低至 $O(Ldm)$ 。

18. 在深度学习模型训练中，混合精度训练（Mixed Precision Training）技术的主要作用是什么？

A. 通过使用单精度浮点（float32）主副本维护参数精度，结合半精度浮点（float16）计算提升训练速度，并采用损失缩放避免梯度下溢

B. 完全消除半精度浮点的数值精度损失，使其与单精度浮点等效

C. 将半精度浮点数据转换为整型进行计算，以进一步加速训练

D. 通过增加模型参数量来补偿半精度浮点带来的精度损失

答案：A

解析：混合精度训练是一种平衡训练速度和数值精度的技术。它用float16进行前向和反向传播计算以加速训练，同时保留float32的主参数副本以维持精度；并使用损失缩放（loss scaling）技术防止小梯度下溢。这种方法不能完全消除精度损失（B错误），不涉及整型计算（C错误），也不增加参数量（D错误）。

19. 在提示学习中，以下哪项不属于人工提示设计的基本原则？

A. 提示应清晰明确，避免歧义

B. 提示应包含足够上下文信息

C. 提示应尽可能复杂以测试模型极限

D. 提示应考虑目标模型的特性

答案：C

解析：好的提示设计应该遵循清晰性、信息充分性和适配性原则。提示应该简洁明确而非复杂化，过于复杂的提示反而可能混淆模型、降低性能。有效的提示设计需要：避免歧义（A）、提供充足上下文（B）、根据模型特点调整（D），而不是刻意增加复杂度。

20. 某预训练任务初始批次大小（Batch Size）为256，采用梯度累积（Gradient Accumulation）策略每4步更新一次，实际等效批次大小是多少？若改为动态调整策略，在训练中期将批次大小增至512，请计算梯度累积步数应如何调整以保持等效批次大小不变？

A. 等效1024，累积步数改为2步

B. 等效512，累积步数改为2步

C. 等效1024，累积步数改为8步

D. 等效512，累积步数改为8步

答案：A

解析：等效批次大小 = 单次批次大小 × 梯度累积步数。初始情况：256 × 4 = 1024。当批次大小增至512时，为保持等效批次大小1024不变，需要调整累积步数：1024 ÷ 512 = 2步。梯度累积是一种在显存受限时模拟大批次训练的技术，通过累积多个小批次的梯度后再更新参数。

编程题 1

同青少年组 编程题 2

编程题 2

多头注意力（Multi-Head Attention, MHA） 是 Transformer 架构的核心组件，在大语言模型中扮演着至关重要的角色。MHA 通过并行计算多个注意力头，能够让模型同时关注序列中不同位置的不同表示子空间，从而捕获更丰富的语义信息和位置关系。在标准 MHA 中，每个注意力头都维护独立的 Key (K) 和 Value (V) 缓存（KV Cache），这在推理阶段会带来显著的显存开销。对于一个 H 头的注意力层，KV Cache 的显存占用为 O(B × L × H × d)，其中 B 是批次大小，L 是序列长度，d 是每个头的维度。随着模型规模增大和上下文长度增加，KV Cache 成为部署大模型的主要显存瓶颈。

多头潜在注意力（Multi-Head Latent Attention, MLA） 是 DeepSeek-V2/V3 等模型提出的创新机制，通过**低秩分解**显著优化 KV Cache 的显存占用。MLA 将所有注意力头的 KV 表示投影到一个**共享的低秩潜在空间（Compressed Key-Value dimension）**（维度 Ckv），而不是为每个头存储独立的高维 KV，将显存占用降低至 O(B × L × Ckv)，其中 Ckv << H × d，实现数倍显存节省。通过**头依赖的权重矩阵**在查询时动态恢复各头的独立表示，MLA 在保持模型表达能力的同时大幅降低了显存开销。

