TACLer: Tailored Curriculum Reinforcement Learning for Efficient Reasoning

Anonymous ACL submission

Abstract

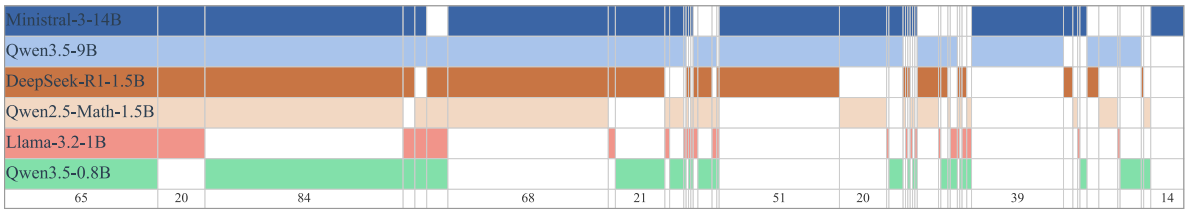
While Large Language Models (LLMs) have shown remarkable performance on complex reasoning tasks, eliciting long chains-of-thought (CoT) typically requires large-scale reinforcement learning (RL) training and often leads to overthinking with redundant intermediate steps. To improve learning and reasoning efficiency, we propose **TACLer**, a model-tailored curriculum reinforcement learning framework that gradually increases the complexity of the data based on the model's proficiency in multi-stage RL training. TACLer features two core components: (i) a tailored curriculum that identifies knowledge gaps and progressively schedules training data to balance knowledge acquisition and consolidation; (ii) a hybrid *Thinking/NoThinking* reasoning paradigm that flexibly allocates inference-time computation by enabling or disabling explicit thinking. Our experiments show that TACLer yields a twofold advantage in learning and reasoning: (i) it reduces computational cost, cutting training compute by over 50% compared to long thinking models and reducing inference token usage by over 42% relative to the base model; and (ii) it improves accuracy by over 9% on the base model, consistently outperforming state-of-the-art baselines across four math datasets with complex problems.[1]

1 Introduction

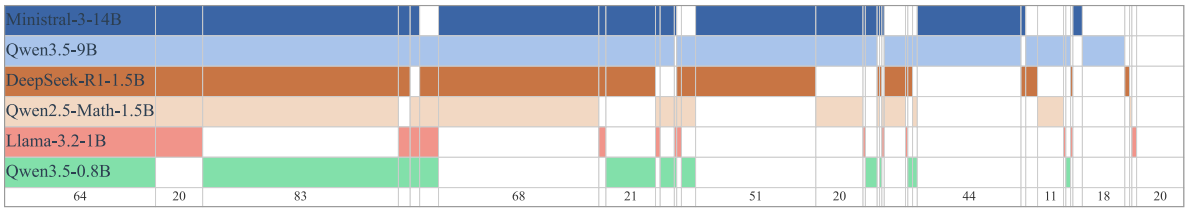
Many recent developments in Large Language Models (LLMs) have focused on improving their ability to solve problems involving complex reasoning, which has long been considered one of the most challenging tasks (Wei et al., 2022; Wang et al., 2023; Zhou et al., 2023; Yue et al., 2024). Particularly, DeepSeek-R1 (Guo et al., 2025) and OpenAI's o1 (OpenAI, 2025) show that long chain-of-thought (CoT, Wei et al., 2022) sequences can be used to boost reasoning capabilities through large-scale reinforcement learning (RL), playing a crucial role in solving complex problems. The long CoT responses often contain a thinking process with reflection, backtracking, and self-validation, which helps to form the solution and arrive at the final answer. This test-time scaling paradigm (Wu et al., 2024; Muennighoff et al., 2025) enhances reasoning abilities at the cost of significantly increased token usage. Consequently, this introduces two major bottlenecks: (i) training models with reinforcement learning over large contexts demands substantial computational resources, e.g., a small model with 1.5 billion parameters requires 70,000 A100 GPU hours (Luo et al., 2025b); and (ii) so-called "overthinking" can occur during response generation, leading to redundant reasoning steps and an even heavier computational cost (Chen et al., 2025b; Luo et al., 2025a; Arora and Zanette, 2025).

To improve training efficiency, recent studies have proposed approaches which gradually control response length, such as the iterative lengthening scheme (Luo et al., 2025b), or the difficulty of training instances as in curriculum learning (Song et al., 2025). To curb overthinking, reduction in token usage in the thinking process has been explored through techniques such as length-based rewards (Arora and Zanette, 2025; Shen et al., 2025), thinking pruning (Luo et al., 2025a), and adaptive thinking (Zhang et al., 2025; Tu et al., 2025). Nevertheless, these approaches often entail substantial training while only achieving limited reductions in response length, or may even degrade performance. Furthermore, models that rely on adaptive reasoning strategies constrain users' options since in some scenarios users might prefer to access long thinking traces even for relatively simple tasks, for purposes such as interpretability and verification. Therefore, enabling the coexistence of *Thinking* and *Nothinking* modes is particularly important, as it balances computational efficiency with model performance while offering users flexible user control.

[1] Code and model will be made available upon publication.



(a) Set of overlapping answers.



(b) Set of overlapping correct answers.

Figure 1: Answer overlap across models in MATH 500. The top diagram shows the intersection of answers for all questions, while the bottom focuses on cases where models agree on the correct answers.

To this end, we propose TACLer, a *tailored* curriculum reinforcement learning framework, aimed at improving both learning and reasoning efficiency. Unlike conventional curriculum learning that treats difficulty as a standardised concept, TACLer adapts the learning process to the model's evolving proficiency. Specifically, we gradually increase the data complexity based on the model's own proficiency, enabling efficient learning of challenging tasks that are difficult to learn from scratch. Building upon this adaptive learning process, TACLer further incorporates a hybrid *Thinking/NoThinking* paradigm to achieve efficient reasoning at inference time. This provides a unified perspective of *resource allocation*: the tailored curriculum allocates training effort toward unresolved knowledge gaps, while the hybrid paradigm controls the depth of thinking in inference.

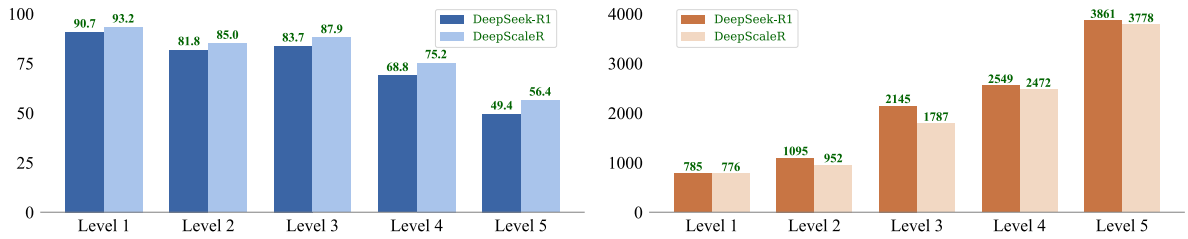
We show that TACLer reduces the required training compute by over 50% compared to other small long-thinking models, e.g., DeepScaleR-1.5B-Preview (Luo et al., 2025b). Particularly, our experimental results on four mathematical reasoning benchmarks (MATH 500, AMC, AIME 2024 and AIME 2025) show that TACLer achieves the best performance compared to various *Nothinking* and *Thinking* baselines, while significantly reducing the average number of token usage by over 42%, effectively mitigating the overthinking problem.

Our key contributions are as follows: (1) we introduce tailored curriculum learning, which determines what knowledge the model specifically lacks and needs to learn in progressive stages; (2) we incorporate both *Thinking* and *NoThinking* modes, showing that this can effectively balance reasoning accuracy and efficiency; (3) we propose TACLer, a novel RL-based framework that enables models to learn more efficiently through tailored curriculum learning and to reason effectively via hybrid reasoning modes; and (4) we report extensive experiments and analysis to validate the efficiency of TACLer on both learning and reasoning. While our experiments focus on mathematical reasoning, TACLer is potentially applicable to a wider range of structured multi-step reasoning problems.

2 Learning and Reasoning Efficiency

2.1 From Data Difficulty to Model-specific Curriculum

The computational burden of training language models is often enormous, especially for tasks involving long reasoning chains (Luo et al., 2025b; Song et al., 2025). A key inefficiency arises from the mismatch between data difficulty and model capability (Song et al., 2025). During training, a significant portion of samples may be overly difficult, leading to incomplete or low-quality responses. This not only wastes computational resources but also limits effective learning. Beyond this mismatch, we observe a more fundamental phenomenon: *model knowledge is heterogeneous and instance-dependent*. As shown in Figure 1, we find that while models tend to agree on certain problems, a substantial number of instances yield divergent answers, especially among lower-performing mod-



(a) Reasoning accuracy (%).

(b) Response length.

Figure 2: Comparison of DeepSeek-R1-Distill-Qwen-1.5B and DeepScaleR-1.5B-Preview using *NoThinking* mode across different difficulty levels of MATH 500 dataset.

els.[2] This suggests that model competence is not uniformly distributed, but instead reflects a distinct knowledge profile for each model.

Therefore, we introduce a shift from fixed to *adaptive* curriculum learning. Unlike approaches based on input length or human annotation, we treat learning as model-specific and dynamically adaptive to current performance. Specifically, training instances get organised according to whether they are solved, partially solved, or unsolved in a first inference pass, enabling a balance between knowledge consolidation and acquisition. This can progressively reduce a model's knowledge gaps, leading to more efficient training.

2.2 A Unified View of Curriculum and Reasoning Modes

While curriculum learning determines *what* the model learns during training, reasoning strategies govern *how* the model solves problems during inference. Recent advances in long thinking show that enabling detailed thinking can significantly improve performance, but often at the cost of increased token usage and "overthinking". To better understand whether improvements in reasoning ability persist beyond long thinking, we evaluate models in a *NoThinking* setting. Figure 2 compares DeepSeek-R1 and DeepScaleR under this configuration, where the latter is trained on top of the former in the *Thinking* mode to enhance long CoT reasoning. Interestingly, even in the NoThinking mode, DeepScaleR consistently outperforms DeepSeek-R1 across questions of varying difficulty, while also producing shorter responses.

