ThinkEdit: Interpretable Weight Editing to Mitigate Overly Short Thinking in Reasoning Models

Chung-En Sun
UCSD CSE
cesun@ucsd.edu

Ge Yan
UCSD CSE
geyan@ucsd.edu

Tsui-Wei Weng
UCSD HDSI
lweng@ucsd.edu

Abstract

Recent studies have shown that Large Language Models (LLMs) augmented with chain-of-thought (CoT) reasoning demonstrate impressive problem-solving abilities. However, in this work, we identify a recurring issue where these models occasionally generate overly short reasoning, leading to degraded performance on even simple mathematical problems. Specifically, we investigate how reasoning length is embedded in the hidden representations of reasoning models and its impact on accuracy. Our analysis reveals that reasoning length is governed by a linear direction in the representation space, allowing us to induce overly short reasoning by steering the model along this direction. Building on this insight, we introduce ***ThinkEdit***, a simple yet effective weight-editing approach to mitigate the issue of overly short reasoning. We first identify a small subset of attention heads (approximately 4%) that predominantly drive short reasoning behavior. We then edit the output projection weights of these heads to remove the short reasoning direction. With changes to only 0.2% of the model's parameters, ***ThinkEdit*** effectively reduces overly short reasoning and yields notable accuracy gains for short reasoning outputs (+6.39%), along with an overall improvement across multiple math benchmarks (+3.34%). Our findings provide new mechanistic insights into how reasoning length is controlled within LLMs and highlight the potential of fine-grained model interventions to improve reasoning quality. Our code is available at: https://github.com/Trustworthy-ML-Lab/ThinkEdit

1 Introduction

Recently, Reinforcement Learning (RL) has been applied to enhance Large Language Models (LLMs), equipping them with strong chain-of-thought (CoT) reasoning abilities (Guo et al., 2025). These models, often referred to as reasoning models, first generate an intermediate reasoning process – a "thinking step" – where they reason step-by-step and then self-correct before producing a final response. As a result, they achieve remarkable improvement on mathematical reasoning tasks and demonstrate a strong ability to generate detailed CoT reasoning (Jaech et al., 2024; Guo et al., 2025; Muennighoff et al., 2025).

However, despite these improvements, reasoning models still exhibit a non-negligible gap from perfect accuracy on relatively simple benchmarks such as GSM8K (Cobbe et al., 2021). As shown in Section 2, we found that Deepseek-distilled reasoning models occasionally generate overly short reasoning chains, which correlate with lower accuracy (about 20% drop on MATH-level5 benchmark (Hendrycks et al., 2021b)). This issue appears consistently across models of different sizes, suggesting that reasoning length plays a crucial role in problem-solving effectiveness. Yet, the mechanisms governing reasoning length within the model's internal representation remain underexplored, despite being crucial for understanding reasoning models.

To bridge this gap, in this work, we first investigate how reasoning length is encoded within the hidden representations of reasoning models. By performing a novel analysis of the residual stream, we extract a *reasoning length direction*—a latent linear representation in the residual stream that enables direct control over reasoning length as shown in Figure 2 (left). Our analysis reveals that overly short, abstract, or high-level reasoning significantly degrades model performance, and this characteristic is primarily embedded in the middle layers of the model. Furthermore, we identify a small subset (approximately 4%) of attention heads in the middle layers that disproportionately contribute to short reasoning. Building on this insight, we propose ***ThinkEdit***, a simple and effective weight-editing technique to remove the short-reasoning component from these attention heads' output pro-

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 17001–17025
November 4-9, 2025 ©2025 Association for Computational Linguistics

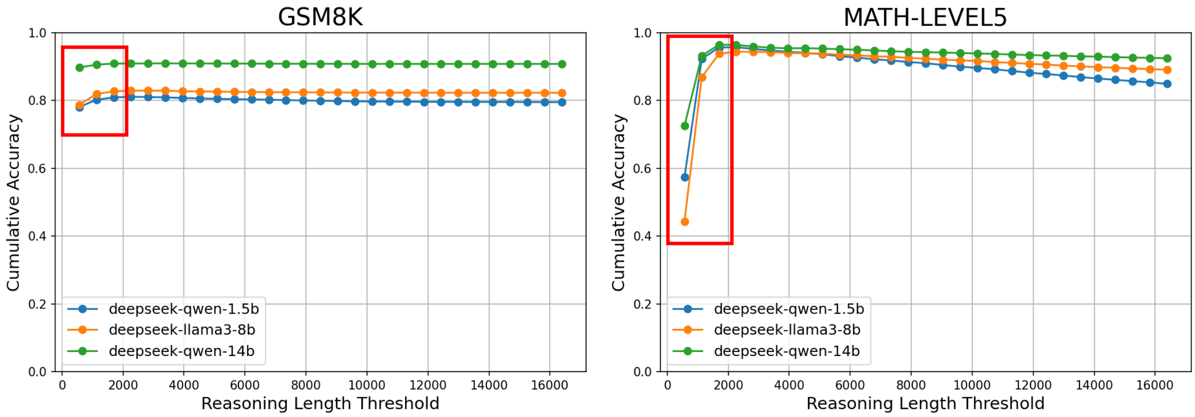

Figure 1: Cumulative accuracy as a function of the reasoning length threshold. The x-axis represents the cutoff threshold on reasoning length, and the y-axis shows the average accuracy of all responses with reasoning length below that threshold. Models consistently exhibit lower accuracy for overly short reasoning (e.g. length <1000).

jection layers, as shown in Figure 2 (right). Our findings demonstrate that disabling these components leads to a non-trivial improvement in accuracy when the model generates short reasoning while also enhancing overall performance. Our contributions are summarized as follows:

- We identify the prevalence of overly short reasoning across Deepseek-distilled reasoning models of different scales and highlight its impact on the performance of math benchmarks.
- We extract a *reasoning length direction* in the model's hidden representations, revealing that **middle layers** play a crucial role in controlling reasoning length. To the best of our knowledge, this is the first work to systematically study the internal representations of reasoning models.
- We discover a small set of **"short reasoning" heads** that strongly contribute to the generation of brief reasoning chains and propose ***ThinkEdit***. By editing the output projection weights of just 4% heads (0.2% of the model's total parameters), ***ThinkEdit*** effectively mitigates short reasoning, leading to improved performance both when short reasoning occurs (+6.39%) and in overall accuracy (+3.34%).

2 Unexpectedly Low Accuracy in Short Reasoning Cases

We begin our study by highlighting a consistent issue observed in Deepseek-distilled reasoning models across a variety of sizes: significantly lower accuracy when the reasoning length is short. This pattern holds across datasets such as GSM8K (Cobbe et al., 2021) and MATH-Level5 (Hendrycks et al., 2021b). Figure 1 illustrates this trend, with the x-axis indicating a cutoff threshold on reasoning length. For example, a threshold of 2000 denotes that we calculate the average accuracy over all responses whose reasoning length is at most 2000 tokens. The y-axis shows the corresponding cumulative accuracy. The details of the experimental setup are provided in Section 4.4.

Contrary to intuition, one might expect shorter reasoning to correspond to easier questions, as such problems should require fewer steps to solve. This expectation is partially supported by the trend in Figure 1 (right), where accuracy tends to decrease as reasoning length exceeds 2000. However, the region with reasoning length below 2000 (highlighted in red boxes) exhibits a different pattern: models consistently underperform on these short-reasoning cases, with accuracy dropping significantly below the overall average. This suggests that, rather than efficiently solving simple problems with brief reasoning, models often fail when producing overly short chains of thought.

Motivated by this observation, we focus on investigating how a model's internal representations govern reasoning length and influence accuracy. In Section 3, we analyze the relationship between hidden representations, reasoning length, and model performance. Building on these insights, we propose ***ThinkEdit***, a simple yet effective weight-editing method, in Section 4, which modifies the output layer of a few key attention heads to mitigate the problem of overly short reasoning.

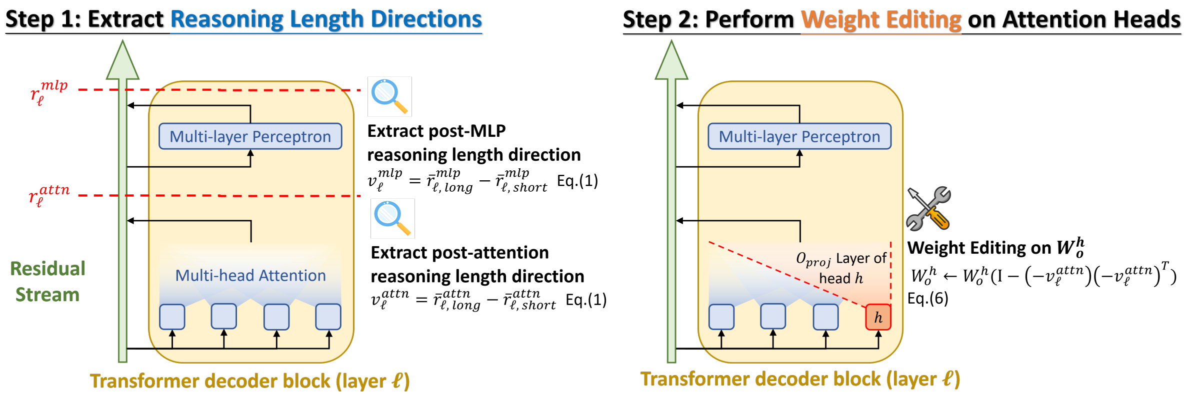

Figure 2: The overview of ***ThinkEdit*** framework. We first identify that there exist linear directions for controlling reasoning length in the hidden space, and then perform weight editing on the key attention heads.

3 Understanding How Representations Affect Reasoning Length

In this section, we explore how reasoning length is encoded in the hidden representation of a reasoning model. In Section 3.1, we provide an overview of the transformer structure, highlighting the specific points in the residual stream where the representation of interest resides. Then, in Section 3.2, we present our method for extracting linear directions that allow control over reasoning length. Finally, in Section 3.3, we analyze the performance of reasoning models when guided by these extracted reasoning length directions.

3.1 Background of Transformer Structure and Notations

A transformer model consists of multiple stacked layers, each containing a *multi-headed self-attention* (Attn) module followed by a *feed-forward Multi-Layer Perceptron* (MLP). The model maintains an evolving *Residual Stream*, where representations are progressively refined as they pass through layers. The update at each layer ℓ can be expressed as:

$$r_\ell^{\text{attn}} = r_{\ell-1}^{\text{mlp}} + \text{Attn}(\text{LayerNorm}(r_{\ell-1}^{\text{mlp}}))$$
$$r_\ell^{\text{mlp}} = r_\ell^{\text{attn}} + \text{MLP}(\text{LayerNorm}(r_\ell^{\text{attn}}))$$

where $r_{\ell-1}^{\text{mlp}}$ is the hidden state entering layer ℓ, which is also the output of the MLP from the previous layer $\ell - 1$, r_ℓ^{attn} represents the intermediate state of the residual stream after the self-attention module, and r_ℓ^{mlp} denotes the final output after the MLP transformation.

Our focus is on the hidden representations r_ℓ^{attn} and r_ℓ^{mlp} as illustrated in Figure 2 (left), which capture the model's state after the self-attention and MLP transformations, respectively.

3.2 Extracting Reasoning Length Directions

To investigate how *reasoning length* is encoded in a model's hidden representation, we begin by collecting the model's responses to 2,000 problems from the GSM8K (Cobbe et al., 2021) training set. In each response, the chain-of-thought (CoT) is enclosed between special tags `<think>` and `</think>`. We measure the length of each CoT by counting only the tokens within these tags. We then construct two datasets $\mathcal{D}_{\text{long}}$ and $\mathcal{D}_{\text{short}}$, where $\mathcal{D}_{\text{long}}$ consists of responses whose CoT exceeds 1000 tokens and $\mathcal{D}_{\text{short}}$ includes those under 100 tokens. Each entry in these datasets contains: (1) the problem statement, (2) the extracted CoT, enclosed by `<think>` and `</think>` tags, and (3) the step-by-step calculation process leading to the final answer.

Next, we input the problem statement along with its CoT into the model and extract hidden representations at each layer ℓ for both the *post-attention* and *post-MLP* residual streams, denoted as r_ℓ^{attn} and r_ℓ^{mlp}, respectively. Specifically, let $r_\ell^{\text{attn}}(i,t)$ and $r_\ell^{\text{mlp}}(i,t)$ represent the hidden representations at layer ℓ for token position t in the response to problem i. We first compute the mean hidden representation over the chain-of-thought (CoT) tokens, where $\mathcal{T}_i$ denotes the set of token positions enclosed within the `<think>` and `</think>` tags, and then compute the mean across all problems in the datasets $\mathcal{D}_{\text{long}}$ and $\mathcal{D}_{\text{short}}$, yielding layerwise embeddings:

$$\bar{r}_{\ell,y}^{x} = \frac{1}{|\mathcal{D}_y|} \sum_{i \in \mathcal{D}_y} \frac{1}{|\mathcal{T}_i|} \sum_{t \in \mathcal{T}_i} r_\ell^x(i,t),$$

where $x \in \{\text{attn}, \text{mlp}\}$ denotes the representation type and $y \in \{\text{long}, \text{short}\}$ indicates the reasoning-length group. Finally, we define the *reasoning-length direction* at layer ℓ as the vector difference between the "long" and "short" embeddings:

$$v_\ell^{\text{attn}} = \overline{r}_{\ell,\text{long}}^{\text{attn}} - \overline{r}_{\ell,\text{short}}^{\text{attn}}, \quad v_\ell^{\text{mlp}} = \overline{r}_{\ell,\text{long}}^{\text{mlp}} - \overline{r}_{\ell,\text{short}}^{\text{mlp}}. \tag{1}$$

These two vectors, $v_\ell^{\text{attn}}, v_\ell^{\text{mlp}} \in \mathbb{R}^d$ (with d denoting the hidden dimension), capture how the model's representation differs when reasoning chains are notably longer or shorter. In the next section, we analyze how modifying these directions in the residual stream influences both reasoning length and overall model performance.