MLA 的核心创新在于**解耦内容匹配与位置建模**。查询向量被分为两个独立部分：一部分处理语义内容匹配，需要先通过头依赖的权重矩阵投影到共享的低秩空间再与缓存交互；另一部分处理位置关系，直接与位置缓存进行相关性计算。两路径的相关性分数需要合理合并并进行适当缩放，同时自回归解码需要确保模型只能看到当前及之前的位置，未来位置必须被屏蔽，最终通过归一化操作得到注意力分布。注意力加权后的结果位于低秩潜在空间中，需要通过头依赖的权重矩阵将其映射回各头的值空间，恢复多头的独立表示能力。

以下代码实现了 MLA 的单步解码函数及演示用法。请阅读以下代码，并根据描述完成空缺部分。

代码块

```python
import torch
import math

def mla_decode_step(
    q_nope: torch.Tensor,    # (B,H,Q,Dn)
    q_pe: torch.Tensor,      # (B,H,Q,Dr)
    kv_cache: torch.Tensor, # (B,Lk,Ckv)
    pe_cache: torch.Tensor, # (B,Lk,Dr)
    wkv_b: torch.Tensor,     # (H, Dn+V, Ckv)
    softmax_scale: float,
    causal_mask: torch.Tensor  # (1,1,1,Lk); 可见=1, 不可见=0
):
```

```
    # === 维度准备与权重切分 ===
    Dn = q_nope.size(-1)
    Vd = wkv_b.size(1) - Dn
    k_nope_w = wkv_b[:, :Dn, :]     # (H, Dn, Ckv)
    v_w      = wkv_b[:, -Vd:, :]    # (H,  Vd, Ckv)

    # 1) 将 q_nope 投影到 KV 低秩空间
    q_nope_proj = ____[1]____       # -> (B,H,Q,Ckv)

    # 2) 用 kv_cache 做相关得到 scores_nope
    scores_nope = ____[2]____       # -> (B,H,Q,Lk)

    # 3) 合并两部分分数并缩放
    scores_pe   = torch.einsum("bhqr,btr->bhqt", q_pe, pe_cache)
    logits = ____[3]____

    # 4) 应用因果掩码（不可见位置置为 -inf）
    logits = ____[4]____
    attn = torch.softmax(logits, dim=-1)

    # 5) 聚合得到输出并映射回值空间
    x = torch.einsum("bhqt,btc->bhqc", attn, kv_cache)    # (B,H,Q,Ckv)
    out = ____[5]____                                     # (B,H,Q,Vd)
    return out

def demo_mla_step():
    B, H, Dn, Dr, Vd, Lk, Ckv = 2, 8, 64, 32, 128, 10, 512
    q_nope = torch.randn(B, H, 1, Dn)
    q_pe   = torch.randn(B, H, 1, Dr)
    kv_cache = torch.randn(B, Lk, Ckv)
    pe_cache = torch.randn(B, Lk, Dr)
    wkv_b = torch.randn(H, Dn + Vd, Ckv)
    softmax_scale = 1.0 / math.sqrt(Dn + Dr)
    causal_mask = torch.ones(1, 1, 1, Lk)  # 全可见示例

    out = mla_decode_step(q_nope, q_pe, kv_cache, pe_cache, wkv_b,
softmax_scale, causal_mask)
    print("MLA step out shape:", out.shape)  # torch.Size([2, 8, 1, 128])

if __name__ == "__main__":
    demo_mla_step()
```

[1] 需要将 q_nope 通过头依赖的权重矩阵 k_nope_w 投影到 KV 低秩空间。请选择正确的张量运算方式：

A. `torch.einsum("bhqd,hdc->bhqc", q_nope, k_nope_w)`

B. `torch.einsum("bhqd,chd->bhqc", q_nope, k_nope_w)`

C. `torch.matmul(q_nope, k_nope_w)`

D. `torch.einsum("bhdq,hdc->bhqc", q_nope, k_nope_w)`

答案：A

解析：`q_nope` 的形状是 `(B,H,Q,Dn)`，`k_nope_w` 的形状是 `(H,Dn,Ckv)`，需要对每个头分别进行矩阵乘法，将维度 Dn 投影到 Ckv。Einstein求和约定 `"bhqd,hdc->bhqc"` 表示：保持B、H、Q维度，对d维度求和，输出c维度，得到 `(B,H,Q,Ckv)`。B选项的维度顺序错误；C选项无法处理头维度H；D选项的q和d位置颠倒。