Therefore, we argue that curriculum learning and reasoning modes are fundamentally connected through the lens of resource allocation. Specifically: (i) curriculum learning allocates training effort by selecting data that are most beneficial for learning; (ii) reasoning modes control the reasoning depth during inference, balancing accuracy and response length. Under this unified view, both mechanisms aim to improve efficiency at different stages of the pipeline: a reinforcement learning curriculum-based training improves how the model learns, which in turn can be expressed either through detailed thinking or concise responses during inference. In the following section, we introduce TACLer, which integrates tailored curriculum learning with a hybrid reasoning paradigm to achieve this goal.

3 Methodology

Figure 3 gives an overview of TACLer and its three components: (i) tailored curriculum learning; (ii) hybrid reasoning mode; and (iii) RL training.

3.1 Tailored Curriculum Learning

As discussed in Section 2, the imbalanced data distribution leads to a substantial portion of samples being overly difficult. Furthermore, model competence is unevenly distributed, with each model exhibiting a distinct knowledge profile (Figure 1). To address these challenges, we introduce a tailored curriculum learning, which progressively increases data complexity based on the model's performance.

To construct the curriculum, we perform inference over the training set and categorise instances into three groups based on the model's responses: (i) correct reasoning (CR), (ii) completed but incorrect reasoning (CIR), and (iii) incomplete reasoning (IR), as presented in Table 1. This categorisation aligns naturally with distinct learning signals: correctly solved instances reinforce existing knowledge, incorrect but complete responses highlight reasoning errors that can be refined, while truncated responses indicate excessive difficulty and are likely to result in inefficient training.

[2]Detailed results on answer overlap across varying difficulty levels are provided in Appendix A.4.

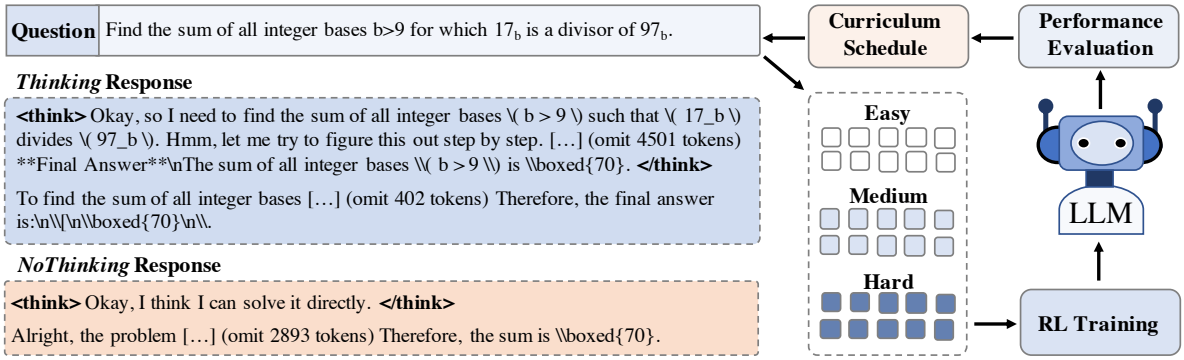


Figure 3: Overview of TACLer, our tailored curriculum reinforcement learning framework.

Category	Definition
CR	The model produces a correct final answer with a complete reasoning trace.
CIR	The model generates a complete trace and a final answer, but the final answer is incorrect.
IR	The reasoning trace is truncated or insufficient to reach a final answer.

Table 1: Categorisation of training instances.

Based on this categorisation, we design a three-stage curriculum. In Stage 1, we run inference on the training set and construct a subset by collecting problems from the CR and CIR categories.[3] The model is then trained on the curated subset, forming a combined *review and learning* paradigm that reinforces previously acquired knowledge while learning new skills.

In Stage 2, we repeat the same inference–selection–training pipeline, further refining the data distribution and the model's capabilities.

In Stage 3, we perform final training on the full training set using the model obtained after Stage 2, in order to consolidate the learned knowledge and stabilise performance.

3.2 Hybrid Reasoning Mode

We adopt a hybrid reasoning mode that integrates both *Thinking* and *NoThinking*.

Thinking In this mode, the model is prompted to generate a detailed reasoning process enclosed within `<thinking>` and `</thinking>`, and provide the corresponding solution and answer. This approach encourages the model to reflect, backtrack, and self-verify during the thinking process.

NoThinking This mode enforces a concise response, where the model generating the solution and answer without containing thinking steps. Instead of using "`Okay, I think I have finished thinking`" to induce the model to state that it has completed its reasoning (Ma et al., 2025), we explicitly prompt the model to solve the problem without thinking: `<thinking> Okay, I think I can solve it directly. </thinking>`.

For each training instance, we construct both *Thinking* and *NoThinking* variants (see examples in Figure 6), ensuring that every problem is learned under both reasoning modes. We employ this hybrid paradigm throughout all three curriculum learning stages, based on the intuition that the two modes can complement and reinforce each other. Specifically, a model trained solely in the *NoThinking* mode may compensate for missing explicit thinking by expanding its reasoning steps, reducing the risk of producing incorrect answers. In contrast, when trained jointly with the *Thinking* mode, the model can leverage a compression effect, where improved reasoning ability is distilled into shorter responses (see Sec. 4.3).

3.3 Training with GRPO

We adopt the Group Relative Policy Optimisation (GRPO, Shao et al., 2024) to train our model. For each problem q, the algorithm samples a group of responses $\{o_1, o_2, ..., o_G\}$ from the old policy π_{old}, and updates the policy π_θ by maximising the following objective:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{q\sim\mathcal{D},\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|q)} \tag{1}$$
$$\frac{1}{G}\sum_{i=1}^{G}\min\left(\frac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{\text{old}}}(o_i \mid q)}A_i,\right.$$
$$\left.\text{clip}\left(\frac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{\text{old}}}(o_i \mid q)}, 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}}\right)A_i\right),$$

where ϵ_{low} and ϵ_{high} are hyperparameters for the

[3] See Appendix A.1 for details on setting and computational cost.

	MATH 500		AIME 2024		AMC		AIME 2025		Average	
	Acc	Len ↓	Acc	Len ↓	Acc	Len ↓	Acc	Len ↓	ΔAcc	ΔLen ↓
R1-Qwen$_{Thinking}$	81.2	4856	27.7	12306	60.8	8754	21.5	12182	-	-
STILL-3	83.6	3797	30.4	10605	66.3	7091	24.4	10415	+3.4	-17.3%
DeepScaleR	87.8	3030	40.4	8565	73.8	5616	**31.3**	8239	+10.5	-34.1%
FastCuRL	87.8	3894	39.8	10091	73.9	6756	27.9	9723	+9.6	-20.2%
TACLer$_{Thinking}$	**88.4**	**3010**	**42.1**	**6868**	**74.6**	**4871**	30.8	**6807**	**+11.2**	**-42.7%**

Table 2: Model performance in long thinking mode across four math reasoning datasets. Acc: accuracy; Len: response length (tokens). Bold numbers indicate the best result for each dataset.

clipping range of the importance sampling ratio. A_i is the advantage, caculated by the rewards $\{r_1, r_2, ..., r_G\}$ of responses in the same group:

$$A_i = \frac{r_i - \text{mean}(\{r_i\}_{i=1}^G)}{\text{std}(\{r_i\}_{i=1}^G)}. \quad (2)$$

Following prior work (Guo et al., 2025), we employ a rule-based reward scheme, assigning a binary reward of 1 to correct final answers and 0 otherwise.

4 Experiments

4.1 Experimental Setup

Model and Dataset We select DeepSeek-R1-Distill-Qwen-1.5B (R1-Qwen, Guo et al., 2025) as our backbone, given its proven strong performance on mathematical tasks. We train TACLer on DeepScaleR-Dataset (Luo et al., 2025b), and perform evaluation on four math benchmarks: MATH 500 (Lightman et al., 2024), AMC, AIME 2024, and AIME 2025.[4]

Baselines We compare TACLer with 10 state-of-the-art 1.5B baselines, including long and efficient reasoning.[5]

4.2 Main Results

Reasoning Accuracy and Efficiency in Thinking Mode Table 2 compares TACLer in *Thinking* mode against several long thinking models. Across four datasets, TACLer achieves the highest accuracy on three of them, and performs slightly lower than DeepScaleR but higher than other models on AIME 2025. Overall, it achieves the **highest average accuracy**, with an average +11.2% improvement over the base model R1-Qwen. In addition, **TACLer reduces reasoning length by 42.7%, substantially outperforming all baselines** (e.g., -34.1% for DeepScaleR and -20.2% for FastCuRL), suggesting that the hybrid mode can leverage the compression effect to refine reasoning into concise responses. These results highlight that **TACLer in *Thinking* mode provides a more effective balance between reasoning accuracy and response length than existing methods**.

