看了你的GRPO训练代码，我有以下改进建议：

🎯 核心改进方向

1. 数据增强与质量优化

```
# 添加数据增强
def augment_gsm8k_data(dataset):
    # 改写问题表述
    # 添加反向问题
    # 生成类似的变体问题
    pass

# 过滤低质量样本
def filter_by_reward_threshold(dataset, threshold=5.0):
    # 预先用模型评估，只保留高质量训练样本
    pass
```

2. 优化Reward函数

当前reward函数可以改进：

```
# 加入推理过程质量评估
def evaluate_reasoning_quality(prompts, completions, **kwargs):
    scores = []
    for completion in completions:
        score = 0
        # 检查是否包含逐步推理
        if "步骤" in completion or "Step" in completion:
            score += 1.0
        # 检查计算过程
        if re.search(r'\d+[\+\-\*/]\d+', completion):
            score += 0.5
        # 惩罚过短的推理
        reasoning = extract_reasoning(completion)
        if reasoning and len(reasoning) < 50:
            score -= 0.5
        scores.append(score)
    return scores
```

```python
# 添加一致性奖励（多次采样答案一致）
def consistency_reward(prompts, completions, **kwargs):
    # 同一问题生成多个答案，一致的给更高奖励
    pass
```

3. 超参数调优

```python
# 当前配置较保守，可以尝试：
RANK = 128  # 增加LoRA rank
ALPHA = 128.0

NUM_GENERATIONS = 8  # 增加生成多样性

BETA = 0.05  # 适当降低KL惩罚，允许更多探索
TEMPERATURE = 1.0  # 提高温度

LEARNING_RATE = 5e-6  # 稍微提高学习率
NUM_EPOCHS = 3  # 多训练几个epoch

MAX_GRAD_NORM = 1.0  # 放宽梯度裁剪
```

4. 课程学习策略

```python
# 从简单到困难训练
def curriculum_learning(dataset):
    # 按问题难度排序（如数字大小、步骤数）
    sorted_data = sorted(dataset, key=lambda x: estimate_difficulty(x))

    # 分阶段训练
    easy = sorted_data[:len(sorted_data)//3]
    medium = sorted_data[len(sorted_data)//3:2*len(sorted_data)//3]
    hard = sorted_data[2*len(sorted_data)//3:]

    return easy, medium, hard
```

5. 集成学习

```python
# 训练多个不同配置的模型
configs = [
```

```
    {"rank": 64, "temp": 0.8},
    {"rank": 128, "temp": 0.9},
    {"rank": 64, "temp": 1.0},
]

# 推理时集成多个checkpoint
def ensemble_predict(question, checkpoints):
    predictions = []
    for ckpt in checkpoints:
        pred = generate_with_ckpt(question, ckpt)
        predictions.append(pred)
    # 投票或平均
    return majority_vote(predictions)
```

6. 优化采样策略

```
GENERATION_CONFIGS = {
    "diverse": {
        "temperature": 1.2,  # 更高温度
        "top_k": 100,
        "top_p": 0.98,
        "num_beams": 4,  # 添加beam search
    },
    "self_consistency": {
        # 生成多个答案取最常见的
        "num_samples": 5,
        "temperature": 0.7,
    }
}
```

7. 添加验证集early stopping

```
# 防止过拟合
best_val_score = 0
patience = 3
no_improve_count = 0

for step in training_steps:
    if step % EVAL_EVERY_N_STEPS == 0:
```

```
        val_score = evaluate(val_dataset)
        if val_score > best_val_score:
            best_val_score = val_score
            save_checkpoint()
            no_improve_count = 0
        else:
            no_improve_count += 1
            if no_improve_count >= patience:
                break
```

8. 错误分析与针对性改进

```
# 收集训练中的错误案例
def analyze_errors(test_results):
    error_types = {
        "format_error": [],
        "calculation_error": [],
        "reasoning_error": [],
    }

    # 对错误类型做针对性数据增强
    for error_type, samples in error_types.items():
        create_synthetic_data(samples)
```

9. 使用更大的base model

如果资源允许，尝试：

- Gemma2-9B 或 Gemma2-27B
- Qwen2.5-7B/14B（数学能力更强）

10. 后处理优化

```
def post_process_answer(raw_output):
    # 提取所有数字
    # 验证计算逻辑
    # 使用符号计算验证（sympy）
    # 格式标准化
    pass
```

🚀 快速见效的Top 3建议

1. **增加NUM_EPOCHS=3, NUM_GENERATIONS=8** - 立即提升性能
2. **优化reward函数** - 加入reasoning quality评估
3. **Self-consistency采样** - 推理时生成多个答案投票

希望这些建议对你的竞赛有帮助！🎉

采样策略：

🎯 核心概念：如何让模型"选择"下一个词

想象模型在写作文，每次要选下一个词时，它会给每个候选词一个概率：

```
例子：模型要接着写 "The cat is ___"

候选词的概率分布：
"sleeping"  → 40%
"playing"   → 25%
"eating"    → 20%
"running"   → 10%
"flying"    → 3%
"dancing"   → 2%
...其他词    → 更低
```

不同的采样策略就是用**不同的规则**来选这个词。

1️⃣ Greedy（贪婪采样）

```
"greedy": {
    "temperature": 1e-4,  # 接近0
    "top_k": 1,
    "top_p": 1.0
}
```

📌 工作原理

永远选概率最高的那个词

```
"The cat is ___"
候选词：sleeping(40%), playing(25%), eating(20%)...

Greedy选择：永远选 "sleeping" ✓
```