[2] 投影后的查询向量需要与 kv_cache 做相关性计算以获得注意力分数。请选择正确的实现方式：

A. `torch.einsum("bhqc,cbt->bhqt", q_nope_proj, kv_cache)`

B. `torch.matmul(q_nope_proj, kv_cache.transpose(-2, -1))`

C. `torch.einsum("bhcq,btc->bhqt", q_nope_proj, kv_cache)`

D. `torch.einsum("bhqc,btc->bhqt", q_nope_proj, kv_cache)`

答案：D

解析：`q_nope_proj` 的形状是 `(B,H,Q,Ckv)`，`kv_cache` 的形状是 `(B,Lk,Ckv)`。需要对Ckv维度求内积，得到注意力分数 `(B,H,Q,Lk)`。Einstein求和 `"bhqc,btc->bhqt"` 表示：对c维度（Ckv）求和，保留b、h、q维度，t对应序列长度Lk。A选项的kv_cache维度顺序错误；B选项未考虑批次和头维度；C选项q_nope_proj的维度顺序错误。

[3] 需要合并两部分注意力分数并应用 softmax_scale 进行缩放。请选择正确的实现方式：

A. `(scores_nope + scores_pe) / q_nope.size(1)`

B. `scores_nope + scores_pe`

C. `(scores_nope + scores_pe) * softmax_scale`

D. `(scores_nope @ scores_pe) / softmax_scale`

答案：C

解析：MLA 将内容匹配分数（scores_nope）和位置关系分数（scores_pe）相加后，需要应用缩放因子 `softmax_scale`（通常为 `1/sqrt(Dn+Dr)`）来稳定训练和推理。直接相乘实现缩放。A选项除以头数量不正确；B选项缺少缩放；D选项使用矩阵乘法且公式错误，两个分数应该相加而非相乘。

[4] 为实现因果注意力，需要将掩码中不可见位置的分数设置为负无穷。请选择正确的实现方式：

A. `logits * causal_mask`

B. `logits.masked_fill(causal_mask == 0, float("-inf"))`

C. `logits.masked_fill(causal_mask == 1, float("-inf"))`

D. `logits + causal_mask`

答案：B

解析：因果注意力要求自回归解码时只能看到当前及之前的位置。掩码中0表示不可见位置，1表示可见位置。`masked_fill(condition, value)` 会将满足条件的位置填充为指定值。将不可见位置（`causal_mask == 0`）设置为 `-inf`，经过softmax后这些位置的注意力权重会变为0。A选项的乘法无法实现-inf填充；C选项将可见位置设为-inf，逻辑相反；D选项的加法无法正确屏蔽。

[5] 最后需要使用 v_w 将聚合后的向量 x 从 KV 空间映射回值空间。请选择正确的实现方式：

A. `torch.einsum("bhqt,hdc->bhqd", attn, v_w)`

B. `torch.matmul(x, v_w)`

C. `torch.einsum("bhqd,hdc->bhqc", x, v_w)`

D. `torch.einsum("bhqc,hdc->bhqd", x, v_w)`

答案：D

解析：注意力加权后的 `x` 形状为 `(B,H,Q,Ckv)`，位于低秩潜在空间中。需要通过头依赖的权重矩阵 `v_w`（形状 `(H,Vd,Ckv)`）将其映射回各头的值空间，得到 `(B,H,Q,Vd)`。Einstein求和 `"bhqc,hdc->bhqd"` 表示：对c维度（Ckv）求和，将其转换为d维度（Vd），保留B、H、Q维度。A选项使用了attn而非x；B选项无法处理头维度；C选项的输出维度是c而非d。