	MATH 500		AIME 2024		Average	
	Acc	Len	Acc	Len	ΔAcc	ΔLen
R1-Qwen$_{Thinking}$	81.2	4856	27.7	12306	-	-
R1-Qwen$_{NoThinking}$	67.8	1069	14.8	4689	-13.2	-69.9%
OverThink	81.2	4131	28.3	11269	+0.3	-11.7%
DAST	83.0	2428	26.9	7745	+0.5	-43.5%
O1-Pruner	82.2	3212	28.9	10361	+1.1	-24.8%
TLMRE	85.0	3007	29.2	8982	+2.6	-32.5%
ModelMerging	63.0	2723	18.1	10337	-13.9	-30.0%
AdaptThink	86.0	2511	34.8	9279	+5.9	-36.4%
AutoThink	83.8	**2128**	31.7	8167	+3.3	-44.9%
TACLer$_{NoThinking}$	**88.2**	2532	**39.6**	**6056**	**+9.5**	**-49.3%**
	AMC		AIME 2025		Average	
	Acc	Len ↓	Acc	Len ↓	ΔAcc	ΔLen ↓
R1-Qwen$_{Thinking}$	60.8	8754	21.5	12182	-	-
R1-Qwen$_{NoThinking}$	48.6	2264	13.3	4062	-10.2	-70.4%
AdaptThink	67.4	5489	25.6	9117	+5.4	-31.2%
AutoThink	66.7	4596	23.8	7647	+4.1	-42.4%
TACLer$_{NoThinking}$	**72.7**	**4312**	**27.9**	**5710**	**+9.2**	**-51.9%**

Table 3: Model performance in efficient reasoning mode across four math datasets. For MATH 500 and AIME 2024, results for all baselines except AutoThink are taken from Zhang et al. (2025). Acc: accuracy; Len: response length (tokens). Bold numbers indicate the best result for each dataset.

Reasoning Accuracy and Efficiency in Saving Mode Table 3 compares TACLer in *NoThinking* mode with recent efficient reasoning approaches. The first observation is that TACLer consistently achieves the best overall performance, with the only exception being response length on the MATH 500 dataset. Specifically, R1-Qwen initially performs poorly in *NoThinking* mode, and during training, it tends to compensate for the lack of explicit reasoning by slightly increasing its token usage. After training, TACLer in *NoThinking* mode substantially outperforms R1-Qwen in *Thinking* mode, achieving over 9% higher accuracy while using roughly

[4]See Appendix A.1 for details on training and evaluation.
[5]See Appendix A.2 for more details on baselines.

	MATH 500		AIME 2024		AMC		AIME 2025		Average	
	Acc	Len ↓	Acc	Len ↓	Acc	Len ↓	Acc	Len ↓	ΔAcc	ΔLen ↓
R1-Qwen$_{Thinking}$	81.2	4856	27.7	12306	60.8	8754	21.5	12182	-	-
NoThinking Mode										
Direct-Train	87.8	**2217**	38.1	**5974**	72.1	**4093**	26.9	**5550**	+8.4	**-53.4%**
TACLer	**88.2**	2532	**39.6**	6056	**72.7**	4312	**27.9**	5710	**+9.3**	-50.6%
Thinking Mode										
Direct-Train	88.0	**2962**	37.5	7085	73.0	4920	27.5	**6736**	+8.7	-42.5%
TACLer	**88.4**	3010	**42.1**	**6868**	**74.6**	**4871**	**30.8**	6807	**+11.2**	**-42.7%**

Table 4: Comparison of curriculum learning and direct training (training from the beginning with all data at once). Acc: accuracy; Len: response length (tokens). Bold numbers indicate the best result for each dataset.

	MATH 500		AIME 2024		AMC		AIME 2025		Average	
	Acc	Len ↓	Acc	Len ↓	Acc	Len ↓	Acc	Len ↓	ΔAcc	ΔLen ↓
R1-Qwen$_{Thinking}$	81.2	4856	27.7	12306	60.8	8754	21.5	12182	-	-
Pure-NoThinking	84.0	2325	**32.1**	8380	65.1	5223	24.8	8220	+3.7	-39.2%
TACLer$_{NoThinking}$	82.0	**1472**	25.6	**5882**	62.1	**3479**	22.5	**5455**	+0.3	**-59.3%**
TACLer$_{Thinking}$	**84.8**	3287	30.0	9168	**68.2**	5875	**25.2**	8873	**+4.3**	-29.5%

Table 5: Comparison between hybrid reasoning and pure *NoThinking* in the first curriculum learning stage. Acc: accuracy; Len: response length (tokens). Bold numbers indicate the best result for each dataset.

half the tokens (49.3% and 51.9%).[6] In comparison, efficient reasoning baselines such as AutoThink improve efficiency by reducing token usage, but offer smaller accuracy gains; ModelMerging even suffers from substantial performance degradation. **On average, across the four datasets, TACLer achieves higher reasoning accuracy and generates shorter responses**.

4.3 Analysis and Discussion

Curriculum Learning Yields More Efficient and Accurate Reasoning We compare TACLer with Direct-Train, which is trained directly using all of the data at once rather than staggered based on difficulty, with all other settings staying the same (Table 4). While Direct-Train tends to generate shorter responses than TACLer under the *NoThinking* mode, this trend reverses in the *Thinking* mode, where TACLer produces more concise outputs than Direct-Train. Moreover, TACLer consistently achieves a better performance across both modes in comparison to the model trained without staggered data, showing an effective balance between conciseness and accuracy.

Hybrid Mode Yields Concise Reasoning We compare TACLer with a model trained solely in the *NoThinking* mode, with both models trained only in the first curriculum learning stage, as shown in Table 5. TACLer achieves the highest accuracy in the *Thinking* mode, while generating the shortest responses in the *NoThinking* mode. In contrast, Pure-NoThinking achieves higher accuracy than TACLer in the *NoThinking* mode but produces substantially longer responses, failing to support concise reasoning. These results show that the hybrid reasoning mode allows the model to benefit from both strategies, making reasoning under *NoThinking* more concise with no heavy loss in performance.

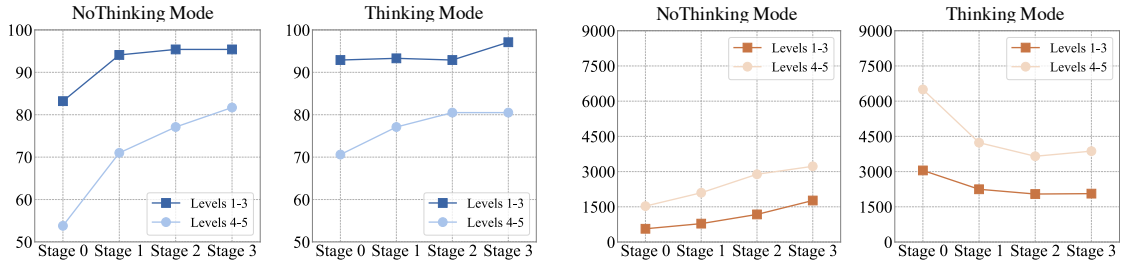
[6]See appendix A.5 for example responses.

	MATH 500	AIME 2024	AMC	AIME 2025	Avg.
Oracle Union	91.4	52.1	80.8	33.8	-
NoThinking	88.2	39.6	72.7	27.9	86.3
Thinking	88.4	42.1	74.6	30.8	90.2

Table 6: Comparison of TACLer to the Oracle Union of two reasoning modes. Avg. denotes the average ratio of TACLer's accuracy to the Oracle Union accuracy across all four datasets.

Theoretical Performance Ceiling We introduce an "oracle union" as a reference to estimate the theoretical performance ceiling of our hybrid reasoning paradigm, where a prediction is considered correct if either of the two modes produces the correct answer. As shown in Table 6, the oracle union represents the best achievable accuracy if we could perfectly combine the strengths of both modes. Across four datasets, TACLer achieves over 86% of this upper bound on average.

TACLer's Performance at Different Stages Figure 4 shows reasoning accuracy and response



(a) Reasoning accuracy. (b) Response length.

Figure 4: Comparison of training stages on different difficulty levels of MATH 500 (stage 0 = base model).

length of TACLer across different training stages. For reasoning accuracy, we observe a consistent upward trend across the three training stages. The largest improvement occurs in the first stage, with higher gains in the *NoThinking* mode. Notably, in the third stage TACLer in *Thinking* mode obtains higher scores on easier problems (levels 1-3). We hypothesise that some questions categorised as "easy" by humans may actually be more challenging for the model, and can be successfully solved after the model's capabilities are substantially enhanced through curriculum learning (see examples in Figure 13.) When looking at response length, TACLer shows a different trend for *Nothinking* and *Thinking* modes: in the former the length of responses gradually increases with training, while in the latter it gradually decreases. This is consistent with our hypothesis: *NoThinking* mode tends to compensate for the lack of explicit reasoning by slightly increasing its token usage, while *Thinking* mode leverages the compression effect to refine reasoning into concise responses.

Computation Cost and Learning Efficiency Table 7 summarises the training configurations for different long thinking models for comparing their computational costs. DeepScaleR and FastCuRL employ progressive training strategies that increase both the context length (from 8K to 24K) and the number of rollouts (from 8 to 16) across training stages. In contrast, TACLer achieves superior performance using a fixed 8K context and a fixed rollout number of 8. This suggests that the gains primarily arise from more efficient learning rather than long-context scaling. Regarding computational cost, Luo et al. (2025b) indicate that doubling the context window size increases training computation by at least 2 times. In our experiments, we observe that doubling the number of rollouts roughly increase the computation by about 1 time, indicating that TACLer reduces the required

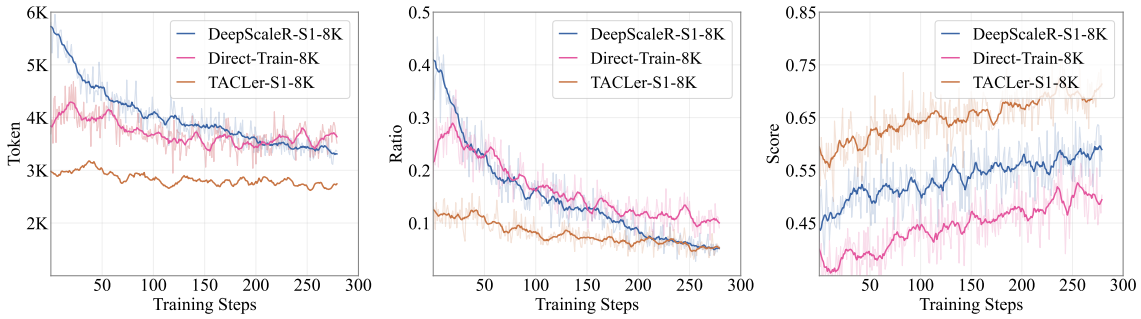
Stage	Context	Batch Size	Step	Rollout
DeepScaleR-1.5B-Preview				
Stage 1	8K	128	1040	8
Stage 2	16K	128	500	16
Stage 3	24K	128	210	16
FastCuRL-1.5B-Preview				
Stage 1	8K	128	160	8
Stage 2	16K	64	590	8
Stage 3	24K	64	230	8
Stage 4	16K	64	580	16
TACLer				
Stage 1	8K	128	280	8
Stage 2	8K	128	350	8
Stage 3	8K	128	1250	8

Table 7: Training configuration comparison.

training compute by over 50% compared to those models.[7] To validate learning efficiency, in Figure 5 we present the response length, clip ratio, and reward for TACLer and Direct-Train in the first training stage, and compare them with DeepScaleR. We observe that TACLer generates the shortest responses, indicating faster and more efficient reasoning. This also leads to a lower response clipping rate, which contributes to more efficient training. In addition, TACLer consistently achieves higher rewards throughout training. These observations highlight TACLer's efficient learning abilities.