🎨 特点

- ✅ **确定性强**：同样的输入，永远得到同样的输出
- ✅ **连贯性好**：输出很"正常"、很"安全"
- ❌ **缺乏创造性**：输出很无聊、很可预测
- ❌ **容易陷入重复**：可能一直重复同样的模式

💡 适用场景

```
# 适合需要"标准答案"的任务
问题："2 + 2 = ?"
Greedy输出："4" ✓ （确定、正确）

问题："法国的首都是？"
Greedy输出："巴黎" ✓ （准确）
```

🔧 参数解释

- `temperature = 0.0001`（接近0）→ 让概率分布变得极端

  ```
  原始：sleeping(40%), playing(25%), eating(20%)temperature=0.0001
  后:sleeping(99.9%), playing(0.05%), eating(0.03%)→ 基本只能选到
  sleeping
  ```

2️⃣ Standard（标准采样）

```
"standard": {
    "temperature": 0.7,
    "top_k": 50,
    "top_p": 0.95
}
```

📌 工作原理

在合理的范围内随机选择

```
"The cat is ___"
候选词: sleeping(40%), playing(25%), eating(20%), running(10%)...

Standard会:
1. 只考虑前50个最可能的词 (top_k=50)
2. 只考虑累计概率达到95%的词 (top_p=0.95)
3. 用温和的随机性选择 (temperature=0.7)

可能的输出:
- "sleeping" (最常见)
- "playing"  (有时)
- "eating"   (偶尔)
- "running"  (很少)
```

🎨 特点

- ✅ **平衡性好**：既合理又有一些变化
- ✅ **适合大多数场景**：通用性强
- ✅ **有一定随机性**：每次输出可能不同
- ⚖️ **中庸之道**：不太极端

💡 适用场景

```
# 适合需要"自然对话"的任务
问题："今天天气怎么样？"
Standard输出：
- "今天阳光明媚" ✓
- "今天天气不错" ✓
- "今天有点多云" ✓
（都合理，但有变化）

问题："给我讲个故事"
Standard输出：会生成合理但不完全一样的故事
```

🔧 参数解释

- `temperature = 0.7` → 稍微平滑概率分布

  ```
  原始：sleeping(40%), playing(25%), eating(20%)

  temperature=0.7后：
  sleeping(38%), playing(27%), eating(22%)

  → 高概率词稍微降低，低概率词稍微提高
  ```

- `top_k = 50` → 只从前50个词中选

  ```
  候选词有1000个，但只考虑概率最高的50个
  → 避免选到很奇怪的词
  ```

- `top_p = 0.95` → 只考虑累计概率95%的词

  ```
  sleeping(40%) + playing(25%) + eating(20%) + running(10%) = 95%
  → 只从这4个词中选，忽略剩下的5%
  ```

3️⃣ Diverse（多样化采样）

```
"diverse": {
    "temperature": 1.2,
    "top_k": 100,
    "top_p": 0.98
}
```

📌 工作原理

鼓励模型"冒险"，选择更多样化的词

```
"The cat is ___"
候选词：sleeping(40%), playing(25%), eating(20%), running(10%),
flying(3%)...

Diverse会：
1. 考虑前100个词 (top_k=100)
2. 考虑累计概率98%的词 (top_p=0.98)
3. 用更强的随机性 (temperature=1.2)

可能的输出：
- "sleeping" (常见)
- "playing"  (常见)
- "eating"   (常见)
- "running"  (经常)
- "flying"   (有时) ← 更有创造性!
- "dancing"  (偶尔) ← 意想不到!
```

🎨 特点

- ✅ **创造性强**：输出更有趣、更意外
- ✅ **多样性高**：每次都很不一样
- ❌ **可能不连贯**：有时会说"奇怪"的话
- ❌ **准确性降低**：可能给出错误答案

💡 适用场景

```
# 适合需要"创意"的任务
问题："写一首关于猫的诗"
Diverse输出：
- 可能用意想不到的比喻
- 可能有独特的表达
- 每次都很不一样

# ❌ 不适合数学题
问题："2 + 2 = ?"
Diverse可能输出："5" ✗ (太随机了!)
```

🔧 参数解释

- `temperature = 1.2` → 让概率分布更平坦

  ```
  原始: sleeping(40%), playing(25%), eating(20%), running(10%), flying(3%)temperature=1.2后:sleeping(30%), playing(24%), eating(20%), running(15%), flying(8%)→ 低概率词的机会大大增加!
  ```

🎯 你的GSM8K数学题应该用哪个?

```
# 训练时（GRPO）
TEMPERATURE = 0.9  # 需要一定多样性来探索
# 原因：让模型尝试不同的推理路径

# 评估时
"greedy"    → 最稳定，看"最佳性能"
"standard" → 平衡，看"实际表现"
"diverse"   → 不推荐用于数学题（太随机）
```

希望这个解释清楚了！有任何问题随时问我 😊

让我用最简单的方式解释 Self-Consistency!

想象你在做一道数学题，不确定答案：

方法1（普通方法）：

```
做一次题 → 得到答案 → 直接提交
风险：可能算错了
```

方法2（Self-Consistency）：

```
做5次题（每次用不同方法）：
第1次：答案是 42
第2次：答案是 42
第3次：答案是 45  ← 这次算错了
第4次：答案是 42
第5次：答案是 42

投票：42出现4次，45出现1次
最终答案：42 ✓ （更有信心！）
```

这就是 Self-Consistency 的核心思想！