In practice, when t does not evenly divide T, the final step performs $T \mod t$ operations. To guide the model in generating the desired CoT length, we insert the control token `<t>` after the question and before the beginning of the solution. To preserve the parentheses that indicate the order of operations, we construct expressions in Polish notation. However, for readability, we present each problem in its conventional form throughout the article.

B.2 Contrast to vanilla arithmetic problem

Why pruning? Initially, we intended to create a synthetic dataset for regular arithmetic tasks, but we quickly realized that the computation tree for such tasks is uncontrollable. For example, consider the task $1 * 2 + 3 * 4$. We hoped to compute 2 operators in one step, but found it impossible because the addition needs to be computed after the two multiplications, and we cannot aggregate two multiplications in one subtask. Therefore, pruning the computation tree becomes essential.

Why only focusing on addition? There are two reasons why we focus on arithmetic tasks involving only addition: first, it simplifies pruning, as the order of operations can be controlled solely by parentheses; second, it facilitates the computation of sub-tasks, since parentheses do not affect the final result, and the model only needs to compute the sum of all the numbers when solving a sub-task. We aim for the model to handle longer sub-tasks, thereby allowing a broader study of the impact of CoT length.

Will the simplified synthetic dataset impact the diversity of the data? We need to clarify that even with pruning, the structure of the expressions will still vary because swapping the left and right child nodes of each non-leaf node in the computation tree results in different expressions. When $T > 30$, the number of possible variations exceeds 1×10^9.

C Supplementary Details on Real world Experiment for Optimal CoT Length

C.1 Implementation Details

Solution Length Control. To study the impact of CoT length on performance under a given problem difficulty, we need to induce the model to naturally generate solutions of varying lengths. Simply adding prompts like "*please use 100 tokens to solve this problem*" or "*please use 10 steps to solve this problem*" is not ideal because the model's ability to follow instructions regarding output length is limited, and such fixed-length prompts may not ensure fairness across problems of different difficulties. Moreover, prompting for a specific length might lead the model to generate irrelevant tokens or steps just to "pad the length," without actually changing the number of steps or the complexity of the reasoning. Additionally, controlling `max_length` is also problematic, as overly long responses might get truncated, which would directly lead to lower accuracy for longer outputs. What we really want is for the model to generate a complete and coherent long response on its own, so we can observe the corresponding accuracy.

To create solutions with varying step lengths with different complexity, we follow [12] by using in-context examples (8-shots) with three different levels of complexity to guide the model in generating solutions with different step counts. For each set of in-context examples, we sample 20 times, resulting in a total of 60 samples per question.

Step Segmentation. Simply measuring CoT length by counting tokens is neither rigorous nor meaningful. Since our focus is on final performance rather than efficiency, we care more about using CoT length to reflect the complexity of the reasoning pattern. In this sense, the number of reasoning steps can serve as a more appropriate indicator of CoT length. As we discussed earlier, the step number captures how the model decomposes the problem, which directly reflects the complexity of its reasoning. In contrast, token length fails to capture this because, as the model thinks more deeply and the number of steps increases, the number of tokens per step may decrease—making the total token count unpredictable and unreliable as a proxy for reasoning complexity.

When calculating the number of steps, we separate the full reasoning chain using `"\n"`[12] and remove empty lines caused by `"\n\n"`. Then we consider the total number of lines as the CoT length. Since questions in the MATH dataset are challenging and lead to high variability in final CoT lengths, we normalize the lengths by applying `length = length // bin_width`. For experiments comparing different models (e.g., optimal CoT length per model or optimal vs. longest CoT), the

