24 点推理数据生成与微调实验报告

immediate

2025 年 1 月 26 日

1 数据构建

1.1 24 点游戏实例生成

为了生成实验数据，我们首先需要创建符合要求的 24 点游戏实例。具体而言，我们随机生成 4 个数字（范围为 1 到 13），然后验证这 4 个数字是否可以通过基本数学运算（加、减、乘、除）组合得到结果 24，用于验证的算法如算法 1 所示。通过验证的实例会被加入到数据集中，同时确保数据集中的实例都是唯一的（数字的组合经过去重处理）。最终，我们生成了 1100 个实例，其中1000 个实例用于构建训练集的推理步骤（CoT 数据），另外 100 个实例作为测试集，训练集和测试集之间没有重叠。

1.2 根据实例生成 CoT 数据

1.2.1 CoT 数据格式

Format v1 递归推理. 我们希望构建用于大模型训练的文本数据。所以我们首先要将解决 24 点的过程转化为文本表示。具体而言，我们每次选择两个数字作为操作数，并将操作数用括号括起来。随后选择一种运算方式进行计算，并将计算结果替换原操作数，从而生成一个新的数字组合。每一步操作都包含操作数的计算表达式及更新后的数字组合，逐行记录整个计算过程。我们将这种记录方式称为 format v1，它的灵感来自于 ToT [1]，能够清晰地描述 24 点游戏的推理与计算步骤。最终，当结果达到 24 时，记录完整的表达式以确认成功。这种格式便于追踪每一步操作并验证计算结果的正确性。 Format v1 的示例如下所示：

Algorithm 1: 24 点验证算法

Input: 数字列表 l

Output: True 或者 False

```
if 列表长度为 1 then
    if 列表中数字为 24 then
        return True
    end
    else
        return False
    end
end
for 遍历列表中所有数字的两两组合，表示为 a, b do
    计算 a 和 b 通过四则基本运算所有可能的结果（六种情况）;
    对于每个结果，将其与列表中剩余数字组成新列表 ;
    依次对新列表进行递归，当递归返回 True 时结束，直接返回
    True;
end
return False
```

```
5 13 7 9
(13) + (9) = 22, left: 22, 5, 7
(5) - (22) = -17, left: -17, 7
(7) - (-17) = 24, left: 24
reach 24! expression: (7 - (5 - (13 + 9)))
```

Format v2 引入回滚. 虽然 format v1 能够清晰地表示 24 点游戏的推理步骤，但它未考虑推理过程中可能无法到达正确答案的情况。在实际推理时，模型并不能保证每一步都朝着正确答案前进。当模型选择的计算路径错误时，可能需要回滚到之前的状态，重新选择另一条推理路径。因此，我们在 format v1 的基础上新增了回滚命令，用于处理推理失败的情况。具体而言，当模型判断当前路径无法得出正确答案，此时就触发回滚操作。回滚会撤销最近一步的计算操作，返回到之前的数字组合状态，并尝试其他可能的计算路径。新增的回滚机制使推理过程更加灵活，能够探索不同

的推理分支，从而更接近正确答案。我们将新增回滚的格式称为 format v2，如下所示：

```
5 13 7 9
(13) + (9) = 22, left: 22, 5, 7
(5) - (22) = -17, left: -17, 7
(7) + (-17) = -10, left: -10
roll back, left: -17, 7
(7) - (-17) = 24, left: 24
reach 24! expression: (7 - (5 - (13 + 9)))
```