Data Complexity in Curriculum Learning The model resulting after each training stage is used to curate the difficulty-based dataset to be used in the next stage(s) of tailored curriculum learning. Table 8 summarises the data composition across the three iterative stages, each consisting of three sets of data of varying difficulty, based on the results of the model R1-Qwen. As training progresses, the proportion of harder data increases from 0% in the first stage to 34.6% in the second stage and 55.1%

[7] As shown in Appendix A.1, the additional cost of running the tailored curriculum labelling is relatively small compared to the training cost.



(a) Response length. (b) Response clip ratio. (c) Response reward.

Figure 5: Comparison of TACLer, Direct-Train, and DeepScaleR in terms of response clip ratio, length, and reward.

	CR (%)	CIR (%)	IR (%)	Total (#)
Stage 1	78.1	21.9	0	18110
Stage 2	51.1	14.3	34.6	27692
Stage 3	35.1	9.8	55.1	40315

Table 8: Data complexity statistics across three training stages (see Table 1 for details on CR, CIR, and IR).

in the third stage, while the proportion of easier data decreases accordingly.

5 Related Work

Curriculum Learning for LLM Reasoning Recently, curriculum learning has been adopted in LLM reasoning through two main strategies: (i) progressively scheduling training data from easy to hard using criteria such as input length (Song et al., 2025), human annotation (Parashar et al., 2025), and model assessments (Parashar et al., 2025; Ji et al., 2025); and (ii) designing curriculum rewards for RL training (Chen et al., 2025a). In this work, A **key innovation** of our work relative to previous work is the introduction of a **model-tailored curriculum learning** framework which does not take an absolute concept of difficulty. Rather, **it determines what knowledge a model lacks and needs to learn in progressive stages, thereby facilitating more efficient and effective training**.[8]

Reinforcement Learning for LLM Reasoning LLMs have shown impressive abilities in addressing complex reasoning tasks, leveraging CoT reasoning (Wei et al., 2022), reasoning consistency (Wang et al., 2023; Zhou et al., 2024; Lai et al., 2025; Zhang et al., 2026) and multilingual reasoning (Shi et al., 2023; Lai and Nissim, 2024; Chai et al., 2025). Recent works such as DeepSeek R1 (Guo et al., 2025) show that large-scale RL can effectively elicit long CoT, leading to substantial gains on various challenging tasks. However, long CoT might induce "overthinking", resulting in overly verbose reasoning and increased computational overhead (Sui et al., 2025). To address this problem, a line of work has emerged on efficient reasoning strategies, such as adaptive thinking (Zhang et al., 2025; Tu et al., 2025), length-based rewards (Arora and Zanette, 2025; Shen et al., 2025), and disabling the thinking process (Ma et al., 2025). In this work, we propose a model-tailored curriculum reinforcement learning framework, showing that the coexistence of *Thinking* and *NoThinking* modes can be mutually beneficial.

6 Conclusion

TACLer is **a novel model-tailored curriculum reinforcement learning framework for efficient reasoning**. Leveraging staggered training based on progressive data complexity, TACLer enables the model to learn more efficiently and also effectively. Moreover, by adopting a hybrid reasoning mode, TACLer successfully balances accuracy and efficiency. Our experiments on four math reasoning benchmarks show that **TACLer reduces training compute by over 50% and inference token usage by over 42% while improving accuracy by over 9%**. TACLer introduces a paradigm shift in how we approach the development of (reasoning) models, where difficulty of training instances is conceived as model-dependent rather than intrinsic to data. By moving away from a standardised view of data difficulty and instead adopting a model-tailored curriculum which identifies and leverages each model's "knowledge gap", models learn effectively as training is aligned with their evolving proficiency.

[8] A direct comparison with the models proposed in (Parashar et al., 2025; Ji et al., 2025; Chen et al., 2025a) is not possible due to the lack of publicly available checkpoints; therefore, these models are not included in Tables 2 and 3.

Limitations

While TACLer demonstrates significant gains in efficiency and performance, several limitations remain to be addressed in future work:

- **Task Generalisation**: Our experiments mainly focuses on math reasoning. Future work can extend our framework to other reasoning tasks or even to general-purpose scenarios.
- **Model-tailored Curriculum**: While we do show that proficiency indeed differs across models (Section 2 and Appendix A.4) warranting a model-tailored difficulty conceptualisation, due to limitations of computing and space, we have not yet conducted cross-model curriculum experiments where, e.g., a student model is trained using a curriculum tailored to a different model. Such studies would make it possible to better quantify the impact of model-tailored curriculum learning.
- **Reasoning Mode**: We implement reasoning mode as a user-driven choice, thereby providing options in the way inference is performed. While adaptive strategies (switching between thinking and no-thinking automatically) might improve efficiency, they may not be optimal for users who might on the one hand require long thinking traces, for example in education or verification, or on the other might want to use a minimal budget overall. An ideal system might offer a seamless toggle between adaptive thinking and manual mode selection; we view this kind of implementation as a product-side design issue.

References

Daman Arora and Andrea Zanette. 2025. Training language models to reason efficiently. *arXiv preprint arXiv:2502.04463*.

Linzheng Chai, Jian Yang, Tao Sun, Hongcheng Guo, Jiaheng Liu, Bing Wang, Xinnian Liang, Jiaqi Bai, Tongliang Li, Qiyao Peng, and Zhoujun Li. 2025. xcot: cross-lingual instruction timing for cross-lingual chain-of-thought reasoning. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*.

Xiaoyin Chen, Jiarui Lu, Minsu Kim, Dinghuai Zhang, Jian Tang, Alexandre Piché, Nicolas Gontier, Yoshua Bengio, and Ehsan Kamalloo. 2025a. Self-evolving curriculum for llm reasoning. *arXiv preprint arXiv:2505.14970*.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025b. Do NOT think that much for 2+3=? on the overthinking of long reasoning models. In *Forty-second International Conference on Machine Learning*.

Zhipeng Chen, Yingqian Min, Beichen Zhang, Jie Chen, Jinhao Jiang, Daixuan Cheng, Wayne Xin Zhao, Zheng Liu, Xu Miao, Yang Lu, Lei Fang, Zhongyuan Wang, and Ji-Rong Wen. 2025c. An empirical study on eliciting and improving r1-like reasoning models. *arXiv preprint arXiv:2503.04548*.

Daya Guo, Dejian Yang, Haowei Zhang, and et al. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, page 633–638.

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. 2025. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*.

Yunjie Ji, Sitong Zhao, Xiaoyu Tian, Haotian Wang, Shuaiting Chen, Yiping Peng, Han Zhao, and Xiangang Li. 2025. How difficulty-aware staged reinforcement learning enhances llms' reasoning capabilities: A preliminary experimental study. *arXiv preprint arXiv:2504.00829*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, page 611–626.

Huiyuan Lai and Malvina Nissim. 2024. mCoT: Multilingual instruction tuning for reasoning consistency in language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12012–12026.

Huiyuan Lai, Xiao Zhang, and Malvina Nissim. 2025. Multidimensional consistency improves reasoning in language models. *arXiv preprint arXiv:2503.02670*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. 2025a. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning. *arXiv preprint arXiv:2501.12570*.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. 2025b. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. Notion Blog.

Wenjie Ma, Jingxuan He, Charlie Snell, Tyler Griggs, Sewon Min, and Matei Zaharia. 2025. Reasoning models can be effective without thinking. *arXiv preprint arXiv:2504.09858*.

Yu Meng, Mengzhou Xia, and Danqi Chen. 2024. SimPO: Simple preference optimization with a reference-free reward. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

OpenAI. 2025. Competitive programming with large reasoning models. *arXiv preprint arXiv:2502.06807*.

Shubham Parashar, Shurui Gui, Xiner Li, Hongyi Ling, Sushil Vemuri, Blake Olson, Eric Li, Yu Zhang, James Caverlee, Dileep Kalathil, and Shuiwang Ji. 2025. Curriculum reinforcement learning from easy to hard tasks improves llm reasoning. *arXiv preprint arXiv:2506.06632*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Yi Shen, Jian Zhang, Jieyun Huang, Shuming Shi, Wenjing Zhang, Jiangze Yan, Ning Wang, Kai Wang, Zhaoxiang Liu, and Shiguo Lian. 2025. Dast: Difficulty-adaptive slow-thinking for large reasoning models. *arXiv preprint arXiv:2503.04472*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, page 1279–1297.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. 2023. Language models are multilingual chain-of-thought reasoners. In *The Eleventh International Conference on Learning Representations*.

Mingyang Song, Mao Zheng, Zheng Li, Wenjie Yang, Xuan Luo, Yue Pan, and Feng Zhang. 2025. Fastcurl: Curriculum reinforcement learning with progressive context extension for efficient training r1-like reasoning models. *arXiv preprint arXiv:2503.17287*.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Hanjie Chen, and Xia Hu. 2025. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*.