3.3 Effects of Reasoning-Length Direction

In Section 3.2, we have obtained the steering vectors v_ℓ^{attn} and v_ℓ^{mlp} for reasoning length. We now investigate how modifying the residual stream along these directions affects both reasoning length and model accuracy. We begin with *global steering*, where we apply a uniform shift α across all layers, and then delve into *layerwise* steering experiments to locate the portions of the network most responsible for reasoning length.

Steering Reasoning Models with v_ℓ^{attn} and v_ℓ^{mlp}. Let α be a scalar weight in the range $[-0.08, 0.08]$. For each layer ℓ, we apply the following transformations:

$$r_\ell^{\text{attn}} \leftarrow r_\ell^{\text{attn}} + \alpha\, v_\ell^{\text{attn}}, \quad r_\ell^{\text{mlp}} \leftarrow r_\ell^{\text{mlp}} + \alpha\, v_\ell^{\text{mlp}}. \tag{2}$$

This operation *steers* the model's internal states either *toward longer reasoning* (if $\alpha > 0$) or *toward shorter reasoning* (if $\alpha < 0$).

Experimental Setup. We evaluate the effect of reasoning-length directions using two test sets:

- **GSM8K (200 problems) (Cobbe et al., 2021):** A simpler benchmark, consisting of the first 200 problems from the GSM8K test set.
- **MATH-Level5 (140 problems) (Hendrycks et al., 2021b):** A more challenging benchmark, comprising 140 problems selected from the MATH test set. Specifically, we extract 20 level-5 examples from each of 7 categories.

We set a maximum reasoning length of 8,192 tokens for GSM8K and 16,384 tokens for MATH-Level5. Upon reaching this limit, the model is prompted to finalize its answer immediately. We experiment on three reasoning models of varying sizes: `deepseek-distill-qwen-1.5B`, `deepseek-distill-llama3-8B`, and `deepseek-distill-qwen-14B`.

Global Steering on GSM8K and MATH-Level5. Figure 3 (Top) shows the effect of applying the attention-based direction v_ℓ^{attn} on GSM8K. We vary α from -0.08 (shorter reasoning) to $+0.08$ (longer reasoning). Across all models, *increasing* α extends the length of CoT (Figure 3, top right), indicating that v_ℓ^{attn} indeed encode *reasoning-length* attributes. In terms of accuracy, the larger 8B and 14B models improve when steered toward longer reasoning—particularly `deepseek-distill-llama3-8B` (orange line), which benefits most from positive steering with 10% accuracy improvement. In contrast, the smaller `deepseek-distill-qwen-1.5B` (blue line) model experiences a 10% drop in accuracy. Figure 3 (Bottom) presents the results for the more challenging MATH-Level5 dataset. Similar to GSM8K, our extracted directions effectively control reasoning length as expected, with negative α consistently leading to shorter CoT and positive α extending them. In terms of accuracy, shorter reasoning also consistently degrades performance. However, unlike GSM8K, there is no clear trend indicating that longer reasoning reliably enhances accuracy; while moderate positive α might provide some benefits, excessively long reasoning often negatively impacts performance. We also present results using the MLP-based direction v_ℓ^{mlp} in Appendix A.1, which exhibit similar trends.

Layerwise Steering Analysis. We perform a *layerwise* experiment to identify which layers produce reasoning-length directions with the strongest impact. As shown in Appendix A.2, *middle layers* are most effective, suggesting their key role in encoding reasoning-length representations.

Budget Control with Steering Representations. Recent work (Muennighoff et al., 2025) proposed an interesting approach to enforce budget constraints by stopping the CoT or appending "Wait" to prolong it. However, stopping the CoT prematurely may cause incomplete reasoning and appending "Wait" may risk misalignment with the model's natural CoT. Alternatively, steering representations may allow for a more coherent way to modulate reasoning length (see Appendix A.8) – by directly manipulating the model's internal representations,

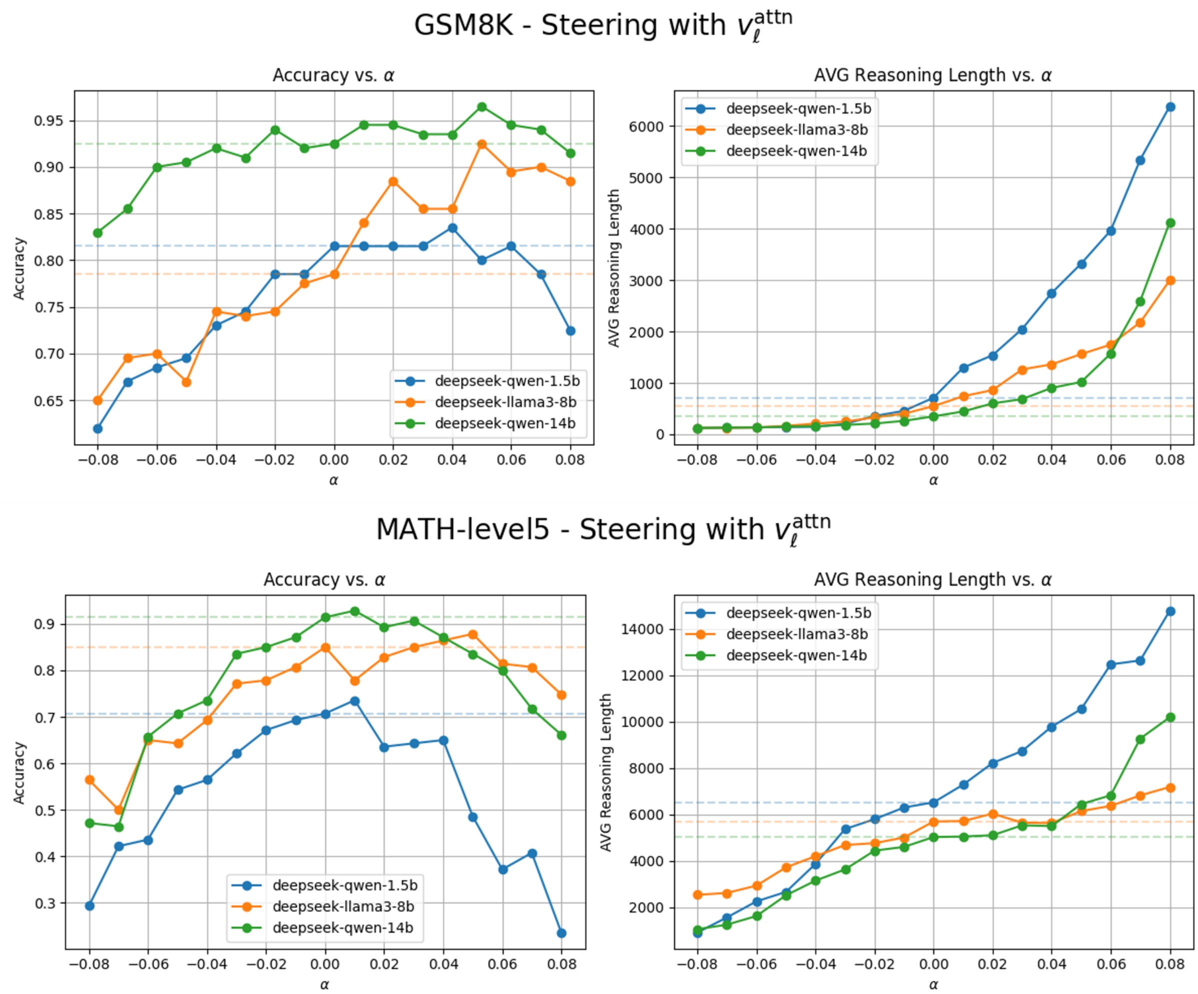

Figure 3: Global steering results. **Top:** On GSM8K, positive α extends reasoning length and improves accuracy in the 8B and 14B models, while negative α shortens reasoning and lowers accuracy. **Bottom:** On MATH-Level5, negative α similarly shortens reasoning and reduces accuracy.

one can more effectively balance the computational cost and performance.

Key insights. Based on these experiments, we observe that:

1. While steering the model toward longer reasoning ($\alpha > 0$) does not always guarantee improved performance, steering toward short reasoning ($\alpha < 0$) consistently degrades accuracy. This suggests that the overly short reasoning with reduced accuracy, as observed in Section 2, is driven by a specific and identifiable pattern in the hidden representations.

2. Layerwise analysis reveals that the *middle layers* play a key role in regulating reasoning length.

Based on these findings, we hypothesize that certain critical components within the middle layers may contribute to short reasoning. In the next section, we pinpoint these components and perform weight editing to mitigate their effects.

4 *ThinkEdit*: Mitigate Overly Short Reasoning through Weight Editing

Building on the insights from Section 3.3, in this section, we propose ***ThinkEdit***, an effective weight-editing method to mitigate overly short reasoning. We start by analyzing whether specific components within reasoning models significantly contribute to the phenomenon of *short reasoning*. Our focus is on pinpointing particular attention heads, as the attention mechanism plays a crucial role in information propagation across tokens. To explore this, we begin with an overview of the multi-head attention mechanism in Section 4.1, where we define the contribution of individual attention heads. Using this definition, we identify short reasoning heads in Section 4.2 and remove the short reasoning component from these heads in Section 4.3. Finally, in Section 4.4, we evaluate ***ThinkEdit***, and show that it effectively mitigates the overly short reasoning issue.

4.1 Overview of Attention-Head Structure

A self-attention layer typically includes multiple *attention heads*, each responsible for capturing distinct token-to-token dependencies. Let d denote the model's hidden dimension, and H the number of attention heads. Each head h operates on a subspace of size $d_h = \frac{d}{H}$ using the following steps:

- **Q, K, and V Projections.** Given a hidden-state $r \in \mathbb{R}^{T\times d}$ (for T tokens), each head h computes:

  $$Q^h = rW_q^h,\ K^h = rW_k^h,\ V^h = rW_v^h,$$

  where $Q^h, K^h, V^h \in \mathbb{R}^{T\times d_h}$. Each head h has its own learnable projection matrices $W_q^h, W_k^h, W_v^h \in \mathbb{R}^{d\times d_h}$, which transform the hidden representation r into query, key, and value vectors.

- **Self-Attention Computation.** The head outputs an attention-weighted combination of V^h:

  $$A^h = \text{softmax}\Big(\frac{Q^h(K^h)^\top}{\sqrt{d_h}}\Big)\, V^h \ \in\ \mathbb{R}^{T\times d_h}.$$

- **Output Projection.** Each head's output A^h is merged back into the residual stream via a learned projection matrix $W_o^h \in \mathbb{R}^{d_h\times d}$, producing the final *per-head contribution* C^h:

  $$C^h := A^h W_o^h \ \in\ \mathbb{R}^{T\times d}. \tag{3}$$

  The final multi-head attention output is then obtained by summing the contributions from all heads, and this result is added to the residual stream.

The *per-head contribution* C^h directly reflects how each attention head modifies the residual stream. This contribution serves as the primary focus of our analysis, as it allows us to pinpoint attention heads that drive *short reasoning* behavior.

4.2 Identify Short Reasoning Attention Heads

For a response to problem i, let $\mathcal{T}_i$ be the set of token positions corresponding to the CoT, i.e., the tokens enclosed by `<think>` and `</think>` tags. Then, the overall average per-head contribution over all problems in the short reasoning dataset $\mathcal{D}_{\text{short}}$ is given by $\overline{C}^h \in \mathbb{R}^d$:

$$\overline{C}^h = \frac{1}{|\mathcal{D}_{\text{short}}|}\sum_{i\in\mathcal{D}_{\text{short}}}\left(\frac{1}{|\mathcal{T}_i|}\sum_{t\in\mathcal{T}_i} C^h(i,t)\right). \tag{4}$$

Equation 4 first averages the per-head contributions $C^h(i,t)$ over the CoT token positions for each problem i and then averages these values across all problems in $\mathcal{D}_{\text{short}}$. Recall that the reasoning length direction after an attention layer is defined as $v_\ell^{\text{attn}} = \overline{r}_{\ell,\text{long}}^{\text{attn}} - \overline{r}_{\ell,\text{short}}^{\text{attn}}$ in Equation 1, $v_\ell^{\text{attn}} \in \mathbb{R}^d$. To quantify the short reasoning contribution of head h, we project $\overline{C}^h$ onto the negative of the reasoning length direction (i.e., the short reasoning direction). Using the unit vector $\hat{v}_\ell^{\text{attn}} = \frac{v_\ell^{\text{attn}}}{\|v_\ell^{\text{attn}}\|}$, we define the scalar projection as $\overline{C}_{\text{short}}^h \in \mathbb{R}$:

$$\overline{C}_{\text{short}}^h = \left\langle \overline{C}^h, -\hat{v}_\ell^{\text{attn}} \right\rangle. \tag{5}$$

Here, $\overline{C}_{\text{short}}^h$ quantifies the degree to which head h's average contribution aligns with the short reasoning direction. Larger values of $\overline{C}_{\text{short}}^h$ indicate that the head strongly promotes short reasoning behavior. We visualize $\overline{C}_{\text{short}}^h$ for each attention head h with heatmap in Figure 4. Only a small subset of heads exhibits notably high alignment with the short reasoning direction, and these heads tend to cluster in the middle layers. This observation aligns with our analysis in section 3.3, where we found that reasoning length is primarily encoded in the middle layers. Crucially, the sparsity of these "short reasoning heads" suggests that it may be possible to effectively mitigate overly short reasoning behavior with minimal modifications to the model. In the following section, we use these insights to develop a targeted intervention ***ThinkEdit*** that removes short reasoning components while leaving the vast majority of the model's parameters unchanged.