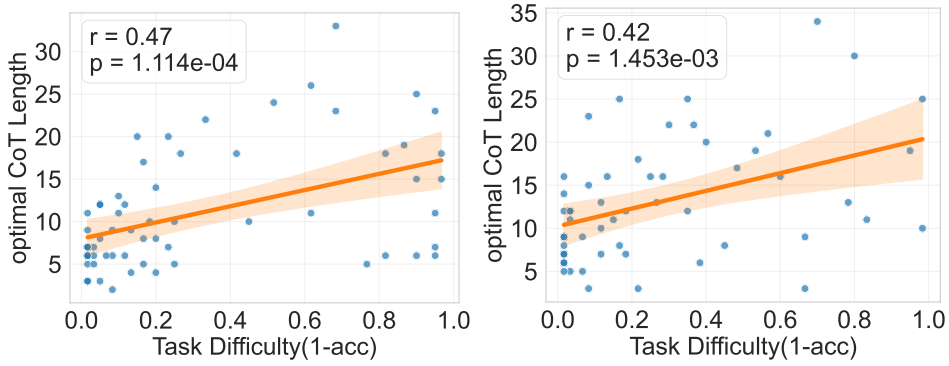
questions within each length bin differ (though only 30 per group), which introduces variability. To reduce this variance and ensure each bin has enough samples, we use a relatively large bin width of 5. In contrast, for analyzing the influence of task difficulty, where each calculation on optimal CoT length only contains one question, we adopt a finer bin width of 2 for better resolution (we also verified that using width 1 yields almost identical results).

More Details of Figure 2b. When evaluating the results, questions with accuracy < 0.01 or > 0.99 (indicating all incorrect or all correct responses) are excluded, as their accuracy does not vary with step length changes.

To better understand the reliability of the observed trend between task difficulty and optimal Chain-of-Thought (CoT) length, we compute a 95% confidence interval around the linear regression line. Specifically, we use standard methods based on the Student's t-distribution to estimate uncertainty in the predicted values. The confidence band reflects how much the estimated mean CoT length is expected to vary given the finite sample size and the distribution of data points.

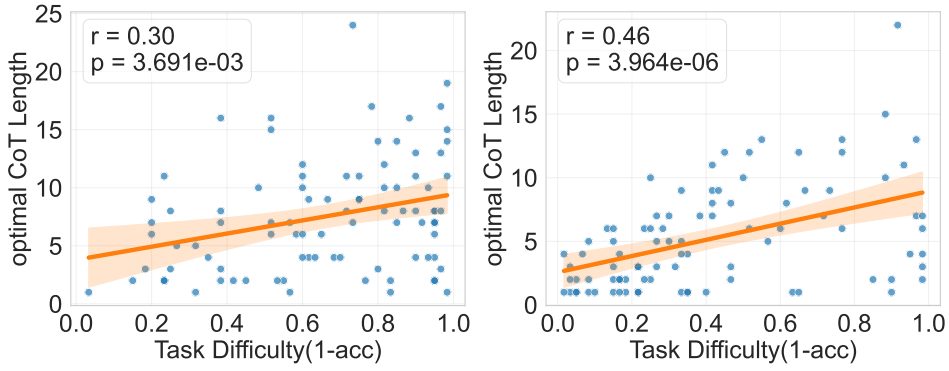
C.2 More Experimental Results

Additional results for Figure 2b. To further investigate the relationship between task difficulty and optimal CoT lengths on real world datasets, we conduct experiments on different models. The results (Figure 6 and 7) are impressive that results on all models show a significant correlation between the task difficulties and optimal lengths.



(a) Qwen2.5 7B Instruct (b) Qwen2.5 14B Instruct

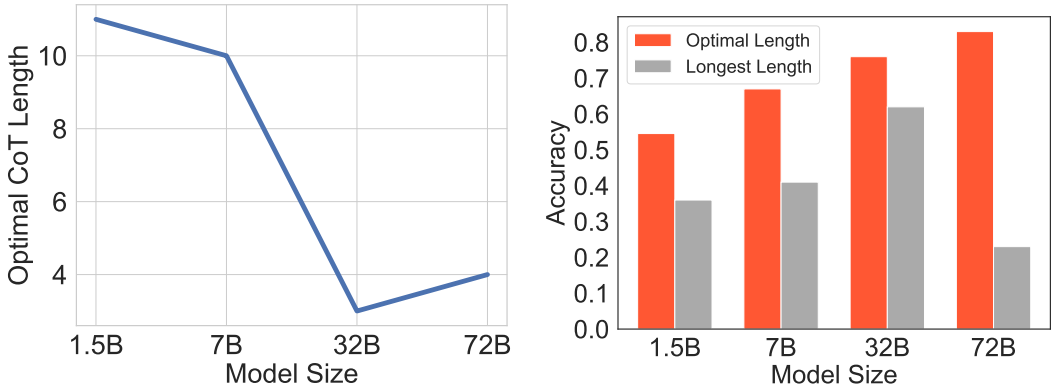
Figure 6: Evaluation between task difficulties and optimal CoT lengths on MATH datasets with Qwen2.5 Series Instruct models.