Songjun Tu, Jiahao Lin, Qichao Zhang, Xiangyu Tian, Linjing Li, Xiangyuan Lan, and Dongbin Zhao. 2025. Learning when to think: Shaping adaptive reasoning in r1-style models via multi-stage rl. *arXiv preprint arXiv:2505.10832*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Han Wu, Yuxuan Yao, Shuqi Liu, Zehua Liu, Xiaojin Fu, Xiongwei Han, Xing Li, Hui-Ling Zhen, Tao Zhong, and Mingxuan Yuan. 2025. Unlocking efficient long-to-short llm reasoning with model merging. *arXiv preprint arXiv:2503.20641*.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024. Inference scaling laws: An empirical analysis of compute-optimal inference for llm problem-solving. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, and 16 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2024. MAmmoTH: Building math generalist models through hybrid instruction tuning. In *The Twelfth International Conference on Learning Representations*.

Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. 2025. Adaptthink: Reasoning models can learn when to think. *arXiv preprint arXiv:2505.13417*.

Xiao Zhang, Qianru Meng, Yongjian Chen, Yumeng Wang, and Johan Bos. 2026. Knowledgeberg: Evaluating systematic knowledge coverage and compositional reasoning in large language models. In *Findings of the Association for Computational Linguistics: ACL 2026*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

Yue Zhou, Yada Zhu, Diego Antognini, Yoon Kim, and Yang Zhang. 2024. Paraphrase and solve: Exploring and exploiting the impact of surface form on mathematical reasoning in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 2793–2804.

A Appendix

A.1 Training and Evaluation Setting

Figure 6 illustrates example prompts used for the *Thinking* and *NoThinking* modes in our framework.

We incorporate several enhancements from recent research into original GRPO, including (1) removing the KL loss to effectively unlock the full potential of the policy model without affecting training stability (Hu et al., 2025); and (2) increasing the upper clip bounds ϵ_{high} in Eq. 1 while fixing the lower clip bounds ϵ_{low} to mitigate the entropy convergence problem and encourage the policy to explore diverse solutions (Yu et al., 2025).

We train TACLer based on VeRL (Sheng et al., 2025) and perform inference using vLLM (Kwon et al., 2023). Across three training stages, we set the context length to 8K, the batch size to 128, and the learning rate to 1e-6. The hyperparameters ϵ_{low} and ϵ_{high} are set to 0.2 and 0.28, respectively. TACLer is trained on 4 NVIDIA H100 GPUs (94 GB memory each) for approximately 96 hours.

Regarding the inference in curriculum labelling, we use greedy decoding and set a context size to 8k. Processing the full dataset takes approximately 5 hours on a single H100 GPU. After the first and second training stages, this time is further reduced due to the substantial improvement in TACLer's reasoning efficiency. Overall, the incurred overhead is relatively small compared to the overall training.

We report both the average reasoning accuracy and the response length based on the number of tokens in our evaluation. Following previous work (Zhang et al., 2025) we adopt the sampling method with a context size of 16K, a temperature of 0.6 and a top-p value of 0.95 to generate the response for each question. We sample 16 responses for each question in AMC, AIME 2024 and AIME 2025, and report the average results due to their limited dataset size.

A.2 Baselines

We compare TACLer with state-of-the-art 1.5B baselines on efficient math reasoning.

- **OverThink** (Chen et al., 2025b): a SimPO (Meng et al., 2024) based method, taking the longest response as the negative example and the first two correct reasoning as the positive ones.
- **DAST** (Shen et al., 2025): a SimPO-based method, where preference data is ranked by pre-sampled responses using a length-based reward function.
- **O1-Pruner** (Luo et al., 2025a): it builds a baseline via pre-sampling and guides the model to generate more concise reasoning under accuracy constraints.
- **TLMRE** (Arora and Zanette, 2025): a method that incorporates a length-based penalty in on-policy RL training.
- **ModelMerging** (Wu et al., 2025): it takes a weighted average of the weights of the reasoning model and the non-reaosning model.
- **AdaptThink** (Zhang et al., 2025): a method trained with a constrained objective to encourage the model to reason in the NoThinking way.
- **AutoThink** (Tu et al., 2025): an RL-based approach optimizes models to invoke long inferences only when necessary.

All above methods use DeepScaleR dataset, where methods (1)-(5) are reproduced by Zhang et al. (2025), thereby having a fair comparison. Additionally, we compare our method with several 1.5B baselines trained for long CoT reasoning in the *Thinking* mode.

- **STILL-3** (Chen et al., 2025c): a method to enhance the model's slow thinking through RL training.
- **DeepScaleR** (Luo et al., 2025b): an iterative training method that iteratively increases the context length from 8K to 24K, allowing the model to improve the quality of long CoT.
- **FastCuRL** (Song et al., 2025): a curriculum RL framework with progressively extended context, in which the training data complexity is categorised based on input length.

A.3 Contrasting Response Lengths in Correct and Incorrect Reasoning

Figure 7 presents the average response length for correct and incorrect answers at the problem level in two reasoning modes. Regarding answer correctness, we find that in both reasoning modes, correct

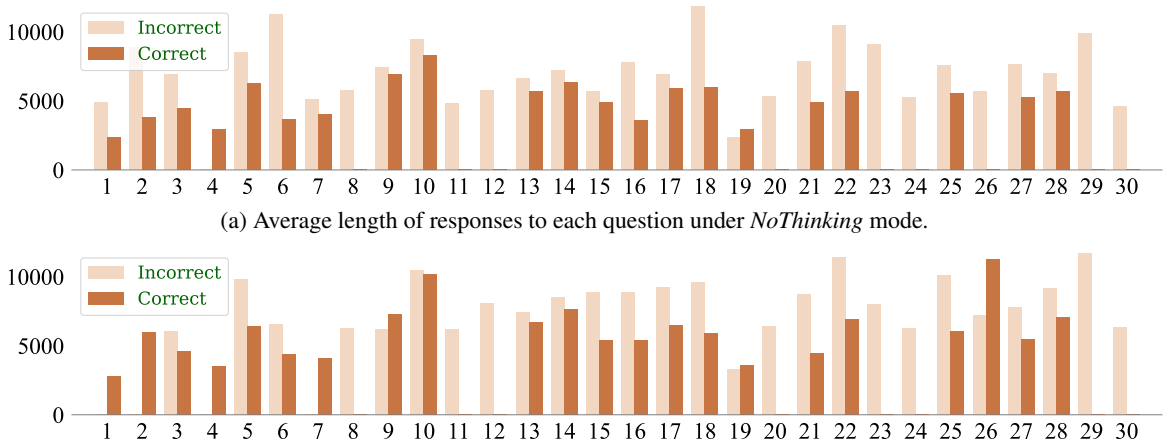
Thinking Mode

< | begin_of_sentence | >< | User | >
Find the sum of all integer bases b>9 for which 17_b is a divisor of 97_b.
Let's think step by step and output the final answer within \\boxed{}.
< | Assistant | ><think>

NoThinking Mode

< | begin_of_sentence | >< | User | >
Find the sum of all integer bases b>9 for which 17_b is a divisor of 97_b.
Let's think step by step and output the final answer within \\boxed{}.
< | Assistant | ><think>
Okay, I think I can solve it directly.
</think>

Figure 6: Example prompts for *Thinking* and *NoThinking* modes.



(a) Average length of responses to each question under *NoThinking* mode.

(b) Average length of responses to each question under *Thinking* mode.

Figure 7: Comparison of average response length between correct and incorrect answers. For each problem, we generate 16 responses and compute the average length separately for correct and incorrect responses; when all responses are of one type (correct or incorrect), the average length for the opposite type is set to 0.

responses are shorter than incorrect ones, as the latter often contain more verbose or repetitive content, an observation consistent with previous work (Song et al., 2025). When comparing the responses between the two reasoning modes, correct answers in the *Nothinking* mode tend to have shorter responses than those in the *Thinking* mode, whereas incorrect answers show no clear length trend between the two modes.

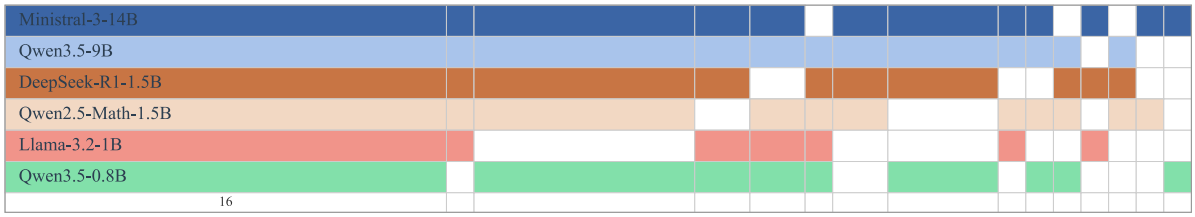
A.4 Answer Overlap Across Difficulty Levels

Figures 8–12 present the answer overlap across models for MATH 500 problems with varying levels of difficulty. We observe that for problems with lower difficulty levels, the models' answers tend to be highly consistent, as most questions can be answered correctly by the majority of models. As the difficulty level increases, the proportion of divergent answers gradually grows, particularly among lower-performing models. This suggests that model competence is not uniformly distributed, but instead reflects a distinct knowledge profile for each model.

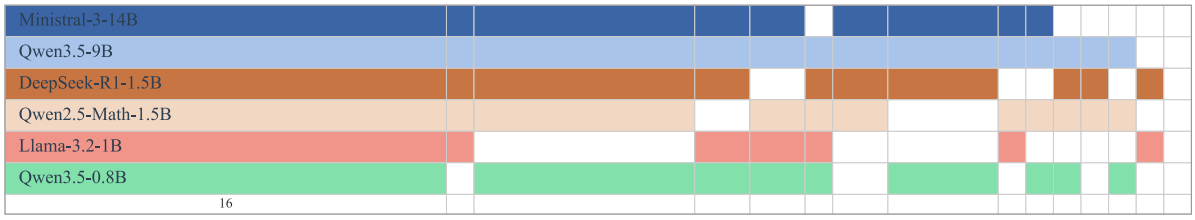
A.5 Case Study

Figure 13 presents a level-2 problem from the MATH 500 dataset, which is considered relatively easy by humans but proves more challenging for language models. The response generated by the base model R1-Qwen under the *Thinking* mode produces an incorrect answer, while TACLer answers correctly with fewer tokens.