4.3 Editing Short Reasoning Heads

We introduce how ***ThinkEdit*** effectively removes the short reasoning direction from the output projection matrices of the "short reasoning heads". Specifically, we identify the top 4% of attention heads with the largest $\overline{C}_{\text{short}}^h$ values (as defined in Section 4.2), marking them as short reasoning heads. Let $W_o^{h_\ell} \in \mathbb{R}^{d_h\times d}$ be the output projection matrix of head h in layer ℓ, and let $-\hat{v}_\ell^{\text{attn}} \in \mathbb{R}^d$ denote the *short reasoning direction* at layer ℓ. We then update $W_o^{h_\ell}$ via:

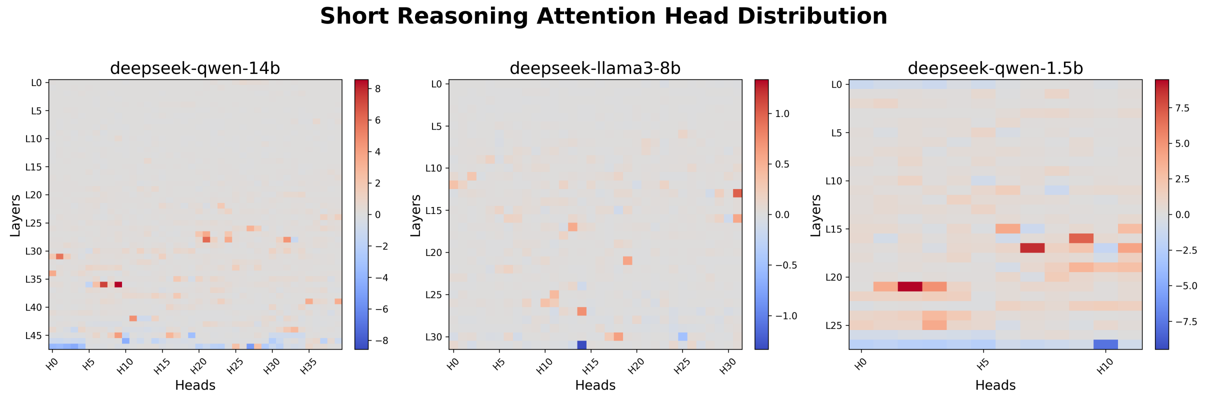

Figure 4: Heatmap illustrating the short reasoning contribution $\overline{C}^{h}_{\text{short}}$ for each attention head h. Heads with higher values (in red) show stronger alignment with short reasoning behavior.

$$W_o^{h_\ell} \leftarrow W_o^{h_\ell}\Big(I - \big(-\hat{v}_\ell^{\text{attn}}\big)\big(-\hat{v}_\ell^{\text{attn}}\big)^\top\Big), \quad (6)$$

where I is the $d \times d$ identity matrix. Intuitively, this operation projects each row of $W_o^{h_\ell}$ onto the subspace orthogonal to $-\hat{v}_\ell^{\text{attn}}$, thereby removes the short reasoning component from the head's output contribution. Unlike the approach in Section 3.3, which adds a fixed direction to activations regardless of the input, ***ThinkEdit*** modifies the weights of selected attention heads. This makes the adjustment input-dependent, allowing more precise control over reasoning length while preserving the model's overall behavior.

4.4 Performance of Reasoning Models after *ThinkEdit*

Experimental Setup. We evaluate the reasoning models after applying ***ThinkEdit*** on four mathematical reasoning benchmarks:

- **GSM8K** (Cobbe et al., 2021): A test set of 1,319 grade-school-level math word problems.
- **MMLU Elementary Math** (Hendrycks et al., 2021a): A subset of 378 elementary school math questions from the MMLU benchmark.
- **MATH-Level1**: A collection of 437 easy (Level 1) problems drawn from the MATH dataset (Hendrycks et al., 2021b).
- **MATH-Level5**: The most challenging subset of the MATH dataset with 1,324 problems.
- **MATH-500** (Lightman et al., 2023): A curated set of 500 high-quality math problems designed to assess advanced mathematical reasoning.

For all datasets, we set a maximum CoT length of 16,384 tokens. If this limit is reached, the model is prompted to immediately finalize its answer. To mitigate randomness, each dataset is evaluated over 10 independent runs, and the mean accuracy is reported. We do not include the phrase "Please reason step by step" in any prompt, aiming to assess the model's inherent reasoning capabilities.

Overall Accuracy. Table 1 reports the overall accuracy (in %) before and after applying ***ThinkEdit***. Across all math benchmarks, we observe consistent improvements in accuracy. Notably, the `deepseek-distill-qwen-1.5B` model shows a substantial gain on the MMLU Elementary Math subset. Manual inspection reveals that the unedited model occasionally ignores the multiple-choice format, leading to wrong answers. In contrast, the edited model adheres to the instructions more reliably. This suggests that ***ThinkEdit*** may not only enhance reasoning quality but also improve instruction-following behavior. On the more challenging MATH-Level5 and MATH-500 datasets, the accuracy gains are more modest but still positive, suggesting that while editing short-reasoning heads has a stronger impact on simpler problems, it might still provide meaningful improvements even for harder tasks that require longer and more complex reasoning chains.

Accuracy Under Short Reasoning. Table 2 shows the average accuracy for the top 5%, 10%, and 20% of responses with the shortest reasoning traces. After applying ***ThinkEdit***, we observe substantial accuracy improvements in these short-reasoning cases across most benchmarks. Interestingly, even for the challenging MATH-Level5 and MATH-500 datasets, short-reasoning accuracy improves noticeably. This suggests that ***ThinkEdit*** can effectively improve the reasoning quality when

Model		GSM8K	MMLU Elem. Math	MATH-Level1	MATH-Level5	MATH-500
deepseek-qwen-14B	Original	90.80 ± 0.36	95.08 ± 0.65	96.32 ± 0.35	90.25 ± 0.72	91.48 ± 0.55
	ThinkEdit (4%)	$\mathbf{93.78 \pm 0.50}$	$\mathbf{96.56 \pm 0.84}$	$\mathbf{96.36 \pm 0.52}$	$\mathbf{91.03 \pm 0.44}$	$\mathbf{91.92 \pm 0.63}$
deepseek-llama3-8B	Original	82.26 ± 0.91	96.01 ± 0.62	93.46 ± 0.84	85.49 ± 0.83	87.26 ± 1.16
	ThinkEdit (4%)	$\mathbf{89.44 \pm 0.55}$	$\mathbf{96.19 \pm 0.73}$	$\mathbf{94.44 \pm 0.31}$	$\mathbf{86.49 \pm 0.54}$	$\mathbf{88.06 \pm 1.09}$
deepseek-qwen-1.5B	Original	79.15 ± 1.08	68.52 ± 1.56	93.00 ± 0.33	$\mathbf{75.48 \pm 0.90}$	82.22 ± 1.29
	ThinkEdit (4%)	$\mathbf{84.56 \pm 0.79}$	$\mathbf{90.66 \pm 0.97}$	$\mathbf{93.66 \pm 0.62}$	75.05 ± 0.82	$\mathbf{82.24 \pm 0.89}$

Table 1: Overall accuracy (%) of each model before and after applying ThinkEdit.

Model		GSM8K	MMLU Elem. Math	MATH-Level1	MATH-Level5	MATH-500
deepseek-qwen-14b	Original	**96.31** / 95.65 / 92.93	93.89 / **96.22** / 95.60	**99.52** / 99.30 / 97.70	89.39 / 94.32 / 96.25	86.40 / 91.40 / 93.50
	ThinkEdit (4%)	**96.31 / 96.18 / 96.77**	**97.78** / 95.14 / **96.53**	**99.52 / 99.53 / 98.62**	**96.67 / 97.88 / 98.11**	**91.20 / 93.20 / 95.00**
deepseek-llama3-8b	Original	88.92 / 87.18 / 85.82	97.22 / 96.49 / 96.80	97.14 / 94.88 / 94.83	78.64 / 88.79 / 93.41	82.00 / 81.40 / 88.30
	ThinkEdit (4%)	**96.31 / 95.50 / 94.68**	**97.78 / 97.57 / 97.60**	**99.05 / 99.07 / 97.82**	**95.76 / 97.42 / 97.46**	**95.60 / 93.80 / 95.40**
deepseek-qwen-1.5b	Original	88.46 / 87.48 / 85.02	62.78 / 62.16 / 60.53	**97.62** / 95.12 / 93.91	91.52 / 95.00 / 95.72	82.40 / 89.80 / 93.40
	ThinkEdit (4%)	**92.62 / 92.90 / 92.32**	**87.78 / 88.11 / 88.67**	95.71 / **95.58 / 96.44**	**95.15 / 96.59 / 97.27**	**90.80 / 92.00 / 94.20**

Table 2: Accuracy (%) of the top 5% / 10% / 20% shortest reasoning responses.

the models generate short CoT.

Reasoning Length of the Shortest Responses. We analyze how ***ThinkEdit*** affects reasoning length in Appendix A.3. It modestly increases the length of the shortest responses (Table 3), helping to address overly brief reasoning. However, as shown in Table 4, the overall reasoning length remains largely stable across datasets, with a net change of -0.27% across all models and datasets.

In summary, ***ThinkEdit*** markedly improves model performance on short-reasoning instances and yields a substantial overall accuracy gain. We also explore different editing percentages and compare our approach to simply appending "Wait" to prompt longer reasoning; detailed results are provided in Appendix A.4. Additionally, results for `deepseek-distill-qwen-32B` are reported in Appendix A.5. To test the generality of our method, we further evaluate on non-math domains in Appendix A.6. Beyond demonstrating accuracy gains, Appendix A.7 examines how ***ThinkEdit*** shapes reasoning behaviors, revealing more explicit and structured chains of thought. Finally, We provide several concrete examples illustrating how ***ThinkEdit*** enhances reasoning quality in Appendix A.9.

5 Related Works

Reasoning Models. Recent advances in reasoning models have significantly improved the problem-solving abilities of LLMs in domains such as mathematics, coding, and science. OpenAI's `o1` (Jaech et al., 2024) represents a major shift toward deliberate reasoning by employing reinforcement learning (RL) to refine its strategies. By generating explicit "Thinking" steps before producing answers, `o1` achieves strong performance on complex tasks. As a more cost-efficient alternative, DeepSeek-`r1` (Guo et al., 2025) demonstrates that pure RL can also effectively enhance reasoning. It introduces Group Relative Policy Optimization (GRPO) (Shao et al., 2024), a novel method that eliminates the need for a separate reward model, enabling more efficient RL training.

Controllable Text Generation. Controllable text generation has been explored across various domains (Liang et al., 2024), with methods generally classified into training-time and inference-time control. These approaches aim to steer LLMs toward generating text with specific attributes while preserving fluency and coherence. Training-time control is achieved through fine-tuning (Zeldes et al., 2020; Wei et al., 2022) or reinforcement learning (Ouyang et al., 2022; Rafailov et al., 2023), leveraging diverse datasets to shape model behavior. Inference-time control encompasses prompt engineering (Shin et al., 2020; Li and Liang, 2021), representation engineering (Subramani et al., 2022; Zou et al., 2023a; Konen et al., 2024; Oikarinen et al., 2025), interpretable neuron intervention through concept bottleneck models (Sun et al., 2025b, 2024), and decoding-time interventions (Dathathri et al., 2020), allowing flexible and efficient control without retraining.

In this work, we focus on the representation engineering paradigm to investigate how reasoning length is embedded within model representations. Specifically, we introduce a linear "reasoning-length direction" in the representation space to examine its impact on reasoning capabilities.

Attention heads and MLP neurons intervention. A growing body of research explores the role of attention heads and neurons within the Multi-Layer Perceptron (MLP) layers in shaping language model behavior. Studies such as (Zhou et al., 2025; Zhao et al., 2025; Chen et al., 2024) examine how safety mechanisms are embedded in well-aligned models to defend against harmful prompts and jailbreak attacks (Zou et al., 2023b; Liu et al., 2024; Sun et al., 2025a). Findings indicate that a small subset of attention heads and MLP neurons play a critical role in safety alignments. Similarly, research on hallucination mitigation has investigated the contributions of attention heads and MLP neurons. (Ho et al., 2025) demonstrates that filtering out unreliable attention heads can reduce erroneous generations, while (Yu et al., 2024) finds that activating subject-knowledge neurons helps maintain factual consistency. In (Li et al., 2025), the authors design efficient and training-free machine skill unlearning techniques for LLMs through intervention and abstention.

In our work, we investigate how attention heads relate to reasoning processes, examining their impact on the reasoning length and quality.

6 Conclusion

In this work, we first identified overly short reasoning as a common failure mode in Deepseek-distilled reasoning models. To understand how reasoning length is controlled, we analyzed the model's hidden representations and uncovered a latent direction linked to reasoning length. Building on this, we pinpointed 4% of attention heads that drive short reasoning, and propose ***ThinkEdit*** to mitigate the issue, leading to significant accuracy gains for short reasoning outputs (+6.39%), along with an overall improvement (+3.34%) across multiple math benchmarks.

Limitations

A limitation of our work is that ***ThinkEdit*** primarily improves model performance by addressing cases of overly short reasoning. For reasoning models that already tend to produce sufficiently long or verbose outputs, the benefits of ***ThinkEdit*** may be limited. Nonetheless, our study provides valuable insights by highlighting the often-overlooked issue of overly brief reasoning and examining how reasoning length is represented within the model's hidden states. This opens an important research direction for advancing the interpretability of reasoning models by linking internal representations to observable reasoning behaviors.

