24-Point Reasoning Data Generation and Fine-Tuning Experiment Report

Mengqi Liao

February 2, 2025

1 Data Construction

1.1 Generation of 24-Point Game Instances

To generate experimental data, it is first necessary to create valid instances of the 24-point game. Specifically, we enumerate all combinations of four numbers within the range of 1 to 13. Subsequently, we verify whether these four numbers can be combined using basic arithmetic operations (addition, subtraction, multiplication, and division) to yield the result of 24. The verification process is conducted using the algorithm described in Algorithm 1. Instances that pass the verification process are added to the dataset, ensuring that all instances in the dataset are unique (duplicates are removed by treating combinations as equivalent regardless of the order of the numbers). Ultimately, we generated a total of 1362 instances. Among these, 1262 instances were used to construct the reasoning steps for the training set (Chain-of-Thought, CoT data), while the remaining 100 instances were designated as the test set. There is no overlap between the training and test sets.

1.2 Generation of CoT Data from Instances

1.2.1 CoT Data Format

Format v1: Recursive Reasoning. To construct text data suitable for training large models, it is essential to transform the process of solving the 24-point game into a textual representation. Specifically, at each step, two numbers are selected as operands, and these operands are enclosed in parentheses. Subsequently, an arithmetic operation is applied, and the result of the operation replaces the original operands, thereby generating a new combination of numbers. Each step of this process includes the mathematical expression of the operation performed and the updated combination of numbers, with the entire calculation process recorded step by step. This recording method, referred to as format v1, is inspired by ToT [1] and provides a clear description of the reasoning and computational steps involved in the 24-point game. Ultimately, when

Algorithm 1: 24-Point Verification Algorithm

```
Input: List of numbers l
Output: True or False
if The length of the list is 1 then
    if The number in the list is 24 then
        return True;
    end
    else
        return False;
    end
end
for Each pair of numbers in the list, denoted as a and b do
    Compute all possible results of a and b using the four basic
      arithmetic operations (six cases in total);
    For each result, form a new list by combining it with the remaining
      numbers in the original list;
    Recursively apply the algorithm to the new list; if any recursive call
      returns True, terminate and return True;
end
return False;
```

the result reaches 24, the complete expression is recorded to confirm success. This format facilitates the tracking of each operation and the verification of the correctness of the computational results. An example of format v1 is shown below:

```
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: Introducing Rollback. While format v1 provides a clear representation of the reasoning steps for the 24-point game, it does not account for situations where the correct solution cannot be reached during the reasoning process. In practical reasoning scenarios, the model cannot guarantee that every step will progress toward the correct solution. When the computational path chosen by the model is incorrect, it may be necessary to roll back to a previous state and select an alternative reasoning path. To address this limitation, we introduce a rollback command based on format v1 to handle reasoning failures. Specifically, when the model determines that the current path cannot lead to the correct solution, a rollback operation is triggered. This rollback undoes the most recent computational operation, returning to the previous state of the number combination, and attempts other potential computational paths.

The added rollback mechanism enhances the flexibility of the reasoning process, enabling the exploration of different reasoning branches and thus increasing the likelihood of arriving at the correct solution. This extended format is referred to as format v2, as illustrated below:

```
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: Recording Expressions. Although format v2 introduces a rollback mechanism to handle the undoing and retrying of erroneous paths during reasoning, directly rolling back to a previous state can become increasingly challenging when the reasoning sequence generated by the model is very lengthy. To address this issue, we further refine the format and propose format v3. In format v3, the values on the right-hand side of each step (denoted as ”left” in previous formats) are expanded to include the corresponding mathematical expressions. With this modification, each operation embeds the current reasoning step into a complete expression. When a rollback operation is required, it suffices to remove the outermost parentheses of the expression to return to the previous state, without the need to revisit the sequence and locate past values. This improvement enhances the accuracy of rollback operations while also facilitating verification and tracking. Below is an example of reasoning in format v3:

```
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 Data Generation

Complete Reasoning Path Search. To construct data in the formats described in Section 1.2.1, the search process from Algorithm 1 can be utilized. During the search, each recursive step is recorded in the format of format v3. It is important to note that, to ensure diversity in the generated samples, the order of selecting number pairs is randomized, as is the order of selecting operators. Once the correct solution is found, the search process terminates immediately, resulting in the creation of a complete search tree.

Pruning the Search Tree. However, the number of nodes in the search tree typically exceeds several thousand. If the content of all nodes were converted into text, the token count for the solution sequence of a single sample could exceed 10k. Such excessively long sequences are not only challenging to process but also contain a substantial amount of invalid exploratory paths. Therefore, it is necessary to prune the search tree to reduce its size. Specifically, we employ a random pruning approach: at each iteration, one leaf node of the search tree is randomly deleted, and this process is repeated until the number of nodes in the search tree falls below a predefined threshold. The pruned search tree provides a more compact representation of valid reasoning paths while retaining a degree of randomness and exploratory diversity. Finally, the pruned search tree is traversed to incorporate rollback steps into the text representation. All steps are then combined to form a textual sequence that adheres to the format v3 specification.

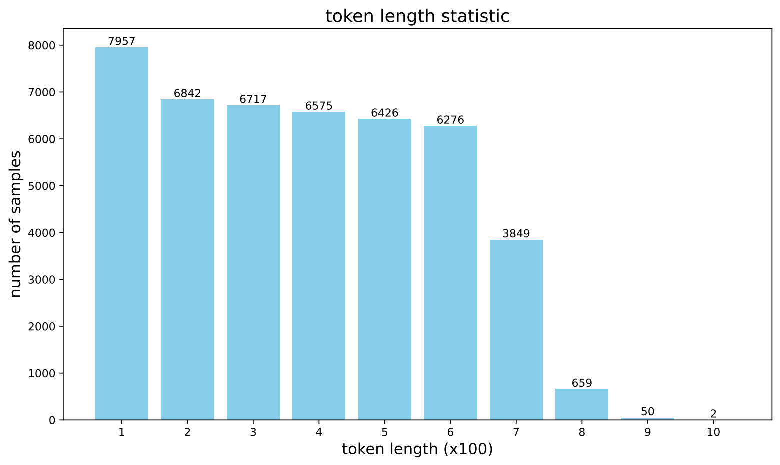

Figure 1: Token length distribution of the generated CoT data.

Dataset	Short [0,300)	Medium [300,550)	Long [550,1100)
Number of Samples	14,799	16,535	14,019

Table 1: Number of samples in the dataset categorized by token length.

For each instance of the 24-point game, we conducted three randomized searches to construct distinct search trees, with the input order being shuffled each time. For every search tree, we applied 12 different pruning thresholds for the number of terminal nodes (ranging from 6 to 17), pruning the tree once for each threshold. The pruned search trees were then expanded into textual sequences, and duplicate sequences were removed. This process ultimately yielded a total of 45,353 data samples.

The token length distribution of the generated reasoning chain (CoT) data is illustrated in Figure 1. Based on token length, the data was divided into three sub-datasets. The number of samples in each sub-dataset is presented in Table

1.

An example of a generated data sample is as follows:

```
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))
```

It can be observed that validation and rollback are not deferred until only one number remains. Instead, rollback can occur at intermediate steps (e.g., when two numbers remain on the fifth line). This constitutes a major distinction from the validation algorithm. **If the model can directly determine whether a combination of numbers can reach 24, it could eliminate a significant amount of exploration.**

2 Experiments

2.1 Experimental Setup

Training Configuration. We employ Qwen 2.5 0.5B as the base model. The batch size is set to 8, with gradient accumulation performed over 4 steps, resulting in an effective batch size of 32. The total number of training steps is capped at 1000. During the training phase, 1% of the training set is designated as the validation set. Validation loss is calculated every 100 steps, and the checkpoint with the lowest validation loss is selected for evaluation on the test set. The learning rate is initialized at 5e-5, with the first 3% of training steps dedicated to learning rate warmup, followed by cosine decay for the remaining steps. The optimizer used is AdamW, with a weight decay coefficient of 0.01. Mixed-precision training is utilized to optimize computational efficiency. The prompt used for the model is as follows:

```
<|im_start|>system
Play 24 game. Given four numbers, determine
if it's possible to reach 24 through basic arithmetic operations.
Output the reasoning steps, one step per line. On the last line,
output the expression, for example, input: '10 1 12 3', the last
line should output: 'reach 24! expression: ((12 + 3) + (10 - 1))'.
<|im_end|>
<|im_start|>user
{input}
```