Figures 14 and 15 present a math problem at difficulty level 3, comparing outputs of the base model R1-Qwen and our model TACLer under two reasoning modes: *Thinking* and *NoThinking*. For R1-Qwen, the output under the *NoThinking* mode is short but the final answer is incorrect, whereas the *Thinking* mode produces a correct answer at the cost of excessively long outputs. In contrast, TACLer produces shorter outputs with correct final answers in both modes, with the output under the *NoThinking* mode being about half the length of that under the *Thinking* mode. We attribute this

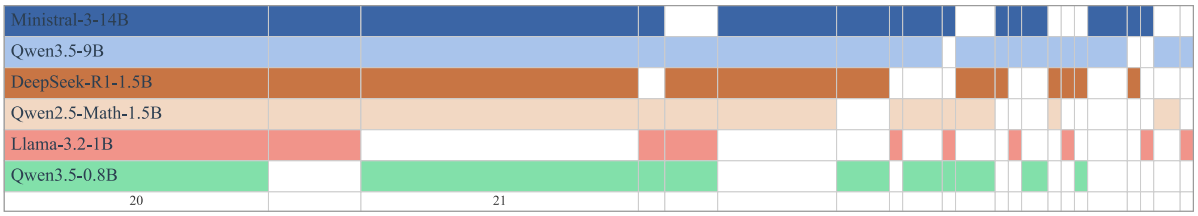


(a) Set of overlapping answers.

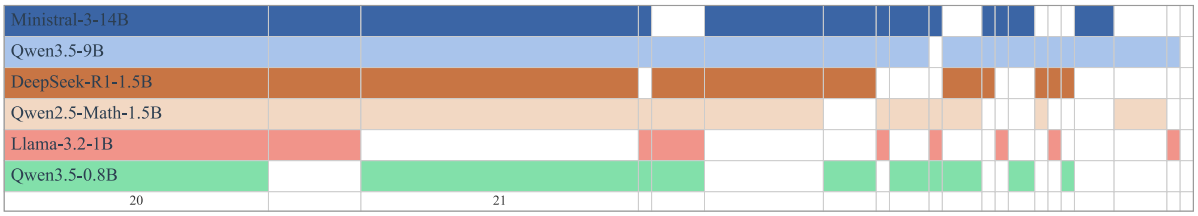


(b) Set of overlapping correct answers.

Figure 8: Answer overlap across models on difficulty level 1 problems in MATH 500.



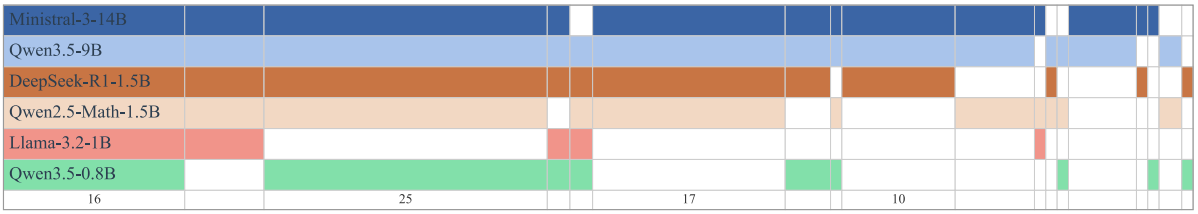
(a) Set of overlapping answers.



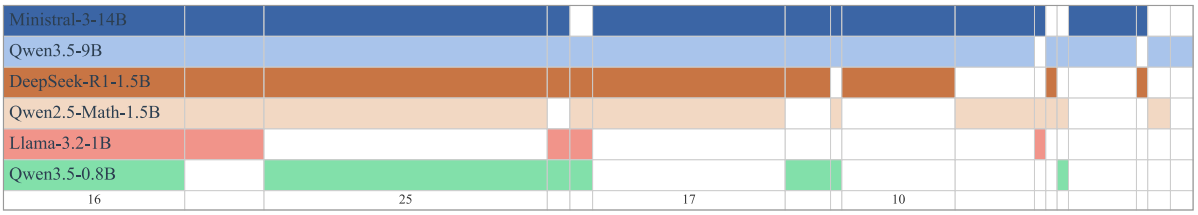
(b) Set of overlapping correct answers.

Figure 9: Answer overlap across models on difficulty level 2 problems in MATH 500.

behavior to the following reasons: (i) for *NoThinking* mode, TACLer compensates for missing explicit thinking by expanding its reasoning steps, thus resulting in a slight increase in output length; (ii) for *Thinking* mode, TACLer leverage a compression effect that refines its reasoning into more concise responses. Overall, these cases highlight distinct differences in reasoning behaviour and performance between the two models across different reasoning modes.

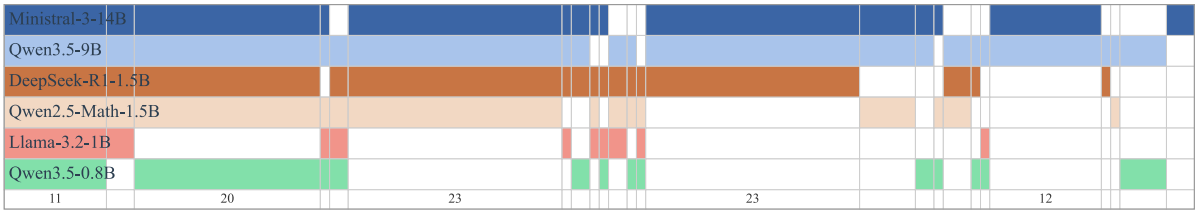


(a) Set of overlapping answers.

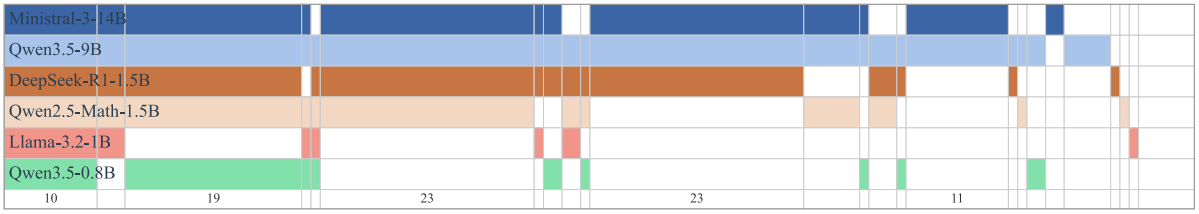


(b) Set of overlapping correct answers.

Figure 10: Answer overlap across models on difficulty level 3 problems in MATH 500.

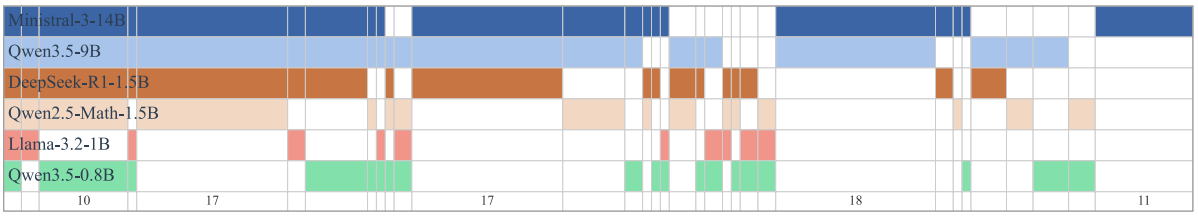


(a) Set of overlapping answers.

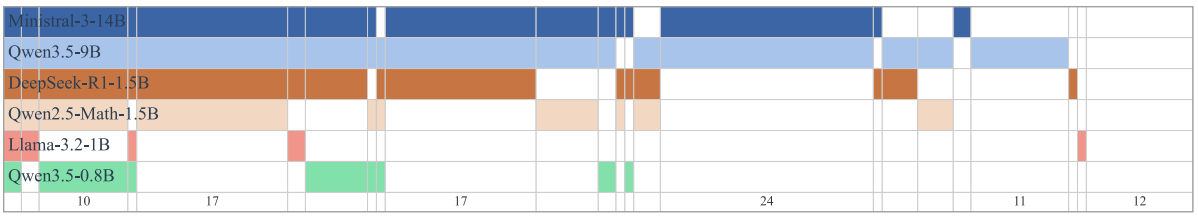


(b) Set of overlapping correct answers.

Figure 11: Answer overlap across models on difficulty level 4 problems in MATH 500.



(a) Set of overlapping answers.



(b) Set of overlapping correct answers.

Figure 12: Answer overlap across models on difficulty level 5 problems in MATH 500.