Format v3 记录表达式. 虽然 format v2 引入了回滚机制，能够处理推理过程中错误路径的撤销和重试，但当模型生成的推理序列非常长时，直接回滚到之前的状态可能变得更为困难。为了解决这一问题，我们对格式进行了进一步改进，提出了 format v3。在 format v3 中，我们将 left 右侧的数值记录扩展为对应的数学表达式。通过这种改写方式，每次操作都会将当前的推理步骤嵌套到一个完整的表达式中。回滚操作只需要拆除表达式最外层的括号即可恢复到之前的状态，而不需要回顾序列去找到过去的值。这种改进使得回滚的正确性得到提高，同时也便于验证和追踪。以下是一个 format v3 的推理示例：

```
5 13 7 9
(13) + (9) 22, left: (13 + 9) = 22, 5, 7
(5) - (22) = -17, left: (5 - (13 + 9)) = -17, 7
(7) + (-17) = -10, left: (7 + (5 - (13 + 9))) = -10
roll back, left: (5 - (13 + 9)) = -17, 7
(7) - (-17) = 24, left: (7 - (5 - (13 + 9))) = 24
reach 24! expression: (7 - (5 - (13 + 9)))
```

1.2.2 CoT 数据生成

完整推理路径搜索. 为了构造 1.2.1 节中所描述格式的数据，我们可以采用算法 1 的搜索流程。在搜索过程中，每次递归都将当前步骤按照 format v3 的格式记录下来。需要注意的是，为了保证生成样本的多样性，每次选择数字对的顺序是随机的，同时选择运算符的顺序也具有随机性。当找到正确答案时，搜索过程立即停止，此时生成了一棵完整的搜索树。

搜索树裁剪. 然而，这棵搜索树的节点数目通常会超过数千个。如果将所有节点的内容转化为文本，一个样例的解步骤序列的 token 数量可能会超过 10k。这种超长的序列不仅难以处理，还包含大量无效的探索路径。因此，我们需要对搜索树进行裁剪以减小其规模。具体来说，我们采用随机裁剪的方法：每次随机删除搜索树中的一个叶子节点，并重复这一操作，直到搜索树的节点数目减少到设定的阈值以下。裁剪后的搜索树能够更紧凑地表示有效的推理路径，同时保留了一定的随机性和探索性。最终，我们遍历裁剪后的搜索树来加入回滚步骤文本，然后将所有步骤的文本组合成为符合 format v3 格式的文本序列。

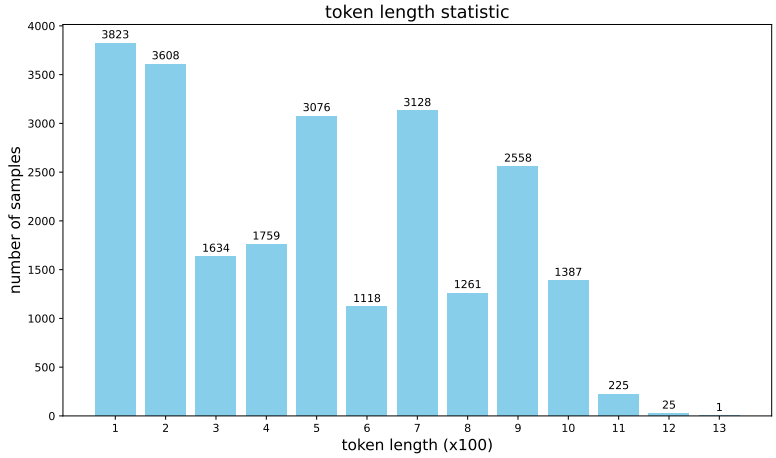

图 1: 生成的 CoT 数据的 token 长度分布

数据集	Short [0,300)	Medium [300,700)	Long [700,1400)
样本数量	7431	7587	8585

表 1: 按 token 长度划分的数据集的样本数量

针对每个 24 点游戏实例，我们随机进行了 5 次搜索，以构建不同的搜索树。对于每棵搜索树，我们设置了多个裁剪停止节点数量阈值：4（对应最佳路径）、8、12、16 和 20，每个我们用每个阈值进行一次裁剪。通过对裁剪后的搜索树展开为文本序列，并对生成的文本序列进行去重处理，最终得到了 23,603 条数据。生成的推理链（CoT）数据的 token 长度分布如图 1 所示。根据 token 长度，我们将数据划分为三个子数据集，划分后的