```
<|im_end|>
<|im_start|>assistant
```

Evaluation Method. We convert instances from the test set into strings, with the four numbers separated by spaces, and directly input them into the large model without using additional prompts. The maximum sequence length is set to 4096; generation ceases once this length is reached. The generated text is split line by line, and the final line is examined for the presence of an output expression. If an expression is present, the numbers in the expression are extracted and compared with the test set instance. If the combination of numbers matches the test set instance and the value of the expression evaluates to 24 (allowing for a division precision error of up to 1×10^{-6}), the generated answer is considered correct. Accuracy is used as the evaluation metric.

2.2 Comparison of Results Across Different Datasets

We fine-tune the model on the Short, Medium, and Long datasets, respectively, and evaluate them on the same test set instances. To analyze the length characteristics of the reasoning chains generated by the model, we calculate the token length distribution of the model's output results, as shown in Figure 2. From the figure, it can be observed that as the reasoning length in the training dataset increases, the length of the reasoning chains generated by the model also exhibits a corresponding growth trend. This indicates that the reasoning length in the training data significantly influences the model's output behavior. For instance, the model fine-tuned on the Short dataset tends to generate shorter reasoning chains, while the model fine-tuned on the Long dataset is inclined to produce longer reasoning chains.

Additionally, the relatively higher number of samples on the far-right of the distribution is due to generated sequences being truncated at the maximum length limit of 4096 tokens. These truncated samples suggest that when trained on datasets with longer reasoning chains, the model tends to generate longer sequences when handling complex reasoning tasks. However, it also becomes more likely to reach the maximum output length limit.

Training Dataset	Short	Medium	Long
Test Set Accuracy	57%	72%	50%

Table 2: Evaluation results of models fine-tuned on different datasets when tested on the same test set.

Table 2 summarizes the test set accuracy of models fine-tuned on different datasets. From the table, it is evident that the model fine-tuned on the Medium dataset performs the best, achieving an accuracy of 72%. Hyperparameter tuning was not conducted, suggesting there may still be room for further improvement in model performance.

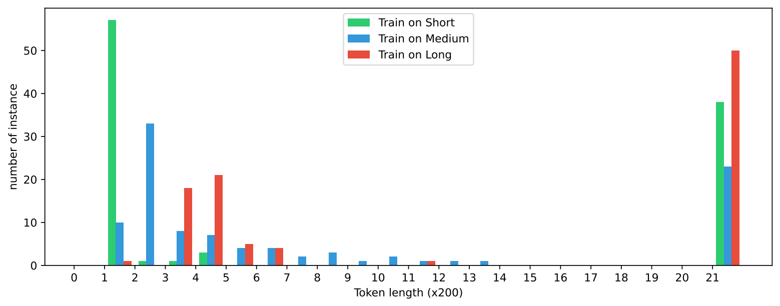

Figure 2: Token length of reasoning steps output by models fine-tuned on different datasets when evaluated on the test set.

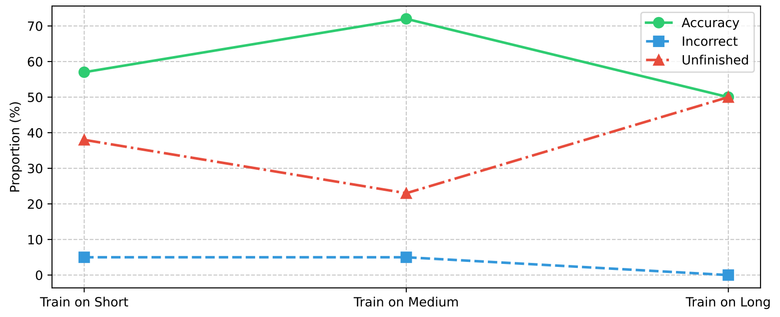

Figure 3: Performance comparison of models fine-tuned on datasets with different reasoning lengths.

In Figure 3, we further analyze the error rate (the proportion of outputs with a final expression where the number combination is incorrect or the evaluated value does not equal 24) and the incompletion rate (the proportion of outputs without a final expression). Compared to the model fine-tuned on the Medium dataset, the model trained on the Long dataset exhibits a lower error rate (0%). Thus, we hypothesize that the lower accuracy of the model trained on the Long dataset compared to the Medium dataset may be constrained by the maximum output length limit. Increasing the output length limit might improve the accuracy. On the other hand, the model fine-tuned on the Long dataset has the highest incompletion rate, potentially due to the excessively lengthy reasoning steps. However, despite the shorter reasoning chains in the Short dataset, the model fine-tuned on the Short dataset has a higher incompletion rate compared to the model fine-tuned on the Medium dataset.

2.3 Experimental Results Using Different Data Formats