Acknowledgement

This research was supported by grants from NVIDIA and utilized NVIDIA's A100 GPUs on Saturn Cloud. The authors are partially supported by Hellman Fellowship, Intel Rising Star Faculty Award, and National Science Foundation under Grant No. 2313105, 2430539.

References

Jianhui Chen, Xiaozhi Wang, Zijun Yao, Yushi Bai, Lei Hou, and Juanzi Li. 2024. Finding safety neurons in large language models. *CoRR*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. Plug and play language models: A simple approach to controlled text generation. In *ICLR*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. In *ICLR*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Zheng Yi Ho, Siyuan Liang, Sen Zhang, Yibing Zhan, and Dacheng Tao. 2025. Novo: Norm voting off hallucinations with attention heads in large language models. *ICLR*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, and 80 others. 2024. Openai o1 system card. *CoRR*.

Kai Konen, Sophie Jentzsch, Diaoulé Diallo, Peer Schütt, Oliver Bensch, Roxanne El Baff, Dominik Opitz, and Tobias Hecking. 2024. Style vectors for steering generative large language models. In *EACL Findings*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *ACL/IJCNLP*.

Yongce Li, Chung-En Sun, and Tsui-Wei Weng. 2025. Effective skill unlearning through intervention and abstention. In *NAACL*.

Xun Liang, Hanyu Wang, Yezhaohui Wang, Shichao Song, Jiawei Yang, Simin Niu, Jie Hu, Dan Liu, Shunyu Yao, Feiyu Xiong, and Zhiyu Li. 2024. Controllable text generation for large language models: A survey. *CoRR*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. 2024. Autodan: Generating stealthy jailbreak prompts on aligned large language models. In *ICLR*.

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.

Tuomas Oikarinen, Ge Yan, and Tsui-Wei Weng. 2025. Evaluating neuron explanations: A unified framework with sanity checks. In *ICML*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *NeurIPS*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *NeurIPS*.

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

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *EMNLP*.

Nishant Subramani, Nivedita Suresh, and Matthew E. Peters. 2022. Extracting latent steering vectors from pretrained language models. In *ACL Findings*.

Chung-En Sun, Xiaodong Liu, Weiwei Yang, Tsui-Wei Weng, Hao Cheng, Aidan San, Michel Galley, and Jianfeng Gao. 2025a. Iterative self-tuning llms for enhanced jailbreaking capabilities. In *NAACL*.

Chung-En Sun, Tuomas Oikarinen, Berk Ustun, and Tsui-Wei Weng. 2025b. Concept bottleneck large language models. In *ICLR*.

Chung-En Sun, Tuomas Oikarinen, and Tsui-Wei Weng. 2024. Crafting large language models for enhanced interpretability. In *ICML 2024 Workshop on Mechanistic Interpretability*.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned language models are zero-shot learners. In *ICLR*.

Lei Yu, Meng Cao, Jackie C. K. Cheung, and Yue Dong. 2024. Mechanistic understanding and mitigation of language model non-factual hallucinations. In *EMNLP Findings*.

Yoel Zeldes, Dan Padnos, Or Sharir, and Barak Peleg. 2020. Technical report: Auxiliary tuning and its application to conditional text generation. *CoRR*.

Yiran Zhao, Wenxuan Zhang, Yuxi Xie, Anirudh Goyal, Kenji Kawaguchi, and Michael Shieh. 2025. Understanding and enhancing safety mechanisms of llms via safety-specific neuron. *ICLR*.

Zhenhong Zhou, Haiyang Yu, Xinghua Zhang, Rongwu Xu, Fei Huang, Kun Wang, Yang Liu, Junfeng Fang, and Yongbin Li. 2025. On the role of attention heads in large language model safety. *ICLR*.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, Shashwat Goel, Nathaniel Li, Michael J. Byun, Zifan Wang, Alex Mallen, Steven Basart, Sanmi Koyejo, Dawn Song, Matt Fredrikson, and 2 others. 2023a. Representation engineering: A top-down approach to AI transparency. *CoRR*.

Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. 2023b. Universal and transferable adversarial attacks on aligned language models. *CoRR*.

Table of Contents

A Appendix

A.1 Gobal Steering with the MLP-based Direction v_ℓ^{mlp}

Figure 5 replicates the global steering analysis using the MLP-based direction v_ℓ^{mlp}. The observed trends closely mirror those from attention-based steering: increasing α extends reasoning length across both datasets, and the effect on accuracy is model- and dataset-dependent. On GSM8K, larger models benefit from longer reasoning, while smaller models degrade. On MATH-Level5, overly long reasoning may hurt performance, despite consistent control over CoT length. These results show that both attention and MLP directions capture similar reasoning-length attributes.

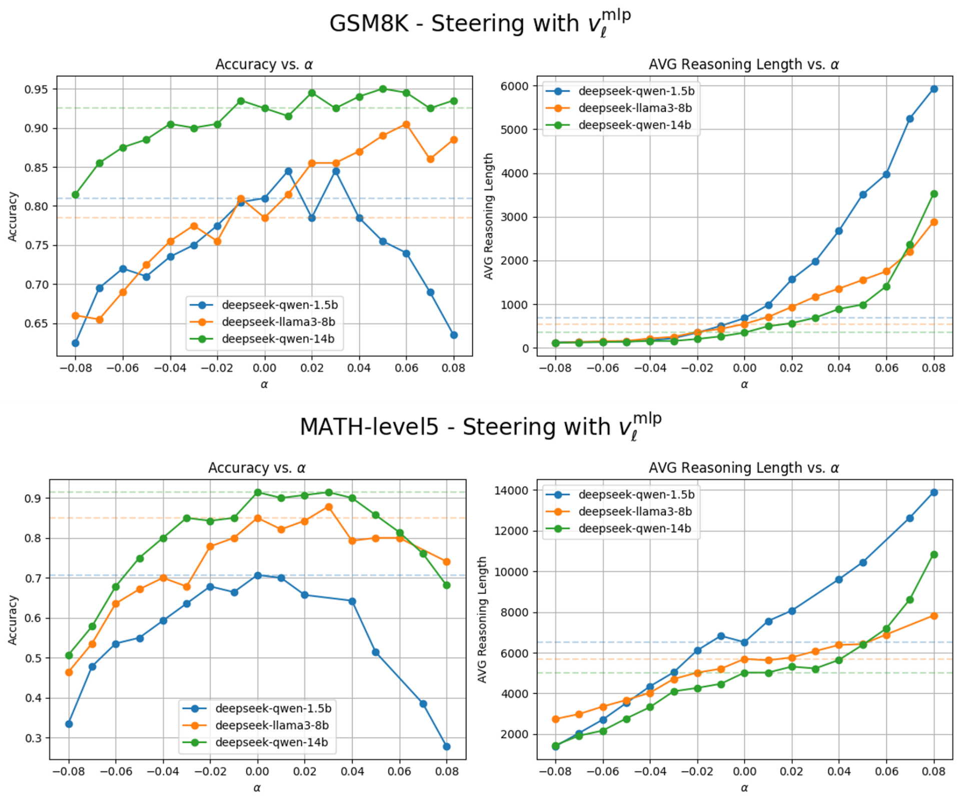

Figure 5: Global steering with the reasoning length direction extracted from MLPs. The trend is similar as steering with attention-based directions.

A.2 Layerwise Analysis of Steering along Reasoning Length Direction

To identify which layers are most influenced by the reasoning-length direction, we perform a *layerwise* experiment on the GSM8K dataset (Figure 6). Specifically, we use v_ℓ^{mlp} to steer each MLP layer *separately*, applying $\alpha = \pm 1$ at a single layer ℓ. Notably, the *middle layers* elicit the largest fluctuations, suggesting they encode key representations for controlling reasoning length.

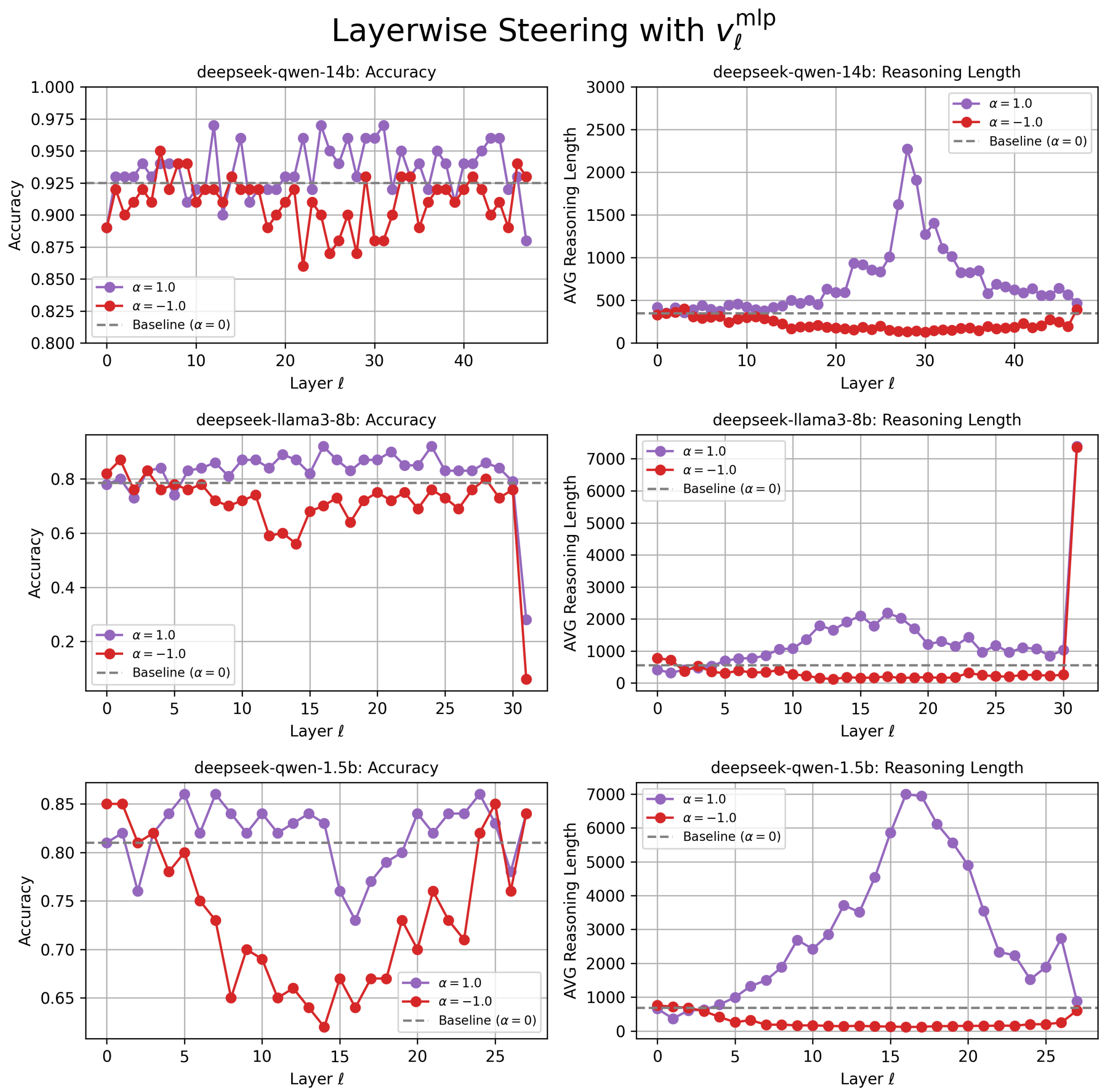

Figure 6: Layerwise steering on GSM8K with v_ℓ^{mlp}. We apply $\alpha = \pm 1$ to one layer at a time, revealing that the middle layers wield the strongest control over reasoning length and accuracy.

A.3 The Impact of ThinkEdit on Reasoning Length

Table 3 reports the average reasoning length among the top 5%, 10%, and 20% shortest responses. We observe that ***ThinkEdit*** consistently increases the reasoning length in these short-answer scenarios, suggesting that the intervention discourages excessively short reasoning, and therefore leads to higher accuracy as shown in Table 2. Interestingly, Table 4 shows that the average reasoning length remains similar between the original and ***ThinkEdit*** models. To summarize these trends, we compute the average change in reasoning length across all datasets: +2.94% for `deepseek-qwen-14b`, +3.53% for `deepseek-llama3-8b`, and -5.73% for `deepseek-qwen-1.5b`, resulting in an overall average change of -0.27%. These results suggest that ***ThinkEdit*** selectively increases reasoning length for short responses without significantly altering overall response length.

Model		GSM8K	MMLU Elem. Math	MATH-Level1	MATH-Level5	MATH-500
deepseek-qwen-14B	Original	76.6 / 86.5 / 99.1	65.8 / 72.2 / 80.6	93.7 / 114.3 / 188.6	628.8 / 858.4 / 1125.9	198.7 / 434.3 / 697.0
	ThinkEdit (4%)	**101.7 / 113.6 / 131.0**	**82.7 / 91.8 / 105.6**	**146.7 / 188.6 / 346.0**	**745.5 / 926.6 / 1163.7**	**361.3 / 559.3 / 764.6**
deepseek-llama3-8B	Original	73.0 / 83.1 / 96.6	371.0 / 438.1 / 518.2	80.3 / 97.2 / 130.3	617.9 / 854.9 / 1126.5	159.5 / 357.5 / 644.5
	ThinkEdit (4%)	**110.3 / 131.8 / 164.6**	**398.5 / 462.4 / 541.8**	**176.3 / 221.3 / 336.0**	**806.1 / 963.3 / 1185.1**	**372.5 / 553.2 / 772.9**
deepseek-qwen-1.5B	Original	78.8 / 89.4 / 103.0	61.6 / 68.5 / 77.6	88.8 / 110.3 / 219.7	804.6 / **1017.9 / 1314.0**	249.7 / 506.5 / 760.7
	ThinkEdit (4%)	**103.3 / 118.9 / 144.8**	**80.6 / 92.5 / 112.9**	**172.7 / 336.9 / 543.6**	**853.0** / 1003.5 / 1221.9	**530.8 / 676.0 / 837.4**

Table 3: Average reasoning length for the top 5% / 10% / 20% shortest responses (in tokens).