样本数量如表 1 所示。

我们生成的一个的一个数据样本如下所示：

```
5 13 7 9
(7) / (9) = 7/9, left: (7 / 9) = 7/9, 5, 13
(7/9) / (13) = 7/117, left: ((7 / 9) / 13) = 7/117, 5
roll back, left: (7 / 9) = 7/9, 5, 13
(5) / (13) = 5/13, left: (5 / 13) = 5/13, (7 / 9) = 7/9
roll back, left: (7 / 9) = 7/9, 5, 13
roll back, left: 5 13 7 9
(7) + (9) = 16, left: (7 + 9) = 16, 5, 13
(16) - (5) = 11, left: ((7 + 9) - 5) = 11, 13
(13) + (11) = 24, left: (13 + ((7 + 9) - 5)) = 24
reach 24! expression: (13 + ((7 + 9) - 5))
```

可以看到我们并不是到只剩一个数字才进行验证和回滚，我们在中间步骤（比如第五行还剩两个数字）就可以回滚。这是和验证算法最大的一个区别，**如果模型能够直接从一些数字组合直接判断是否能到达 24 点，则可以免去大量的探索。**

2 实验

2.1 实验设置

训练设置. 我们采用 Qwen 2.5 0.5B 作为基座模型。我们设置 batch size 为 8，梯度累积 4 次，实际 batch size 为 32。训练最大步数为 1200 步，训练时将训练集的 0.01 划分为验证集，每 100 步计算验证集上的损失，用验证集损失最低的 checkpoint 在测试集上进行评估。学习率设置为 5e-5，最大步数的 3% 使用学习率 warmup，剩余步数学习率余弦衰减，优化器采用 AdamW。 weight decay 设置为 0.01，使用混合精度训练。

评估方法. 我们将测试集的实例转化为字符串，4 个数字间以空格分隔，直接输入大模型中，不使用其他 prompt。我们设置最大长度为 4096，当文本达到这个长度时就停止生成。我们将生成的文本按行划分，检查最后一行是否包含输出表达式。如果包含表达式，我们抽取表达式中的数字，将其与测试集实例进行对比。如果数字组合和测试集实例一致且表达式的

值解析为 24 点（允许 1e-6 的除法精度误差），我们认为生成答案正确。我们采用准确率（Accuracy）作为评估指标。

2.2 用不同数据集训练结果对比

我们在 Short、Medium 和 Long 三个数据集上分别微调模型，并使用相同的测试集实例对其进行评估。为了分析模型生成推理链的长度特性，我们统计了模型输出结果的 token 长度分布，结果如图 2 所示。从图中可以观察到，随着训练数据集中推理长度的增加，模型生成的推理链长度也呈现出相应的增长趋势。这表明训练数据的推理长度对模型的输出行为具有显著影响。例如，微调于 Short 数据集的模型倾向于生成较短的推理链，而微调于 Long 数据集的模型则更倾向于生成更长的推理链。此外，图中最右侧样本数量相对较多是因为被截断的生成都在这个区间（4096 个 token）。这些样本的生成长度超过了模型的最大输出长度限制，说明训练于较长推理链的数据集时，模型在处理复杂推理任务时可能倾向于生成更长的序列，但同时也更容易触碰输出长度的上限。

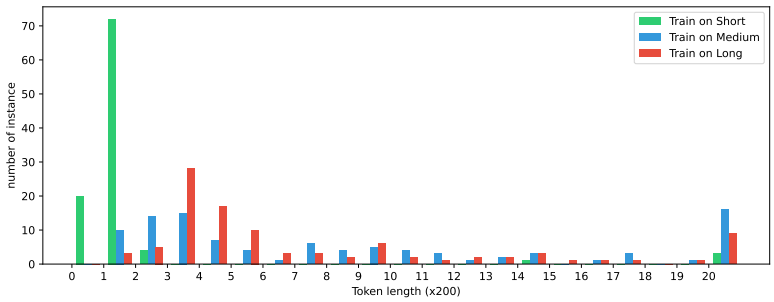

图 2: 不同模型用测试集实例输出的推理步骤的 token 长度

训练数据集	Short	Medium	Long
测试集准确率	30%	69%	59%

表 2: 在不同数据集上微调的模型在测试集上的评测结果

表 2 列出了在不同数据集上微调的模型在测试集上的准确率。从表中可以看出在 Medium 数据集上微调的模型表现最好，准确度达到了 69%。我们没有对超参数进行搜索，模型表现也许还有一定的提升空间。