(a) Llama 3.1 8B Instruct (b) Llama 3.1 70B Instruct

Figure 7: Evaluation between task difficulties and optimal CoT lengths on MATH datasets with LLama3.1 Series Instruct models.

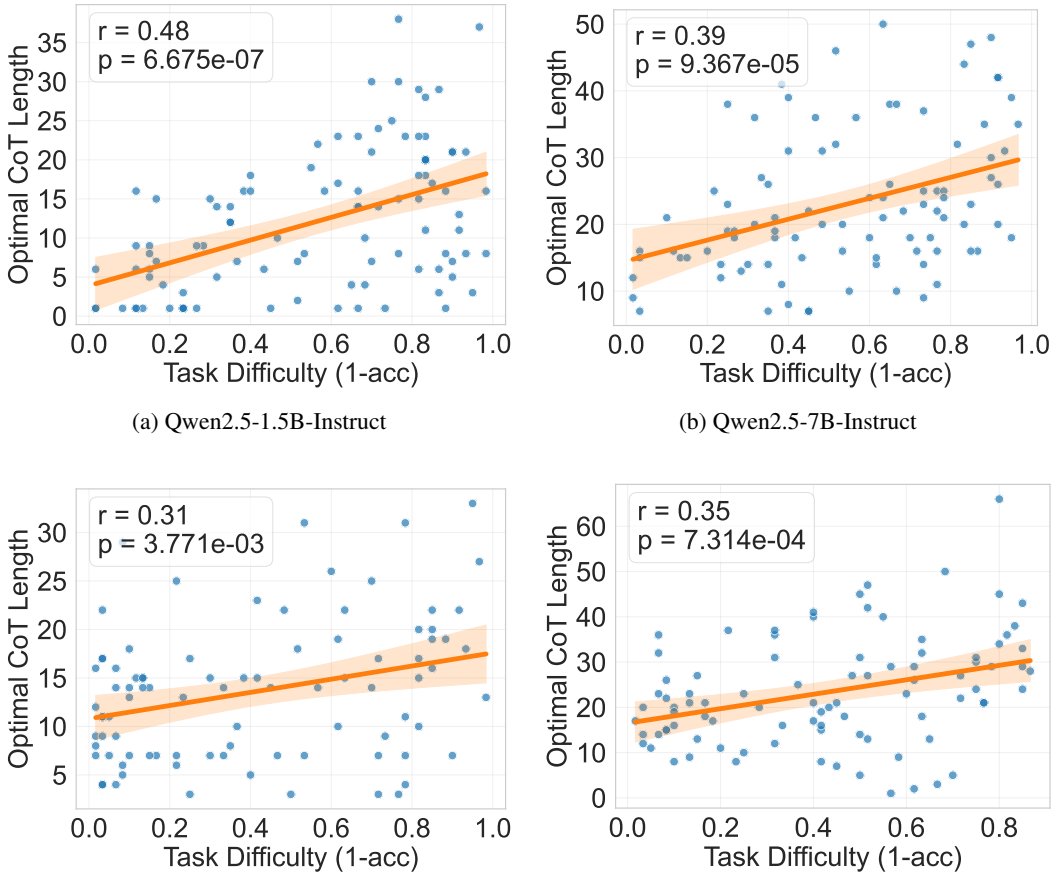
Results on MMLU STEM dataset. We also conduct experiments on the MMLU STEM dataset using the Qwen2.5 Series instruct models under the same settings as the MATH dataset. The results, shown in Figures 8 and 9, exhibit similar trends to those observed on the MATH dataset.



(a) Optimal CoT length vs. Model size (Qwen2.5 series).

(b) Optimal vs. Longest CoT length accuracy on MMLU STEM dataset.

Figure 8: Real-world CoT length observations. (a) Larger models tend to achieve optimal performance with shorter CoTs. (b) Accuracy for CoTs of optimal length is significantly higher than that of the longest CoTs.



(a) Qwen2.5-1.5B-Instruct

(b) Qwen2.5-7B-Instruct

(c) Qwen2.5-32B-Instruct

(d) Qwen2.5-72B-Instruct

Figure 9: Evaluation between task difficulties and optimal CoT lengths on MMLU STEM datasets with Qwen2.5 Series Instruct models.