Model		GSM8K	MMLU Elem. Math	MATH-Level1	MATH-Level5	MATH-500
deepseek-qwen-14B	Original	354.5 ± 684.4	184.9 ± 175.3	1600.5 ± 1885.2	$\mathbf{4306.2 \pm 3816.1}$	$\mathbf{3096.8 \pm 3308.0}$
	ThinkEdit (4%)	$\mathbf{538.2 \pm 829.6}$	$\mathbf{291.4 \pm 607.5}$	$\mathbf{1670.4 \pm 1951.2}$	4243.7 ± 3814.0	3079.7 ± 3276.6
deepseek-llama3-8B	Original	597.3 ± 1109.0	1486.6 ± 2036.7	1646.6 ± 2275.0	$\mathbf{4789.1 \pm 4315.4}$	3507.6 ± 3917.5
	ThinkEdit (4%)	$\mathbf{927.7 \pm 1486.3}$	$\mathbf{1517.9 \pm 2041.5}$	$\mathbf{1723.7 \pm 2152.3}$	4773.5 ± 4327.4	$\mathbf{3509.5 \pm 3842.9}$
deepseek-qwen-1.5B	Original	768.1 ± 1837.2	517.0 ± 1502.8	$\mathbf{2080.9 \pm 2740.5}$	$\mathbf{6360.0 \pm 5336.4}$	$\mathbf{4260.3 \pm 4668.2}$
	ThinkEdit (4%)	$\mathbf{1126.6 \pm 2018.0}$	$\mathbf{768.9 \pm 1651.4}$	1946.3 ± 2438.4	5522.4 ± 5036.9	3821.1 ± 4384.9

Table 4: Overall reasoning length (in tokens) before and after applying ThinkEdit (4% edit rate).

A.4 ThinkEdit with Varying Editing Rates vs. the "Wait" Appending Baseline

We conduct a comprehensive evaluation of ***ThinkEdit*** with different editing rates and compare it against a simple baseline that appends the word "Wait" to reasoning sequences shorter than 500 tokens. This baseline is intended to prompt the model to continue thinking before answering when the reasoning is too short. The methods compared are:

- ***ThinkEdit (8%)***: Edits 8% of total attention heads.
- ***ThinkEdit (4%)***: Edits 4% of total attention heads.
- ***ThinkEdit (2%)***: Edits 2% of total attention heads.
- **Append "Wait"**: Appends "Wait" to reasoning with fewer than 500 tokens to artificially extend reasoning length.
- **Original**: The unmodified model output.

As shown in Table 5, higher editing rates in ***ThinkEdit*** consistently improve performance on GSM8K and MMLU Elementary Math. However, for the MATH-series datasets, moderate editing rates yield better results than the most aggressive edits. The "Append Wait" baseline yields only marginal gains across all datasets, indicating that simply forcing the model to produce longer reasoning does not necessarily improve accuracy. A closer look at the short reasoning cases in Table 7 shows that all versions of ***ThinkEdit*** substantially outperform the "Append Wait" baseline. This further supports the observation that longer reasoning alone is insufficient without proper internal adjustment of the model.

In terms of reasoning length (Tables 6 and 8), the "Append Wait" method generally leads to longer outputs than ***ThinkEdit (2%)***, confirming that it effectively increases response length. However, despite this, it fails to match the performance of ***ThinkEdit***, highlighting that ***ThinkEdit*** is a more effective strategy addressing the accuracy drops of overly short reasoning.

Model		GSM8K	MMLU Elem. Math	MATH-Level1	MATH-Level5	MATH-500
deepseek-qwen-14B	ThinkEdit (8%)	$\mathbf{94.30 \pm 0.28}$	$\mathbf{96.93 \pm 0.50}$	96.09 ± 0.35	90.92 ± 0.41	91.26 ± 0.52
	ThinkEdit (4%)	93.78 ± 0.50	96.56 ± 0.84	96.36 ± 0.52	91.03 ± 0.44	$\mathbf{91.92 \pm 0.63}$
	ThinkEdit (2%)	93.50 ± 0.31	96.53 ± 0.54	96.50 ± 0.46	$\mathbf{91.15 \pm 0.59}$	91.78 ± 0.58
	Append "Wait"	91.30 ± 0.55	95.37 ± 0.70	$\mathbf{96.52 \pm 0.55}$	90.60 ± 0.41	91.22 ± 0.57
	Original	90.80 ± 0.36	95.08 ± 0.65	96.32 ± 0.35	90.25 ± 0.72	91.48 ± 0.55
deepseek-llama3-8B	ThinkEdit (8%)	$\mathbf{90.18 \pm 0.60}$	96.11 ± 0.67	94.39 ± 0.61	86.13 ± 0.46	87.64 ± 0.88
	ThinkEdit (4%)	89.44 ± 0.55	$\mathbf{96.19 \pm 0.73}$	$\mathbf{94.44 \pm 0.31}$	$\mathbf{86.49 \pm 0.54}$	$\mathbf{88.06 \pm 1.09}$
	ThinkEdit (2%)	88.97 ± 0.78	96.08 ± 0.86	94.12 ± 0.47	85.91 ± 0.48	87.60 ± 0.81
	Append "Wait"	84.28 ± 0.64	95.93 ± 0.70	93.96 ± 0.55	85.33 ± 0.79	87.66 ± 1.26
	Original	82.26 ± 0.91	96.01 ± 0.62	93.46 ± 0.84	85.49 ± 0.83	87.26 ± 1.16
deepseek-qwen-1.5B	ThinkEdit (8%)	$\mathbf{84.81 \pm 0.69}$	$\mathbf{92.38 \pm 1.04}$	$\mathbf{94.00 \pm 0.75}$	75.32 ± 1.11	82.56 ± 1.21
	ThinkEdit (4%)	84.56 ± 0.79	90.66 ± 0.97	93.66 ± 0.62	75.05 ± 0.82	82.24 ± 0.89
	ThinkEdit (2%)	83.34 ± 0.79	86.24 ± 1.12	93.89 ± 0.76	74.94 ± 0.85	82.74 ± 0.77
	Append "Wait"	79.81 ± 0.77	76.64 ± 1.18	93.34 ± 0.86	75.06 ± 0.72	$\mathbf{82.98 \pm 1.00}$
	Original	79.15 ± 1.08	68.52 ± 1.56	93.00 ± 0.33	$\mathbf{75.48 \pm 0.90}$	82.22 ± 1.29

Table 5: Overall accuracy (%) of ThinkEdit with different editing rates.

Model		GSM8K	MMLU Elem. Math	MATH-Level1	MATH-Level5	MATH-500
deepseek-qwen-14B	ThinkEdit (8%)	$\mathbf{598.1 \pm 1011.8}$	$\mathbf{336.6 \pm 550.3}$	1586.1 ± 1827.4	4150.5 ± 3819.1	3009.5 ± 3336.7
	ThinkEdit (4%)	538.2 ± 829.6	291.4 ± 607.5	$\mathbf{1670.4 \pm 1951.2}$	4243.7 ± 3814.0	3079.7 ± 3276.6
	ThinkEdit (2%)	479.8 ± 968.5	285.1 ± 756.8	1645.4 ± 1946.6	$\mathbf{4327.2 \pm 3863.4}$	$\mathbf{3138.3 \pm 3372.8}$
	Append "Wait"	447.3 ± 652.6	273.0 ± 215.8	1595.8 ± 1810.5	4265.9 ± 3749.0	3071.5 ± 3275.6
	Original	354.5 ± 684.4	184.9 ± 175.3	1600.5 ± 1885.2	4306.2 ± 3816.1	3096.8 ± 3308.0
deepseek-llama3-8B	ThinkEdit (8%)	$\mathbf{971.8 \pm 1447.7}$	1488.3 ± 1979.5	1692.8 ± 2200.5	4642.1 ± 4253.3	3463.3 ± 3800.1
	ThinkEdit (4%)	927.7 ± 1486.3	1517.9 ± 2041.5	1723.7 ± 2152.3	4773.5 ± 4327.4	$\mathbf{3509.5 \pm 3842.9}$
	ThinkEdit (2%)	849.7 ± 1454.8	$\mathbf{1520.1 \pm 2103.0}$	$\mathbf{1765.7 \pm 2315.1}$	$\mathbf{4825.2 \pm 4383.4}$	3503.8 ± 3838.4
	Append "Wait"	670.2 ± 1073.0	1514.4 ± 2009.1	1639.9 ± 2134.8	4795.3 ± 4296.2	3502.5 ± 3859.1
	Original	597.3 ± 1109.0	1486.6 ± 2036.7	1646.6 ± 2275.0	4789.1 ± 4315.4	3507.6 ± 3917.5
deepseek-qwen-1.5B	ThinkEdit (8%)	$\mathbf{1166.2 \pm 1986.4}$	$\mathbf{890.7 \pm 1851.7}$	1912.8 ± 2287.6	5567.4 ± 5083.4	3772.6 ± 4296.0
	ThinkEdit (4%)	1126.6 ± 2018.0	768.9 ± 1651.4	1946.3 ± 2438.4	5522.4 ± 5036.9	3821.1 ± 4384.9
	ThinkEdit (2%)	912.7 ± 1835.3	701.0 ± 1748.9	1918.0 ± 2420.6	5641.9 ± 5101.5	3880.3 ± 4402.4
	Append "Wait"	847.1 ± 1835.7	660.1 ± 1823.7	$\mathbf{2163.7 \pm 2847.0}$	$\mathbf{6363.9 \pm 5352.9}$	$\mathbf{4287.1 \pm 4710.3}$
	Original	768.1 ± 1837.2	517.0 ± 1502.8	2080.9 ± 2740.5	6360.0 ± 5336.4	4260.3 ± 4668.2

Table 6: Overall reasoning length (in tokens) of ThinkEdit with different editing rates.

Model		GSM8K	MMLU Elem. Math	MATH-Level1	MATH-Level5	MATH-500
deepseek-qwen-14B	ThinkEdit (8%)	96.46 / **97.02** / **97.38**	97.22 / 95.95 / 95.73	98.57 / 97.91 / 97.47	**98.48** / **98.56** / **98.22**	**91.60** / 93.00 / 94.60
	ThinkEdit (4%)	96.31 / 96.18 / 96.77	**97.78** / 95.14 / 96.53	99.52 / 99.53 / 98.62	96.67 / 97.88 / 98.11	91.20 / **93.20** / **95.00**
	ThinkEdit (2%)	**96.62** / 96.03 / 96.12	96.11 / 96.22 / 96.27	**100.00** / **99.77** / **98.85**	95.76 / 97.65 / 98.07	89.60 / 92.60 / 94.70
	Append "Wait"	94.62 / 94.20 / 93.19	96.67 / **97.30** / **96.93**	**100.00** / 99.30 / 98.39	90.15 / 94.47 / 96.33	85.20 / 89.20 / 93.30
	Original	96.31 / 95.65 / 92.93	93.89 / 96.22 / 95.60	99.52 / 99.30 / 97.70	89.39 / 94.32 / 96.25	86.40 / 91.40 / 93.50
deepseek-llama3-8B	ThinkEdit (8%)	96.31 / **96.49** / **95.97**	**97.78** / 97.57 / **98.40**	99.05 / **99.30** / 98.85	**97.12** / **97.58** / 97.39	95.20 / **94.20** / 94.80
	ThinkEdit (4%)	96.31 / 95.50 / 94.68	**97.78** / 97.57 / 97.60	99.05 / 99.07 / 97.82	95.76 / 97.42 / **97.46**	**95.60** / 93.80 / **95.40**
	ThinkEdit (2%)	**97.08** / 95.27 / 93.95	**97.78** / **98.65** / 97.87	**100.00** / **99.30** / 98.62	95.61 / 96.89 / 97.12	92.80 / 93.60 / 94.40
	Append "Wait"	88.15 / 89.01 / 88.29	**97.78** / 97.57 / 97.87	98.57 / 97.21 / 95.75	79.55 / 89.02 / 93.45	86.40 / 86.00 / 90.70
	Original	88.92 / 87.18 / 85.82	97.22 / 96.49 / 96.80	97.14 / 94.88 / 94.83	78.64 / 88.79 / 93.41	82.00 / 81.40 / 88.30
deepseek-qwen-1.5B	ThinkEdit (8%)	**95.38** / **94.20** / **92.97**	**93.89** / **92.70** / **91.87**	94.76 / 96.05 / 96.90	**96.21** / **97.20** / 96.78	**94.00** / 93.60 / 94.40
	ThinkEdit (4%)	92.62 / 92.90 / 92.32	87.78 / 88.11 / 88.67	95.71 / 95.58 / 96.44	95.15 / 96.59 / **97.27**	90.80 / 92.00 / 94.20
	ThinkEdit (2%)	92.46 / 92.37 / 92.05	77.22 / 80.54 / 79.73	96.19 / 95.81 / **97.36**	93.79 / 95.83 / 95.80	92.80 / **94.40** / **94.90**
	Append "Wait"	88.92 / 87.10 / 86.77	82.22 / 79.46 / 76.13	96.67 / **96.74** / 96.44	92.27 / 94.85 / 95.72	86.00 / 90.60 / 92.30
	Original	88.46 / 87.48 / 85.02	62.78 / 62.16 / 60.53	**97.62** / 95.12 / 93.91	91.52 / 95.00 / 95.72	82.40 / 89.80 / 93.40

Table 7: Accuracy (%) on the top 5% / 10% / 20% shortest responses for ThinkEdit with different editing rates.