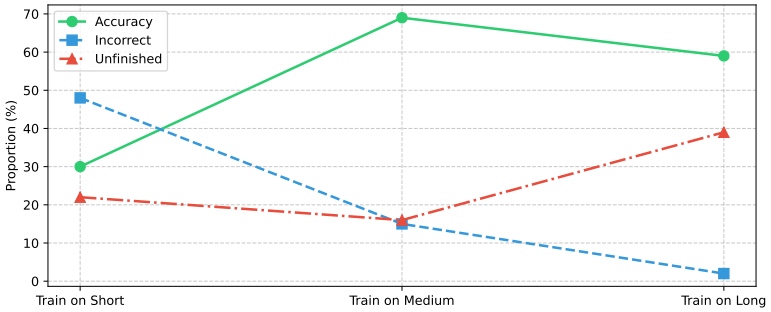

图 3: 不同数据集（推理长度不一样）上微调的模型的表现对比

在图 3 中，我们进一步对评测的错误率（输出最终表达式，但数字组合不对或解析值不为 24 的结果比率）和未完成率（没有输出最终表达式的结果比率）进行分析。相较于 Medium 数据集训练的模型，Long 上训练的模型错误率更低（2%）。所以我们推测 Long 上训练的模型准确率不如 Medium 上训练的模型可能是受制于最大输出长度限制，当将输出长度增加后，准确率可能会有所提高。另一方面，在 Short 上训练的模型比在 Medium 上训练的模型未完成率更高，尽管图 2 表明其大部分输出没有达到最大长度。这说明在没有达到最大长度的情况下， Short 上训练的模型也输出了很多不完整的推理。

2.3 消融实验

为了验证我们设计的推理格式的有效性，我们将 format v3 的 Medium 数据集转化为 format v1 和 format v2 数据集。然后在其上进行微调，我们在附录提供了一个转化的样例。实验结果如表 3 所示。Format v3 取得了最高的准确率。

训练数据集	Format v1	Format v2	Format v3 (Medium)
测试集准确率	46%	56%	69%

表 3: 在不同数据集上微调的模型在测试集上的评测结果

图 4 的结果也表明格式的改进能稳步提高准确率，错误率也在同步降低。三个数据集上的未完成率则相对稳定，都在 20% 左右。

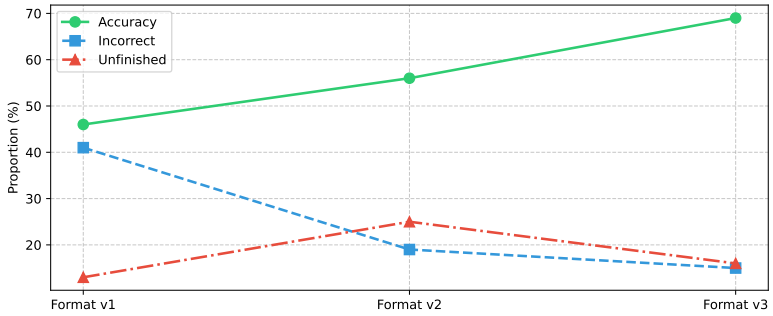

图 4: 不同格式的数据上训练的模型的表现对比

3 讨论

这一节我们将讨论可能有效，但由于时间限制，我们没有进行进一步实验验证的方法。

3.1 混合数据微调

Short 数据集包含最优路径，能够增强模型选择最优路径的能力，但是缺乏对错误路径的训练，容易产生幻觉。Long 数据集则包含许多回滚操作，可以让模型更好地识别无效数字组合性。Medium 数据集微调效果好，可能是因为结合了两者的优点。如果用一定比例混合三个数据集，可能能取得更好的表现。