Question (MATH500): Evaluate\n\\[\\sin (\\arcsin 0.4 + \\arcsin 0.5) \\cdot \\sin (\\arcsin 0.5 - \\arcsin 0.4).\\]
Difficulty Level: 2
Answer: \\frac{9}{100}

DeepSeek-R1-Distill-Qwen-1.5B (*Thinking* Mode): ✗
<think>\n Okay, so I have this problem here: I need to evaluate the expression \\(\\sin (\\arcsin 0.4 + \\arcsin 0.5) \\cdot \\sin (\\arcsin 0.5 - \\arcsin 0.4)\\). Hmm, that looks a bit complicated, but maybe I can break it down step by step.\n\nFirst, I remember that there are some trigonometric identities that might help simplify this. I recall the sine addition and subtraction formulas. Let me write them down:\n\n1. \\(\\sin(A + B) = \\sin A \\cos B + \\cos A \\sin B\\)\n2. \\(\\sin(A - B) = \\sin A \\cos B - \\cos A \\sin B\\)\n\n So, if I let \\(A = \\arcsin 0.4\\) and \\(B = \\arcsin 0.5\\), then the expression becomes \\(\\sin(A + B) \\cdot \\sin(A - B)\\). That seems manageable.\n\nLet me compute each part separately. Let me denote \\(A = \\arcsin 0.4\\), which means that \\(\\sin A = 0.4\\). Similarly, \\(B = \\arcsin 0.5\\), so \\(\\sin B = 0.5\\). Since both arcsin functions are in the range \\([- \\pi/2, \\pi/2]\\), both A and B are in the first quadrant, so their cosines will be positive.\n\n **[…] (omit 7105 tokens)** \n\n**Final Answer**\n\\boxed{-\\dfrac{9}{100}}\n**</think>**\n\nTo evaluate the expression \\(\\sin (\\arcsin 0.4 + \\arcsin 0.5) \\cdot \\sin (\\arcsin 0.5 - \\arcsin 0.4)\\), we start by letting \\(A = \\arcsin 0.4\\) and \\(B = \\arcsin 0.5\\). This transforms the expression into \\(\\sin(A + B) \\cdot \\sin(A - B)\\).\n\nUsing the sine addition and subtraction formulas:\n\\[\n\\sin(A + B) = \\sin A \\cos B + \\cos A \\sin B\n\\]\n\\[\n\\sin(A - B) = \\sin A \\cos B - \\cos A \\sin B\n\\]\n\nWe know:\n- \\(\\sin A = 0.4\\)\n- \\(\\sin B = 0.5\\)\n- \\(\\cos A = \\sqrt{1 - (0.4)^2} = \\sqrt{0.84}\\)\n- \\(\\cos B = \\sqrt{1 - (0.5)^2} = \\sqrt{0.75}\\)\n\nUsing the identity for the product of sines:\n\\[\n\\sin(A + B) \\cdot \\sin(A - B) = \\sin^2 A \\cos^2 B - \\cos^2 A \\sin^2 B\n\\]\n\nSubstituting the known values:\n\\[\n\\sin^2 A = (0.4)^2 = 0.16\n\\]\n\\[\n\\cos^2 B = (\\sqrt{0.75})^2 = 0.75\n\\]\n\\[\n\\cos^2 A = (\\sqrt{0.84})^2 = 0.84\n\\]\n\\[\n\\sin^2 B = (0.5)^2 = 0.25\n\\]\n\nCalculating each term:\n\\[\n\\sin^2 A \\cos^2 B = 0.16 \\times 0.75 = 0.12\n\\]\n\\[\n\\cos^2 A \\sin^2 B = 0.84 \\times 0.25 = 0.21\n\\]\n\nSubtracting these results:\n\\[\n0.12 - 0.21 = -0.09\n\\]\n\nThus, the value of the expression is:\n\\[\n\\boxed{-\\dfrac{9}{100}}\n\\] **(8508 tokens in total)**

TACLer-1.5B (*Thinking* Mode): √
<think>\n Okay, so I have this trigonometric expression to evaluate: \\[\\sin (\\arcsin 0.4 + \\arcsin 0.5) \\cdot \\sin (\\arcsin 0.5 - \\arcsin 0.4).\\] Hmm, that looks a bit complicated at first glance, but maybe I can break it down step by step.\n\nFirst, I remember that there are some trigonometric identities that relate sine and arcsine functions. Specifically, the sine of a sum and difference of two angles. Maybe I can use those identities here.\n\nLet me recall the sine addition and subtraction formulas:\n\n1. \\(\\sin(A + B) = \\sin A \\cos B + \\cos A \\sin B\\)\n2. \\(\\sin(A - B) = \\sin A \\cos B - \\cos A \\sin B\\)\n\nSo, the expression is the product of two sine terms:\n\n\\[\n[\\sin(\\arcsin 0.4 + \\arcsin 0.5)] \\cdot [\\sin(\\arcsin 0.5 - \\arcsin 0.4)]\n\\]\n\n Maybe I can compute each sine term separately and then multiply them together.\n\nLet me denote:\n\nLet \\(\\alpha = \\arcsin 0.4 \\) and \\(\\beta = \\arcsin 0.5 \\).\n\nSo, the expression becomes:\n\n\\[\n\\sin(\\alpha + \\beta) \\cdot \\sin(\\beta - \\alpha)\n\\]\n\nHmm, interesting. Now, I can use the sine addition and subtraction formulas on each sine term.\n\nFirst, let‘s compute \\(\\sin(\\alpha + \\beta) \\):\n\nUsing the addition formula:\n\n\\[\n\\sin(\\alpha + \\beta) = \\sin \\alpha \\cos \\beta + \\cos \\alpha \\sin \\beta\n\\]\n\nSimilarly, \\(\\sin(\\beta - \\alpha) \\) can be expanded using the subtraction formula:\n\n\\[\n\\sin(\\beta - \\alpha) = \\sin \\beta \\cos \\alpha - \\cos \\beta \\sin \\alpha\n\\]\n\nSo, the original expression becomes:\n\n\\[\n[\\sin \\alpha \\cos \\beta + \\cos \\alpha \\sin \\beta] \\cdot [\\sin \\beta \\cos \\alpha - \\cos \\beta \\sin \\alpha]\n\\]\n\nLet me write that down:\n\n\\[\n(\\sin \\alpha \\cos \\beta + \\cos \\alpha \\sin \\beta)(\\sin \\beta \\cos \\alpha - \\cos \\beta \\sin \\alpha)\n\\]\n\n **[…] (omit 2353 tokens)** I didn‘t need to compute all those square roots and stuff. Just applying the identity.\n\nTherefore, the answer is 9/100.\n\n **Final Answer**\n\\boxed{\\dfrac{9}{100}}\n**</think>**\n\nTo evaluate the expression \\(\\sin (\\arcsin 0.4 + \\arcsin 0.5) \\cdot \\sin (\\arcsin 0.5 - \\arcsin 0.4)\\), we can use trigonometric identities and properties of inverse sine functions.\n\nFirst, let \\(\\alpha = \\arcsin 0.4\\) and \\(\\beta = \\arcsin 0.5\\). The expression becomes:\n\\[\n\\sin(\\alpha + \\beta) \\cdot \\sin(\\beta - \\alpha)\n\\]\n\nUsing the identity for the product of sines, we recognize that:\n\\[\n\\sin(\\alpha + \\beta) \\cdot \\sin(\\beta - \\alpha) = \\sin^2(\\beta) - \\sin^2(\\alpha)\n\\]\n\nSince \\(\\alpha = \\arcsin 0.4\\) and \\(\\beta = \\arcsin 0.5\\), we have:\n\\[\n\\sin(\\alpha) = 0.4 \\quad \\text{and} \\quad \\sin(\\beta) = 0.5\n\\]\n\nThus, we can compute:\n\\[\n\\sin^2(\\beta) - \\sin^2(\\alpha) = (0.5)^2 - (0.4)^2 = 0.25 - 0.16 = 0.09 = \\frac{9}{100}\n\\]\n\nTherefore, the final answer is:\n\\[\n\\boxed{\\dfrac{9}{100}}\n\\] **(3395 tokens in total**

Figure 13: An example MATH 500 problem at level 2 and the responses generated by R1-Qwen and TACLer under *Thinking* mode.

Question (MATH500): How many positive two-digit integers are factors of both 100 and 150?
Difficulty Level: 3
Answer: 3

DeepSeek-R1-Distill-Qwen-1.5B (*NoThinking* Mode): ✗
<think>\nOkay, I think I can solve it directly.**</think>**\n\n**Problem:** \nHow many positive two-digit integers are factors of both 100 and 150?\n\n**Solution:**\n\nTo find the positive two-digit integers that are factors of both 100 and 150, we'll follow these steps:\n\n1. **Find the Greatest Common Factor (GCF) of 100 and 150.** \n - **Prime Factorization of 100:** \n \\(100 = 2^2 \\times 5^2\\)\n - **Prime Factorization of 150:** \n \\(150 = 2 \\times 3 \\times 5^2\\)\n - **Identify Common Factors:** \n The common prime factors are \\(2\\) and \\(5^2\\). \n So, \\(GCF = 2 \\times 5^2 = 50\\).\n\n2. **Determine the Two-Digit Factors of 50.** \n - **List the Factors of 50:** \n 1, 2, 5, 10, 25, 50 \n - **Filter for Two-Digit Integers:** \n From the list, the two-digit factors are 10 and 25.\n\n3. **Count the Two-Digit Factors:** \n There are **2** positive two-digit integers (10 and 25) that are factors of both 100 and 150.\n\n**Final Answer:** \n\\boxed{2}. **(372 tokens)**