Model		GSM8K	MMLU Elem. Math	MATH-Lvl1	MATH-Lvl5	MATH-500
deepseek-qwen-14B	ThinkEdit (8%)	113.2 / 129.4 / 153.6	86.9 / 99.0 / 117.2	**180.7** / **238.5** / **372.3**	**768.1** / 925.6 / 1136.0	**414.7** / **573.9** / 759.0
	ThinkEdit (4%)	101.7 / 113.6 / 131.0	82.7 / 91.8 / 105.6	146.7 / 188.6 / 346.0	745.5 / **926.6** / **1163.7**	361.3 / 559.3 / **764.6**
	ThinkEdit (2%)	95.4 / 106.3 / 120.2	79.1 / 87.1 / 98.7	125.1 / 150.2 / 243.4	698.5 / 906.6 / 1157.2	270.2 / 492.6 / 733.3
	Wait	**127.2** / **145.0** / **166.0**	**104.1** / **114.4** / **127.6**	159.3 / 191.8 / 281.9	672.1 / 875.5 / 1132.1	293.6 / 495.7 / 720.6
	Original	76.6 / 86.5 / 99.1	65.8 / 72.2 / 80.6	93.7 / 114.3 / 188.6	628.8 / 858.4 / 1125.9	198.7 / 434.3 / 697.0
deepseek-llama3-8B	ThinkEdit (8%)	**160.4** / **185.7** / **225.2**	426.0 / 484.4 / 559.4	**209.5** / **267.2** / **380.8**	**825.3** / **978.8** / **1190.7**	**422.6** / **567.4** / 759.5
	ThinkEdit (4%)	110.3 / 131.8 / 164.6	398.5 / 462.4 / 541.8	176.3 / 221.3 / 336.0	806.1 / 963.3 / 1185.1	372.5 / 553.2 / **772.9**
	ThinkEdit (2%)	93.2 / 106.9 / 127.4	396.5 / 464.2 / 543.2	137.4 / 173.3 / 277.1	791.2 / 954.8 / 1185.1	305.2 / 506.3 / 737.6
	Wait	132.2 / 148.2 / 169.1	**444.5** / **501.7** / **565.9**	148.4 / 179.2 / 244.0	680.8 / 887.3 / 1147.1	277.9 / 452.1 / 693.5
	Original	73.0 / 83.1 / 96.6	371.0 / 438.1 / 518.2	80.3 / 97.2 / 130.3	617.9 / 854.9 / 1126.5	159.5 / 357.5 / 644.5
deepseek-qwen-1.5B	ThinkEdit (8%)	115.9 / **138.2** / **180.1**	87.4 / 103.7 / **130.1**	**247.3** / **396.1** / **577.3**	**859.4** / 1003.7 / 1216.6	**595.9** / **719.8** / **871.6**
	ThinkEdit (4%)	103.3 / 118.9 / 144.8	80.6 / 92.5 / 112.9	172.7 / 336.9 / 543.6	853.0 / 1003.5 / 1221.9	530.8 / 676.0 / 837.4
	ThinkEdit (2%)	97.2 / 109.4 / 126.3	75.9 / 85.0 / 99.5	127.9 / 174.1 / 416.4	818.0 / 984.5 / 1214.3	435.0 / 612.9 / 800.6
	Wait	**120.6** / 137.0 / 158.0	**101.6** / **112.9** / 128.0	147.9 / 180.1 / 310.2	822.7 / **1020.9** / 1306.0	341.8 / 556.6 / 791.8
	Original	78.8 / 89.4 / 103.0	61.6 / 68.5 / 77.6	88.8 / 110.3 / 219.7	804.6 / 1017.9 / **1314.0**	249.7 / 506.5 / 760.7

Table 8: Average reasoning length (in tokens) of the top 5% / 10% / 20% shortest responses for ThinkEdit with different editing rates.

A.5 ThinkEdit Results for 32B Reasoning Model

We report results for the larger `deepseek-distill-qwen-32B` model. Although ThinkEdit does not yield overall accuracy gains on the MATH-series datasets (Table 9), it consistently achieves higher accuracy on short reasoning responses similar to the smaller models (Table 11).

Model		GSM8K	MMLU Elem. Math	MATH-Level1	MATH-Level5	MATH-500
deepseek-qwen-32B	ThinkEdit (8%)	$\mathbf{95.34 \pm 0.23}$	$\mathbf{98.10 \pm 0.17}$	95.31 ± 0.38	91.16 ± 0.45	91.44 ± 0.57
	ThinkEdit (4%)	95.25 ± 0.25	98.02 ± 0.31	96.02 ± 0.42	91.31 ± 0.50	91.60 ± 0.65
	ThinkEdit (2%)	94.90 ± 0.34	97.49 ± 0.49	96.27 ± 0.54	91.31 ± 0.29	$\mathbf{91.62 \pm 0.74}$
	Append "Wait"	92.72 ± 0.54	96.16 ± 0.45	96.27 ± 0.39	$\mathbf{91.32 \pm 0.46}$	91.46 ± 0.51
	Original	92.97 ± 0.39	95.93 ± 0.83	$\mathbf{96.41 \pm 0.45}$	91.27 ± 0.53	$\mathbf{91.62 \pm 0.58}$

Table 9: Overall accuracy (%) of `deepseek-distill-qwen-32B` with different ThinkEdit edit-rates.

Model		GSM8K	MMLU Elem. Math	MATH-Level1	MATH-Level5	MATH-500
deepseek-qwen-32B	ThinkEdit (8%)	$\mathbf{665.6 \pm 762.8}$	$\mathbf{312.3 \pm 332.0}$	$\mathbf{1548.6 \pm 1473.4}$	3676.7 ± 3388.7	2665.6 ± 2885.1
	ThinkEdit (4%)	445.8 ± 684.7	287.7 ± 600.0	1484.7 ± 1587.7	3821.1 ± 3518.3	2736.4 ± 2948.8
	ThinkEdit (2%)	405.3 ± 620.5	238.8 ± 315.9	1485.3 ± 1622.1	3947.0 ± 3564.7	2816.1 ± 3029.2
	Append "Wait"	405.5 ± 519.0	280.6 ± 401.5	1484.8 ± 1619.1	$\mathbf{4103.9 \pm 3606.0}$	$\mathbf{2878.8 \pm 3029.3}$
	Original	306.2 ± 515.4	168.9 ± 105.3	1457.6 ± 1621.0	4100.7 ± 3608.7	2872.0 ± 3034.8

Table 10: Overall reasoning length (in tokens) for `deepseek-distill-qwen-32B`.

Model		GSM8K	MMLU Elem. Math	MATH-Level1	MATH-Level5	MATH-500
deepseek-qwen-32B	ThinkEdit (8%)	**99.08 / 98.47 / 97.95**	**98.33 / 97.57** / 97.07	99.52 / 98.60 / 97.36	**99.55 / 99.39 / 98.64**	**94.40 / 95.40 / 96.10**
	ThinkEdit (4%)	98.92 / 97.71 / 97.83	97.78 / **97.57 / 97.20**	**100.00 / 100.00** / 98.74	98.03 / 98.64 / 97.99	92.00 / 94.40 / 95.80
	ThinkEdit (2%)	98.92 / 98.24 / 97.68	96.67 / 97.03 / 96.80	99.05 / 98.84 / 98.51	97.58 / 98.26 / 98.22	90.00 / 92.60 / 94.70
	Append "Wait"	97.08 / 96.03 / 95.21	95.00 / 96.76 / 96.27	99.52 / 99.30 / 98.05	94.09 / 96.89 / 97.61	84.80 / 90.40 / 93.20
	Original	98.31 / 97.18 / 96.20	97.78 / 97.03 / 95.87	**100.00 / 100.00 / 98.97**	93.03 / 96.36 / 97.35	86.40 / 92.00 / 94.00

Table 11: Accuracy (%) on the top 5% / 10% / 20% shortest responses for `deepseek-distill-qwen-32B`.

Model		GSM8K	MMLU Elem. Math	MATH-Lvl1	MATH-Lvl5	MATH-500
deepseek-qwen-32B	ThinkEdit (8%)	105.2 / 121.8 / 148.6	89.2 / 100.5 / 117.7	**367.8 / 492.8 / 625.4**	**793.5 / 919.5** / 1094.6	**567.1 / 677.0 / 811.1**
	ThinkEdit (4%)	95.2 / 105.8 / 120.1	85.9 / 96.1 / 110.6	146.9 / 202.2 / 360.9	751.1 / 905.4 / 1101.0	446.7 / 600.0 / 768.9
	ThinkEdit (2%)	93.2 / 103.6 / 116.6	79.1 / 88.6 / 101.5	124.3 / 155.3 / 307.6	746.4 / 910.8 / 1123.7	371.3 / 563.0 / 759.8
	Append "Wait"	**125.7 / 143.0 / 163.7**	**109.6 / 121.1 / 135.9**	151.4 / 182.0 / 247.2	725.7 / 914.4 / **1153.4**	328.4 / 521.3 / 739.4
	Original	76.7 / 86.7 / 99.6	69.3 / 76.1 / 84.3	89.9 / 109.4 / 149.6	672.7 / 886.7 / 1139.2	216.4 / 454.9 / 705.9

Table 12: Average reasoning length (tokens) of the top 5% / 10% / 20% shortest responses for `deepseek-distill-qwen-32B`.

A.6 ThinkEdit Results on Non-Math Domains

We include supplementary experiments on three non-math subjects from MMLU—*High School Computer Science*, *Formal Logic*, and *Professional Accounting*. These tasks do not directly require mathematical calculation, but they still demand structured reasoning and logical consistency. As shown in Tables 13 and 14, ***ThinkEdit*** not only improves overall accuracy but also mitigates failure cases that arise from overly short reasoning traces. This indicates that ***ThinkEdit*** can boost performance in reasoning-heavy domains beyond mathematics.

Model	Variant	MMLU HS CS	MMLU Formal Logic	MMLU Prof. Accounting
deepseek-qwen-32B	Original	96.50 ± 1.18	93.49 ± 1.29	**84.22 ± 1.05**
	ThinkEdit (4%)	**97.20 ± 0.92**	**94.05 ± 1.80**	83.40 ± 1.89
deepseek-qwen-14B	Original	93.50 ± 1.58	91.27 ± 2.08	75.85 ± 1.89
	ThinkEdit (4%)	**94.80 ± 1.14**	**91.51 ± 1.98**	**77.09 ± 1.63**
deepseek-llama3-8B	Original	84.00 ± 3.83	63.41 ± 1.47	57.06 ± 1.48
	ThinkEdit (4%)	**88.90 ± 1.91**	**65.40 ± 2.85**	**57.52 ± 1.40**
deepseek-qwen-1.5B	Original	63.90 ± 4.38	51.27 ± 3.00	35.71 ± 2.85
	ThinkEdit (4%)	**68.30 ± 3.02**	**52.30 ± 3.07**	**37.09 ± 1.78**

Table 13: Overall accuracy on MMLU non-math subjects.

Model	Variant	MMLU HS CS	MMLU Formal Logic	MMLU Prof. Accounting
deepseek-qwen-32B	Original	98.00 / 98.00 / 97.50	85.00 / **88.33** / 91.60	89.29 / 87.86 / 87.86
	ThinkEdit (4%)	**100.00 / 99.00 / 99.50**	**88.33 / 88.33 / 92.00**	**92.86 / 91.07 / 91.96**
deepseek-qwen-14B	Original	96.00 / 94.00 / 96.00	78.33 / 85.83 / 90.40	82.14 / 82.50 / 84.82
	ThinkEdit (4%)	**100.00 / 99.00 / 99.50**	**85.00 / 90.00 / 92.00**	**90.00 / 91.43 / 90.36**
deepseek-llama3-8B	Original	72.00 / 76.00 / 81.00	75.00 / 75.83 / 74.00	70.71 / 70.71 / 67.32
	ThinkEdit (4%)	**96.00 / 95.00 / 96.00**	**80.00 / 77.50 / 81.20**	**75.00 / 74.64 / 70.54**
deepseek-qwen-1.5B	Original	66.00 / 66.00 / 71.00	48.33 / 51.67 / 61.60	32.86 / 32.14 / 33.21
	ThinkEdit (4%)	**92.00 / 89.00 / 84.00**	**60.00 / 61.67 / 67.20**	**43.57 / 43.21 / 40.36**

Table 14: Accuracy on the top 5% / 10% / 20% shortest responses on MMLU non-math subjects.

A.7 Behavioral Analysis: ThinkEdit Encourages Deeper Reasoning

To better understand how ***ThinkEdit*** influences reasoning quality beyond final correctness, we analyze model behavior over *all* examples (correct and incorrect). We quantify (i) the average number of LaTeX equations written in the `<think>` trace, (ii) the percentage of examples whose final answer is explicitly boxed in `<think>`, and (iii) the average reasoning length (tokens) within `<think>`.

Across the four base models, ***ThinkEdit*** consistently produces more equations (+55.2% on average), boxes the final answer more often (+60.5%), and writes longer reasoning traces (+48.0%) compared to the original models. The effect is especially pronounced for smaller models (e.g., `deepseek-qwen-1.5B`: equations +83.9%, boxing +84.4%). These results indicate that the edits reliably encourage more explicit and structured chains of thought across the board.