3.2 构造偏好数据并利用 DPO 进行优化

在有限的数据上进行微调后，模型已经具备了初步解决 24 点游戏的能力。然而，如果继续在同一构造的微调数据上进行训练，模型可能会面临较大的过拟合风险，从而难以进一步提升其推理能力。为了解决这一问题并进一步提高模型性能，我们可以使用初步微调的模型对训练集实例进行采样，生成多个推理步骤，并利用这些生成结果构造新的训练数据集。

具体而言，对于每个训练实例，我们采样 n 次生成推理步骤，并找出其中包含错误推理的样本。对于每个错误样本，我们依次检查其推理步骤，直到定位到第一个错误步骤。接着，我们将错误步骤之前的文本作为输入 x，将错误步骤及其之后的部分作为错误的输出 y_l。同时，我们利用算法从

x 开始重新构造正确的推理 y_w，并对其进行适当裁剪以确保输出的合理性和长度控制。通过这一过程，我们可以构建一个包含模型推理错误及其修正的对比数据集 $\mathcal{D} = \{(x^{(i)}, y_l^{(i)}, y_w^{(i)})_{i=1,\ldots,N}\}$，其中 N 是总共收集到的错误样本数量。

然后，我们就可以利用收集到的数据结合 DPO 算法 [2] 对模型进行优化。DPO 的目标是直接引导模型更倾向于生成正确的推理 y_w，而非错误的推理 y_l。它通过以下损失函数实现优化：

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} \right) \right]. \tag{1}$$

上述过程可以迭代进行。这种基于错误样本构造和对比优化的训练策略，能够有效缓解过拟合问题，同时引导模型针对推理中的薄弱环节进行改进，从而进一步提升其对复杂推理任务的鲁棒性和准确性。

参考文献

[1] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822. Curran Associates, Inc., 2023.

[2] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

A 附录

Format v3 数据转化为 format v2 和 format v1 数据的样例。这里的 format v1 进行了隐式回滚，如果不进行回滚则只能在最优路径上进行微调（Short 数据集的子集），效果比用 Short 数据集微调还差。

```
format v3
5 13 7 9
(7) / (9) = 7/9, left: (7 / 9) = 7/9, 5, 13
(7/9) / (13) = 7/117, left: ((7 / 9) / 13) = 7/117, 5
roll back, left: (7 / 9) = 7/9, 5, 13
(5) / (13) = 5/13, left: (5 / 13) = 5/13, (7 / 9) = 7/9
roll back, left: (7 / 9) = 7/9, 5, 13
roll back, left: 5 13 7 9
(7) + (9) = 16, left: (7 + 9) = 16, 5, 13
(16) - (5) = 11, left: ((7 + 9) - 5) = 11, 13
(13) + (11) = 24, left: (13 + ((7 + 9) - 5)) = 24
reach 24! expression: (13 + ((7 + 9) - 5))

format v2
5 13 7 9
(7) / (9) = 7/9, left: 7/9, 5, 13
(7/9) / (13) = 7/117, left: 7/117, 5
roll back, left: 7/9, 5, 13
(5) / (13) = 5/13, left: 5/13, 7/9
roll back, left: 7/9, 5, 13
roll back, left: 5 13 7 9
(7) + (9) = 16, left: 16, 5, 13
(16) - (5) = 11, left: 11, 13
(13) + (11) = 24, left: 24
reach 24! expression: (13 + ((7 + 9) - 5))

format v1
5 13 7 9
```

```
(7) / (9) = 7/9, left: 7/9, 5, 13
(7/9) / (13) = 7/117, left: 7/117, 5
(5) / (13) = 5/13, left: 5/13, 7/9
(7) + (9) = 16, left: 16, 5, 13
(16) - (5) = 11, left: 11, 13
(13) + (11) = 24, left: 24
reach 24! expression: (13 + ((7 + 9) - 5))
```