To validate the effectiveness of the reasoning formats we designed, datasets with mixed reasoning lengths were created by combining data of various lengths from format v3. These mixed datasets were then converted into format v1 and format v2 datasets. Fine-tuning was subsequently performed on each dataset. A sample conversion is provided in the appendix.

Training Dataset	Format v1	Format v2	Format v3
Test Set Accuracy	66%	84%	74%
Test Set Error Rate	9%	2%	1%

Table 3: Evaluation results of models fine-tuned on different data formats when tested on the test set.

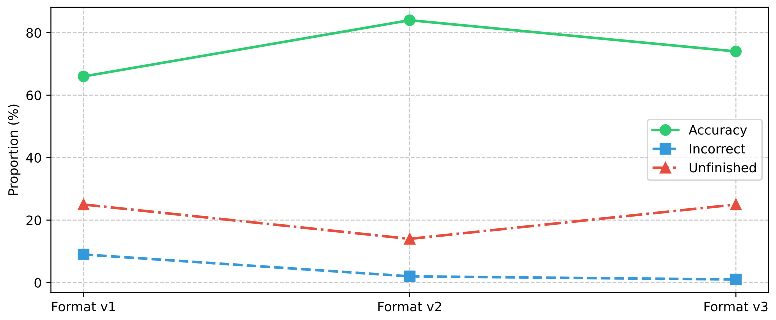

Figure 4: Performance comparison of models trained on datasets with different formats.

The experimental results are presented in Table 3 and Figure 4. Format v2 achieved the highest accuracy, while format v3 exhibited the lowest error rate. These findings suggest that combining data with varying reasoning lengths is more effective than using reasoning data of a single length. Furthermore, the lower accuracy of format v3 compared to format v2 can be attributed to the increased complexity of the v3 output, which makes it more likely to reach the maximum token limit, resulting in incomplete reasoning for many samples. Compared to format v1, format v2 shows significant improvements in both completion rates and error rates.

3 Discussion

In this section, we discuss potentially effective methods that, due to time constraints, we were unable to validate through additional experiments.

3.1 Constructing Preference Data and Optimizing with DPO

After fine-tuning on a limited dataset, the model has developed an initial ability to solve the 24-point game. However, continued training on the same constructed fine-tuning dataset may lead to a significant risk of overfitting, thereby hindering further improvement in the model's reasoning capabilities. To address this issue and further enhance model performance, we propose leveraging the initially fine-tuned model to sample reasoning steps from training set instances, thereby constructing a new training dataset.

Specifically, for each training instance, we sample reasoning steps n times and identify samples containing erroneous reasoning. For each erroneous sample, we sequentially inspect its reasoning steps to locate the first incorrect step. Subsequently, we treat the text preceding the incorrect step as the input x, the incorrect step and all subsequent steps as the incorrect output y_l, and reconstruct the correct reasoning y_w starting from x using the algorithm. The reconstructed reasoning y_w is appropriately trimmed to ensure output reasonableness and length control. This process allows us to construct a contrastive dataset containing the model's reasoning errors and their corrections:

$$\mathcal{D} = \{(x^{(i)}, y_l^{(i)}, y_w^{(i)})_{i=1,\ldots,N}\},$$

where N denotes the total number of collected erroneous samples.

We can then use the collected data in conjunction with the Direct Preference Optimization (DPO) algorithm [2] to optimize the model. The goal of DPO is to directly guide the model towards generating correct reasoning y_w rather than incorrect reasoning y_l. This is achieved through the following loss function:

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

The above process can be iteratively performed. This training strategy, based on error sample construction and contrastive optimization, effectively mitigates overfitting while directing the model to address weaknesses in its reasoning. Consequently, the model's robustness and accuracy in handling complex reasoning tasks are further enhanced.

References

[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 Appendix

Examples of converting format v3 data into format v2 and format v1 data are provided below. In the case of format v1, implicit rollback is applied. Without rollback, fine-tuning would be limited to the optimal path (a subset of the Short dataset), resulting in performance inferior to that achieved by fine-tuning with the Short dataset.

```
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))
```