Model	Variant	Equations in CoT	Wrap Answer in CoT (%)	Reasoning Length (tokens)
deepseek-qwen-32B	Original	1.92	11.75	308.32
	ThinkEdit (4%)	**2.85**	**16.00**	**409.03**
deepseek-qwen-14B	Original	2.11	15.39	341.09
	ThinkEdit (4%)	**2.74**	**23.73**	**518.06**
deepseek-llama3-8B	Original	3.21	30.25	606.09
	ThinkEdit (4%)	**5.09**	**50.57**	**957.32**
deepseek-qwen-1.5B	Original	3.83	28.05	753.45
	ThinkEdit (4%)	**7.05**	**51.71**	**1125.64**

Table 15: Behavioral comparison over all datasets. ***ThinkEdit*** generates more equations, is more likely to box the final answer, and produces longer reasoning traces than the original models, indicating more explicit and structured CoT across cases.

A.8 Examples of Steering the Reasoning Length

To demonstrate the effect of steering the reasoning length, we present two examples from gsm8k in Figures 7 and 8. Each figure contains three different reasoning outputs:

- **Left**: The model's response when steered towards shorter reasoning with $\alpha = -0.04$.
- **Middle**: The original unaltered response.
- **Right**: The model's response when steered towards longer reasoning with $\alpha = 0.04$.

These examples highlight that steering along the reasoning direction effectively modulates the reasoning length without causing unintended truncation or artificial elongation. In contrast to naive methods such as forcibly stopping the reasoning process—resulting in incomplete reasoning—or appending redundant tokens like "Wait" to artificially extend responses, our approach enables the model to generate complete and coherent reasoning of varying lengths. By directly manipulating the internal representation, we achieve a more controlled and flexible mechanism for managing reasoning length, which could be particularly useful for constrained computational budgets.

Toulouse has twice as many sheep as Charleston. Charleston has 4 times as many sheep as Seattle. How many sheep do Toulouse, Charleston, and Seattle have together if Seattle has 20 sheep?

✓ Steer to Short Reasoning (α=-0.04)

<think> First, determine the number of sheep in Seattle, which is given as 20.

Next, calculate the number of sheep in Charleston by multiplying Seattle's sheep by 4, resulting in 80 sheep.

Then, find the number of sheep in Toulouse by multiplying Charleston's sheep by 2, which gives 160 sheep.

Finally, add the sheep from all three cities together to find the total number of sheep. </think>

Let's determine the number of sheep each city has and then find the total together.

1. **Seattle's Sheep:**

   [\text{Seattle's sheep} = 20]

2. **Charleston's Sheep:**

   Charleston has **4 times** as many sheep as Seattle.

   [\text{Charleston's sheep} = 4 \times \text{Seattle's sheep} = 4 \times 20 = 80]

3. **Toulouse's Sheep:**

   Toulouse has **twice** as many sheep as Charleston.

   [\text{Toulouse's sheep} = 2 \times \text{Charleston's sheep} = 2 \times 80 = 160]

4. **Total Sheep Together:**

   [\text{Total sheep} = \text{Toulouse's sheep} + \text{Charleston's sheep} + \text{Seattle's sheep} = 160 + 80 + 20 = 260]

[\boxed{260}]

✓ Original (α=0.0)

<think> Okay, so I have this problem here about the number of sheep in different cities: Toulouse, Charleston, and Seattle. It says that Toulouse has twice as many sheep as Charleston. Then, Charleston has four times as many sheep as Seattle. And the question is asking how many sheep all three cities have together, given that Seattle has 20 sheep.

Hmm, let me try to break this down. I think the key here is to find out how many sheep each city has and then add them up. Since we're given information about relationships between the number of sheep in different cities, maybe I can use that to find the numbers.

First, it says Seattle has 20 sheep. That's straightforward. Now, Charleston has four times as many sheep as Seattle. So if Seattle has 20, then Charleston must have 4 times 20. Let me calculate that: 4 * 20 = 80. Okay, so Charleston has 80 sheep.

Next, Toulouse has twice as many sheep as Charleston. Since Charleston has 80, Toulouse must have 2 times 80. Let me do that multiplication: 2 * 80 = 160. So, Toulouse has 160 sheep.

Now, to find the total number of sheep altogether, I need to add up the sheep from each city. That would be Toulouse's sheep plus Charleston's sheep plus Seattle's sheep. So, that's 160 + 80 + 20.

Let me add those numbers: 160 + 80 is 240, and then 240 + 20 is 260. So altogether, they have 260 sheep.

Wait, let me double-check to make sure I didn't make a mistake. If Seattle has 20, Charleston has 4 times that, which is 80. Toulouse has twice as many as Charleston, so 2 times 80 is 160. Adding them together: 20 + 80 is 100, and 100 + 160 is 260. Yeah, that seems right.

I don't think I missed anything here. The relationships were clearly defined: Toulouse relative to Charleston, Charleston relative to Seattle. And the numbers were given for Seattle, so I could work backward from there. So, I think 260 is the correct total number of sheep.

Final Answer The total number of sheep is \boxed{260}. </think>

Seattle has 20 sheep. Charleston has four times as many sheep as Seattle, which is calculated as: [4 \times 20 = 80]

Toulouse has twice as many sheep as Charleston, which is calculated as: [2 \times 80 = 160]

Adding the number of sheep from each city together: [160 + 80 + 20 = 260]

Thus, the total number of sheep is (\boxed{260}).

✓ Steer to Long Reasoning (α=0.04)

<think> Okay, so I have this problem here about the number of sheep in different cities: Toulouse, Charleston, and Seattle. Hmm, let me read it again. It says, "Toulouse has twice as many sheep as Charleston. Charleston has 4 times as many sheep as Seattle. How many sheep do Toulouse, Charleston, and Seattle have together if Seattle has 20 sheep?"

Alright, let me break this down. So, we have three cities: T, C, and S. The problem is giving me relationships between their sheep populations. Let me translate that into equations.

First, "Toulouse has twice as many sheep as Charleston." So, if I let C represent the number of sheep in Charleston, then T is 2*C. Got that.

Second, "Charleston has 4 times as many sheep as Seattle." So, if S is the number in Seattle, then C is 4*S. Hmm, so S is Seattle, which is given as 20 sheep. So, that should help me find C first.

Third, they are asking for the total number of sheep in all three cities together. So, I need to find T + C + S.

Alright, let's write down what we know:

1. T = 2*C
2. C = 4*S
3. S = 20

So, let's plug in S first. Since S is 20, then C is 420. Let me calculate that: 420 is 80. So, C is 80.

Then, T is twice as many as C, so T is 280. 280 is 160. So, T is 160.

So, now, we have:

- T: 160
- C: 80
- S: 20

So, adding them all up: 160 + 80 + 20. Let me do that: 160 + 80 is 240, and 240 + 20 is 260. So, together, they have 260 sheep.

Wait, let me double-check. Maybe I made a mistake in the relationships.

First, T is twice as much as C. So, T = 2*C. Yeah, that seems right.

C is four times as much as S. So, C = 4S. So, 420 is 80. That seems okay.

Then, T is 2*80, which is 160. So, that's correct.

Adding them all up: 160 + 80 is 240, plus 20 is 260. Hmm, that seems straightforward.

Wait, just to make sure, let me think about it differently. If Seattle has 20, then Charleston has 4 times that, which is 80. Then Toulouse is twice that, which is 160. So, 20, 80, 160. Yeah, that seems to make sense.

Is there another way to think about it? Maybe in terms of ratios.

So, T is twice as big as C, and C is four times as big as S. So, T is 22S, which is 4S. Wait, is that right? Wait, if C is 4S, and T is 2C, so T is 24S, which is 8S. Wait, hold on, that's conflicting with my earlier thought.

Wait, so S is 20. So, if T is 8S, that would be 160. Wait, but earlier I thought T was 2C, which is 2*80, which is 160. So, same result. So, maybe the ratio thing is correct.

Wait, maybe I confused myself for a second. So, S is 20. C is 420, which is 80. T is 280, which is 160. So, that's 4 times S, 8 times S, and 16 times S? Wait, no, that doesn't make sense.

Wait, no, T is 2C, which is 24S, which is 8S. So, T is 8S, C is 4S, S is S. So, the ratio is 8:4:1. So, 8 parts T, 4 parts C, 1 part S.

So, if S is 20, that's 1 part. So, 1 part is 20, 4 parts is 80, 8 parts is 160. So, that's the same as before. So, total is 8+4+1=13 parts. 13*20 is 260. So, yeah, that's another way to get 260.

So, that seems to check out.

Wait, so maybe I overcomplicated it, but I think both methods lead to the same answer. So, that gives me more confidence.

Wait, another way: let's use algebra. Let me assign variables.

Let me let S be the number of sheep in Seattle. So, S = 20.

Then, C is 4S, so C = 420 = 80.

Then, T is 2C, so T = 280 = 160.

Total sheep: T + C + S = 160 + 80 + 20 = 260.

So, that's the straightforward way, just plugging in.

I think I'm confident that 260 is the correct answer.

Wait, just to think about if I misread the problem: "Toulouse has twice as many sheep as Charleston. Charleston has 4 times as many sheep as Seattle." So, T is twice as much as C, which is four times as much as S. So, T is 2*(4S) = 8S. So, T is 8S, C is 4S, S is S. So, yeah, 8S, 4S, S.

So, if S is 20, then T is 820=160, C is 420=80, S is 20. So, total is 160 + 80 + 20 = 260.

So, yeah, that seems correct.

Is there a chance that the relationships are the other way around? Like, maybe T is half of C, or something? But the problem says "Toulouse has twice as many sheep as Charleston." So, T is bigger than C. So, T = 2C. So, that's correct.

Similarly, "Charleston has 4 times as many sheep as Seattle." So, C = 4S. So, that's correct.

So, yeah, I think 260 is the right answer.

Wait, just another way: let me think in terms of ratios. If C is 4 times S, so the ratio of C:S is 4:1. Then, T is twice C, so the ratio of T:C is 2:1. So, combining these, T:C:S is 2:1: something? Wait, no.

Wait, let's see. If C is 4 times S, so C:S is 4:1. Then, T is 2 times C, so T:C is 2:1. So, in terms of T:C:S, T is 2, C is 1, S is 0.25. So, 2:1:0.25.

But that's in terms of C. So, if we make it in terms of S, since S is 1, C is 4, T is 8. So, T:C:S is 8:4:1.

So, 8 parts T, 4 parts C, 1 part S.

So, each part is 20 sheep because S is 20, so 1 part is 20.

So, T is 8 parts: 8*20=160.

C is 4 parts: 4*20=80.

S is 1 part: 20.

So, total is 8+4+1=13 parts. 13*20=260. So, yeah, that's another way.

So, all these methods are leading me to 260. So, I think that's the right answer.

I don't think I made any mistakes here. So, yeah, I think 260 is the total number of sheep in Toulouse, Charleston, and Seattle.

Final Answer The total number of sheep is \boxed{260}. </think>

Given the relationships between the number of sheep in Toulouse, Charleston, and Seattle, we start by translating the problem into equations.

1. Toulouse has twice as many sheep as Charleston: (T = 2C)
2. Charleston has 4 times as many sheep as Seattle: (C = 4S)
3. Seattle has 20 sheep: (S = 20)

First, we find the number of sheep in Charleston (C): [C = 4S = 4 \times 20 = 80]

Next, we find the number of sheep in Toulouse (T): [T = 2C = 2 \times 80 = 160]

Now, we have the number of sheep in each city: - Toulouse: 160 - Charleston: 80 - Seattle: 20

Adding them together gives the total number of sheep: [T + C + S = 160 + 80 + 20 = 260]

Thus, the total number of sheep is (\boxed{260}).

Figure 7: GSM8k Example 1: Steering the reasoning length of deepseek-distill-llama3-8b. Left: shorter reasoning ($\alpha = -0.04$), Middle: original response, Right: longer reasoning ($\alpha = 0.04$).

In a dance class of 20 students, 20% enrolled in contemporary dance, 25% of the remaining enrolled in jazz dance, and the rest enrolled in hip-hop dance. What percentage of the entire students enrolled in hip-hop dance?

✓ Steer to Short Reasoning (α=-0.04)

<think> First, determine the number of students enrolled in contemporary dance by calculating 20% of the total 20 students.

Next, find out how many students remain after those who enrolled in contemporary dance. This is done by subtracting the number of contemporary dance students from the total number of students.

Then, calculate the number of students enrolled in jazz dance by taking 25% of the remaining students.

Finally, determine the percentage of students enrolled in hip-hop dance by subtracting the number of students enrolled in contemporary and jazz dances from the total number of students and then converting this to a percentage. </think>

To determine the percentage of students enrolled in hip-hop dance, follow these steps:

1. **Calculate the number of students enrolled in contemporary dance:**

   [\text{Contemporary Dance Students} = 20\% \times 20 = 0.20 \times 20 = 4 \text{ students}]

2. **Find out how many students remain after those who enrolled in contemporary dance:**

   [\text{Remaining Students} = 20 - 4 = 16 \text{ students}]

3. **Calculate the number of students enrolled in jazz dance:**

   [\text{Jazz Dance Students} = 25\% \times 16 = 0.25 \times 16 = 4 \text{ students}]

4. **Determine the number of students enrolled in hip-hop dance:**

   [\text{Hip-Hop Dance Students} = 16 - 4 = 12 \text{ students}]

5. **Calculate the percentage of students enrolled in hip-hop dance:**

   [\text{Percentage of Hip-Hop Dance Students} = \left(\frac{12}{20} \right) \times 100\% = 60\%]

Final Answer:

[\boxed{60\%}]

✓ Original (α=0.0)

<think> Okay, so I have this problem here about a dance class with 20 students. It says that 20% enrolled in contemporary dance, 25% of the remaining enrolled in jazz dance, and the rest enrolled in hip-hop dance. I need to find out what percentage of the entire students enrolled in hip-hop dance.