DeepSeek-R1-Distill-Qwen-1.5B (*Thinking* Mode): √
<think>\nAlright, so I need to figure out how many positive two-digit integers are factors of both 100 and 150. Hmm, okay. Let me break this down step by step. \n\nFirst, I remember that if a number is a factor of both 100 and 150, it must be a common factor of these two numbers. So, I need to find all the common factors of 100 and 150 that are two-digit numbers. That makes sense.\n\nMaybe I should start by finding the factors of each number individually and then see where they overlap. That sounds like a plan. Let me start with 100. \n\nThe factors of 100 are all the numbers that can divide 100 without leaving a remainder. Let me list them out. Starting from 1:\n\n1, 2, 4, 5, 10, 20, 25, 50, 100. \n\nWait, let me check that. 1 times 100 is 100, so that‘s a factor. 2 times 50 is 100, so 2 and 50 are factors. Then 4 times 25 is 100, so 4 and 25. 5 times 20 is 100, so 5 and 20. Then 10 times 10 is 100, so 10 and 10. So yeah, those are all the factors of 100. \n\nNow, let me do the same for 150. The factors of 150 are a bit more, I think. Let me see:\n\n1, 2, 3, 5, 6, 10, 15, 25, 30, 50, 75, 150. \n\nLet me verify. 1 times 150 is 150, 2 times 75 is 150, 3 times 50 is 150, 5 times 30 is 150, 6 times 25 is 150, 10 times 15 is 150. Then 15 times 10 is 150, 25 times 6 is 150, 30 times 5 is 150, 50 times 3 is 150, 75 times 2 is 150, and 150 times 1 is 150. So, that seems comprehensive.\n\nNow, I need to find the common factors of both 100 and 150. So, looking at the factors of 100: 1, 2, 4, 5, 10, 20, 25, 50, 100. And factors of 150: 1, 2, 3, 5, 6, 10, 15, 25, 30, 50, 75, 150.\n\nComparing these two lists, the common factors are the numbers that appear in both. So, let’s see:\n\n1 is common, 2 is common, 5 is common, 10 is common, 25 is common, 50 is common. Wait, what about 150? It‘s only in the factors of 150, not in 100. Similarly, 75 is only in 150, not in 100. So, the common factors are 1, 2, 5, 10, 25, 50. \n\nBut hold on, the question is asking for positive two-digit integers. So, two-digit numbers are from 10 to 99 inclusive. Looking at the common factors, we have 10, 25, and 50. \n\nWait, let me double-check that. 1 is a single-digit number, so it doesn’t count. 2 is also single-digit. 5 is single-digit, 10 is two-digit, 25 is two-digit, 50 is two-digit. So, the common two-digit factors are 10, 25, and 50.\n\nIs that all? Let me make sure I didn‘t miss any. Looking back at the factors of 100: 1, 2, 4, 5, 10, 20, 25, 50, 100. And factors of 150: 1, 2, 3, 5, 6, 10, 15, 25, 30, 50, 75, 150. \n\nSo, the common factors are indeed 1, 2, 5, 10, 25, 50. So, within these, the two-digit numbers are 10, 25, and 50. \n\nBut wait a second, 50 is a two-digit number, yes, but 100 is three-digit, so it’s excluded. So, 10, 25, 50 are the common two-digit factors.\n\nIs there any chance I missed any other common factors? Let me think. 100 is 2^2 * 5^2, and 150 is 2 * 3 * 5^2. So, the greatest common divisor (GCD) of 100 and 150 would be the product of the lowest powers of the common prime factors. So, GCD is 2^1 * 5^2 = 2 * 25 = 50. So, the GCD is 50. That means the common factors are all the factors of 50.\n\nWait, hold on, factors of 50. Let me list the factors of 50: 1, 2, 5, 10, 25, 50. So, yeah, that‘s consistent with what I found earlier. So, the common factors are exactly the factors of 50. So, the two-digit ones are 10, 25, 50.\n\nTherefore, the positive two-digit integers that are factors of both 100 and 150 are 10, 25, and 50. So, that’s three numbers. **[…] (omit 1346 tokens)** \n\nI think that‘s solid. I don’t see any mistakes in the reasoning. So, I‘m confident that the answer is 3.\n\n**Final Answer**\nThe number of positive two-digit integers that are factors of both 100 and 150 is \\boxed{3}.\n**</think>**\n\nTo determine how many positive two-digit integers are factors of both 100 and 150, we start by finding the factors of each number.\n\nThe factors of 100 are: 1, 2, 4, 5, 10, 20, 25, 50, 100.\n\nThe factors of 150 are: 1, 2, 3, 5, 6, 10, 15, 25, 30, 50, 75, 150.\n\nNext, we identify the common factors of both sets. These common factors are: 1, 2, 5, 10, 25, 50.\n\nWe are specifically interested in the two-digit integers among these common factors. The two-digit numbers are 10, 25, and 50.\n\nThus, the number of positive two-digit integers that are factors of both 100 and 150 is \\(\\boxed{3}\\). **(3114 tokens in total)**

Figure 14: An example MATH 500 problem and the corresponding responses generated by R1-Qwen under two reasoning modes.

Question (MATH500): How many positive two-digit integers are factors of both 100 and 150?
Difficulty Level: 3
Answer: 3

TACLer-1.5B (*NoThinking* Mode): √
<think>\nOkay, I think I can solve it directly.**</think>**\n\n Alright, so I need to find how many positive two-digit integers are factors of both 100 and 150. Hmm, that sounds like I need to find the common factors of 100 and 150 that are two-digit numbers. Let me break this down step by step.\n\nFirst, maybe I should find all the factors of 100 and all the factors of 150, and then see which ones they have in common, and among those, how many are two-digit numbers. Alternatively, maybe it‘s easier to find the greatest common factor (GCF) of 100 and 150 first and then find all two-digit factors of that GCF.\n\nWait, that might be a good approach. Let me try that.\n\nSo, first, let me find the GCF of 100 and 150.\n\nTo find the GCF, I can use the prime factorization method.\n\nLet’s factorize 100:\n100 can be broken down into prime factors:\n100 = 2^2 * 5^2\n\nNow, factorize 150:\n150 = 2 * 3 * 5^2\n\nSo, the GCF is the product of the lowest powers of the common primes. **[...] (omit 660 tokens)** \n\nFactors of 150: 1, 2, 3, 5, 6, 10, 15, 25, 30, 50, 75, 150.\n\nCommon factors: 1, 2, 5, 10, 25, 50.\n\nSo, the common two-digit factors are 10, 25, 50.\n\nHence, there are 3 such numbers.\n\nTherefore, the answer is 3.\n\nWait, just to make sure I didn‘t miss any, let me list the two-digit numbers that are factors of both 100 and 150.\n\nFrom the list of common factors: 10, 25, 50.\n\nYep, those are the only two-digit numbers.\n\nSo, the number of positive two-digit integers that are factors of both 100 and 150 is 3.\n\n**Final Answer**\nThe number of positive two-digit integers that are factors of both 100 and 150 is \\boxed{3}. **(1203 tokens in total)**

TACLer-1.5B (*Thinking* Mode): √
<think>\nOkay, so I have this problem: \“How many positive two-digit integers are factors of both 100 and 150?\” Hmm, let me try to figure this out step by step. I remember that when we‘re looking for common factors of two numbers, we can use the concept of the greatest common divisor (GCD). The GCD of two numbers is the largest number that divides both of them without leaving a remainder. But here, we’re not just looking for the GCD, but specifically the two-digit factors of that GCD. Wait, maybe I need to clarify that.\n\nFirst, let me find the factors of 100 and 150 separately and then see what they have in common. But that might take too long because 100 and 150 are not too large, but it‘s better to use the GCD method.\n\nSo, let’s find the GCD of 100 and 150. To do that, I can use the prime factorization method. Let me break down both numbers into their prime factors.\n\nStarting with 100: 100 divided by 2 is 50, divided by 2 again is 25, which is 5 squared. So, 100 is 2^2 * 5^2.\n\nNow for 150: 150 divided by 2 is 75, then 75 divided by 3 is 25, which is 5 squared. So, 150 is 2 * 3 * 5^2.\n\nTo find the GCD, I take the lowest power of all common prime factors. The common primes here are 2 and 5. The lowest power of 2 is 2^1 and the lowest power of 5 is 5^2. So, GCD = 2^1 * 5^2 = 2 * 25 = 50.\n\nSo, the greatest common divisor of 100 and 150 is 50. That means the largest number that divides both 100 and 150 is 50. But the question is asking for two-digit integers that are factors of both. So, we need to find all two-digit numbers that are factors of 50.\n\nWait, hold on. If the GCD is 50, then the factors of both 100 and 150 are exactly the factors of 50. So, now I need to list all the two-digit factors of 50.\n\nWhat are the factors of 50? Let me list them: 1, 2, 5, 10, 25, 50. So, these are the factors of 50. Now, among these, the two-digit numbers are 10, 25, and 50. But wait, 10, 25, and 50. But 50 is a two-digit number? Wait, 50 is a two-digit number, yes.\n\nWait, but hold on. The question is asking for positive two-digit integers that are factors of both 100 and 150. So, if 50 is a factor of both, then 50 is a two-digit number that satisfies the condition. Similarly, 10 and 25 are two-digit numbers.\n\nSo, let me list them: 10, 25, 50. Are these all the two-digit factors of 50? Let‘s see: 10 is 10, 25 is 25, 50 is 50. Are there any other two-digit numbers that are factors of 50? Let me check.\n\n50 divided by 5 is 10, which is two-digit. 50 divided by 10 is 5, which is single-digit. 50 divided by 25 is 2, which is single-digit. 50 divided by 50 is 1, which is single-digit. So, the only two-digit factors are 10, 25, and 50. So that’s three numbers**. [...] (omit 945 tokens)** \n\nYes.\n\nSo, all three numbers are valid.\n\nTherefore, the number of positive two-digit integers that are factors of both 100 and 150 is 3.\n\n**Final Answer**\nThe number of positive two-digit integers that are factors of both 100 and 150 is \\boxed{3}.\n**</think>**\n\nTo determine how many positive two-digit integers are factors of both 100 and 150, we start by finding the greatest common divisor (GCD) of 100 and 150 using their prime factorizations.\n\nThe prime factorization of 100 is:\n\\[100 = 2^2 \\times 5^2 \\]\n\nThe prime factorization of 150 is:\n\\[150 = 2 \\times 3 \\times 5^2 \\]\n\nThe GCD is found by taking the lowest power of all common prime factors:\n\\[\\text{GCD} = 2^1 \\times 5^2 = 2 \\times 25 = 50 \\]\n\nNext, we list the factors of 50, which are:\n\\[1, 2, 5, 10, 25, 50 \\]\n\nWe are interested in the two-digit factors of 50. These are:\n\\[10, 25, 50 \\]\n\nThus, the number of positive two-digit integers that are factors of both 100 and 150 is:\n\\[\n\\boxed{3}\n\\]. **(2227 tokens in total)**

Figure 15: An example MATH 500 problem and the corresponding responses generated by TACLer under two reasoning modes.