Hmm, let me break this down step by step. First, the class has 20 students total. So, I can think of this as 100% of the students.

The first group is 20% enrolled in contemporary dance. Let me calculate how many students that is. 20% of 20 is 0.20 times 20, which is 4 students. So, 4 students are taking contemporary dance.

Now, the problem says 25% of the remaining enrolled in jazz dance. Okay, so after taking out the 20% for contemporary, we have 20 minus 4, which is 16 students left. So, 25% of these 16 students will take jazz dance. Let me compute that. 25% of 16 is 0.25 times 16, which is 4 students. So, 4 students are taking jazz dance.

Now, the rest enrolled in hip-hop dance. So, if we have 16 students left and 4 of them are taking jazz, then the remaining number of students is 16 minus 4, which is 12. So, 12 students are taking hip-hop dance.

The question is asking for the percentage of the entire students who enrolled in hip-hop dance. So, out of the total 20 students, 12 are taking hip-hop. To find the percentage, I can take the number of hip-hop dancers divided by the total number of students and then multiply by 100.

So, that's 12 divided by 20. Let me compute that. 12 divided by 20 is 0.6. Multiply by 100 gives 60%. Therefore, 60% of the students enrolled in hip-hop dance.

Wait, let me double-check my steps to make sure I didn't make a mistake. First, 20% of 20 is 4 for contemporary. Then, 25% of 16 is 4 for jazz. Subtracting those, 16 minus 4 is 12 for hip-hop. 12 out of 20 is indeed 60%. That seems right.

I wonder if there's another way to approach this problem without calculating the exact numbers. Maybe using fractions or percentages directly. Let me think.

So, 20% are in contemporary. That leaves 80% of the students. Then, 25% of the remaining 80% are in jazz. So, 25% of 80% is 20% of the total students. So, 20% for jazz. Therefore, the rest, which is 80% minus 20%, so 60%, are in hip-hop dance. Oh, that's another way to see it. So, 60% of the total students are in hip-hop dance. That matches my earlier calculation.

So, that's a good confirmation. It seems that regardless of whether I calculate it step by step with numbers or do it more abstractly with percentages, I end up with the same answer, 60%.

I think that's solid. So, I'm confident that the percentage is 60%.

Final Answer The percentage of students who enrolled in hip-hop dance is \boxed{60\%}. </think>

First, we note that the dance class has 20 students.

1. 20% of the students enrolled in contemporary dance: [20\% \text{ of } 20 = 0.20 \times 20 = 4 \text{ students}]
2. The remaining students after contemporary dance are: [20 - 4 = 16 \text{ students}]
3. 25% of the remaining students enrolled in jazz dance: [25\% \text{ of } 16 = 0.25 \times 16 = 4 \text{ students}]
4. The remaining students after jazz dance enrolled in hip-hop dance: [16 - 4 = 12 \text{ students}]
5. To find the percentage of the entire students who enrolled in hip-hop dance: [\frac{12}{20} \times 100 = 60\%]

Thus, the percentage of the entire students who enrolled in hip-hop dance is \boxed{60\%}.

✓ Steer to Long Reasoning (α=0.04)

<think> Okay, so I have this problem here about a dance class with 20 students. Let me try to figure it out step by step. Hmm, 20% enrolled in contemporary dance, then 25% of the remaining enrolled in jazz dance, and the rest in hip-hop. I need to find what percentage of all the students are in hip-hop. Alright, let's break it down.

First, the total number of students is 20. 20% of them are in contemporary dance. So, let me calculate that. 20% of 20 is 0.2 times 20, which is 4 students. So, 4 students are doing contemporary dance. That leaves the rest of the students, right? So, 20 minus 4 is 16 students remaining.

Now, 25% of the remaining 16 students enrolled in jazz dance. Let me see, 25% is a quarter, so a quarter of 16 is... 16 divided by 4 is 4. So, 4 students are in jazz dance. That means the rest must be in hip-hop. So, 16 minus 4 is 12 students in hip-hop.

Wait, so out of 20 students, 12 are in hip-hop. So, what percentage is that? I need to convert 12 into a percentage of 20. So, 12 divided by 20 is 0.6, and 0.6 times 100 is 60%. So, 60% of the students are in hip-hop dance.

Let me double-check that because percentages can sometimes be tricky. So, 20 students. 20% is 4 in contemporary. Then, 25% of the remaining 16 is 4 in jazz. So, 4+4 is 8 in contemporary and jazz. 20 minus 8 is 12 in hip-hop. 12 over 20 is 0.6, so 60%. Yeah, that seems right.

Alternatively, maybe I can think about it in fractions. 20% is 1/5, so 1/5 in contemporary, then 25% is 1/4 in jazz. So, 1/5 + 1/4 is... common denominator is 20. So, 4/20 + 5/20 is 9/20. So, 9/20 is 45% in contemporary and jazz. So, 100% minus 45% is 55%? Wait, no, hold on, that doesn't match my previous answer. Hmm, something's wrong here.

Wait, maybe I messed up. Let me think again. So, 20% is 4, 25% of the remaining is 4, so 4+4 is 8. 8 out of 20 is 40%, so 40% in contemporary and jazz. Then, 60% in hip-hop. So, why did I get 45% before? Maybe I did the fractions wrong.

Wait, 20% is 1/5, 25% is 1/4. So, 1/5 + 1/4 is 4/20 + 5/20 is 9/20, which is 0.45, so 45%. Hmm, but that's not matching the actual numbers. Because in reality, 4+4 is 8, which is 40%, so maybe the fractions are wrong because 25% is 1/4 of the remaining, not of the total. So, maybe the way I added them was incorrect.

Yeah, right. Because 25% is 1/4 of the remaining 16, not 1/4 of the total 20. So, when I did the fractions, I was adding 1/5 and 1/4 of the total, which is wrong because the 1/4 is only of the remaining. So, actually, the correct way is 20% is 4, then 25% of 16 is 4, so 4+4=8, so 40% in contemporary and jazz, and 60% in hip-hop.

So, maybe the fractions approach was confusing me because the second percentage is relative to a subset, not the total. So, that's why adding the fractions directly didn't work. So, better to stick with the actual numbers.

So, to recap: 20 students. 20% is 4 in contemporary. 25% of 16 remaining is 4 in jazz. 16 minus 4 is 12 in hip-hop. 12 is 60% of 20. So, yeah, 60% is the answer.

Another way to think about it is, instead of dealing with percentages, maybe use variables. Let me try that.

Let me denote:

Let total students = 20.

Let C be the number in contemporary dance.

Let J be the number in jazz dance.

Let H be the number in hip-hop.

Given that C = 0.2 * 20 = 4.

Then, J is 25% of the remaining students, which is 25% of (20 - C) = 0.25(20 - 4) = 0.2516 = 4.

Therefore, J = 4.

Therefore, H = 20 - C - J = 20 - 4 - 4 = 12.

Therefore, H = 12, which is 12/20 = 0.6 = 60%.

Same result. So, that seems consistent.

So, yeah, I think 60% is the answer. So, the percentage of the entire students enrolled in hip-hop dance is 60%.

Final Answer The percentage of the entire students enrolled in hip-hop dance is \boxed{60\%}. </think>

First, we know that 20% of the 20 students enrolled in contemporary dance. Calculating that, we get:

[0.2 \times 20 = 4]

So, 4 students are enrolled in contemporary dance. This leaves 16 students remaining. Next, 25% of these remaining 16 students enrolled in jazz dance:

[0.25 \times 16 = 4]

So, 4 students are enrolled in jazz dance. Therefore, the remaining students must be enrolled in hip-hop dance:

[16 - 4 = 12]

To find what percentage 12 is of the entire 20 students, we calculate:

[\frac{12}{20} \times 100 = 60\%]

Thus, the percentage of the entire students enrolled in hip-hop dance is \boxed{60\%}.

Figure 8: GSM8k Example 2: Steering the reasoning length of deepseek-distill-llama3-8b. Left: shorter reasoning ($\alpha = -0.04$), Middle: original response, Right: longer reasoning ($\alpha = 0.04$).

A.9 Examples of Fixing the Overly Short Reasoning with *ThinkEdit*

To illustrate the effectiveness of our ***ThinkEdit*** approach in addressing the overly short reasoning issue, we show two representative examples from the GSM8K and MATH-level5 datasets.

Example from GSM8K. The example is shown in Figure 9. In this problem, the *Original* model misinterprets the question and quickly provides an incorrect conclusion. In contrast, after applying ***ThinkEdit***, the model first organizes the given information and displays a detailed reasoning process that leads to the correct answer.

Example from MATH-LEVEL5. The example is shown in Figure 10. Both the *Original* model and the model with ***ThinkEdit*** begin with an incorrect line of reasoning. However, the original model does not revise its approach and proceeds to an incorrect final conclusion. In contrast, the model with ***ThinkEdit*** revisits its reasoning steps, recognizes the mistake, and corrects the process in time to reach the correct solution.

These examples demonstrate ***ThinkEdit*** can facilitate deeper, more accurate reasoning by guiding the model to properly utilize problem details, reconsider faulty steps, and ultimately yield correct solutions.

The Doubtfire sisters are driving home with 7 kittens adopted from the local animal shelter when their mother calls to inform them that their two house cats have just had kittens. She says that Patchy, the first cat, has had thrice the number of adopted kittens, while Trixie, the other cat, has had 12. How many kittens does the Doubtfire family now have?

✗ Original Model

<think> First, identify the total number of kittens the Doubtfire family already has from their two house cats. Patchy has had 3 times as many kittens as Trixie. If Trixie has 12 kittens, then Patchy has 3 multiplied by 12, which equals 36 kittens.

Next, add the number of kittens from the house cats to the number from the adopted kittens. The family has 36 kittens from Patchy and 12 from Trixie, making a total of 48 kittens at home.

Then, add the 7 adopted kittens from the shelter to the existing total. Adding 7 to 48 gives a final total of 55 kittens in the Doubtfire family. </think>

Let's break down the problem step by step to find out how many kittens the Doubtfire family has now.

1. **Identify the Number of Kittens from Each Cat:**
   - **Patchy the Cat:** Has had **3 times** the number of adopted kittens as Trixie.
   - **Trixie the Cat:** Has had **12** adopted kittens.
2. **Calculate the Number of Kittens from Patchy:**

   [\text{Kittens from Patchy} = 3 \times \text{Kittens from Trixie} = 3 \times 12 = 36]
3. **Determine the Total Number of Kittens at Home:**

   [\text{Total kittens from house cats} = \text{Kittens from Patchy} + \text{Kittens from Trixie} = 36 + 12 = 48]
4. **Add the Adopted Kittens from the Shelter:**

   [\text{Total kittens} = \text{Total kittens from house cats} + \text{Adopted kittens} = 48 + 7 = 55]

Final Answer:

[\boxed{55}]

✓ ThinkEdit Model

<think> First, let's identify the information given:

- The Doubtfire sisters have 7 kittens that they adopted from the shelter.
- Their mother says that their two house cats, Patchy and Trixie, have each had their own kittens.
- Patchy has had thrice the number of adopted kittens.
- Trixie has had 12 kittens.

Wait, hold on. It says that Patchy, the first cat, has had thrice the number of adopted kittens, while Trixie, the other cat, has had 12. So, the 7 kittens they adopted from the shelter—does that refer to the house cats or the shelter kittens?

I think the 7 kittens are the ones they adopted from the shelter, separate from their house cats. So, their house cats, Patchy and Trixie, have each had their own kittens. Patchy has had thrice as many as the shelter kittens, and Trixie has had 12.

Wait, maybe the wording is that Patchy has had thrice the number of adopted kittens, meaning compared to Trixie. Or is it thrice the number of the 7 kittens?

Wait, the problem says: "Patchy, the first cat, has had thrice the number of adopted kittens, while Trixie, the other cat, has had 12." So, "adopted kittens"—does that refer to the 7 they adopted from the shelter, or is it referring to the kittens that Patchy and Trixie have had?

I think it's referring to the 7 kittens. So, Patchy has had thrice as many as the 7, so 21, and Trixie has had 12.

So, the total number of kittens would be 7 (shelter kittens) plus 21 (Patchy's kittens) plus 12 (Trixie's kittens).

Wait, but the shelter kittens are 7, and the house cats have their own kittens. So, the total number is 7 + 21 + 12.

But wait, is that correct? Let me read again.

"They are driving home with 7 kittens adopted from the local animal shelter when their mother calls to inform them that their two house cats have just had kittens. She says that Patchy, the first cat, has had thrice the number of adopted kittens, while Trixie, the other cat, has had 12."

So, the 7 kittens are the ones they are driving home. Their mother is telling them about their house cats, Patchy and Trixie, who have each had kittens. So, Patchy has had thrice the number of kittens as the 7 they are driving home. So, thrice 7 is 21. Trixie has had 12.

So, total kittens now are the 7 in the car plus 21 from Patchy and 12 from Trixie. So, 7 + 21 + 12 = 40.

Wait, but is Trixie's 12 also relative to the 7? Or is it 12 total? The problem says "Patchy, the first cat, has had thrice the number of adopted kittens, while Trixie, the other cat, has had 12."

Hmm, the wording is a bit ambiguous. Maybe "number of adopted kittens" refers to the 7 they are driving home. So, Patchy has had thrice that, so 21, and Trixie has had 12. So, total kittens are 7 (shelter) + 21 (Patchy) + 12 (Trixie) = 40.

Alternatively, if "adopted kittens" refers to the kittens Patchy and Trixie have adopted, but that wouldn't make sense because they are the house cats.

I think the first interpretation is correct. So, total kittens now are 40.

But wait, let me double-check. The problem says "their two house cats have just had kittens." So, Patchy has had thrice the number of adopted kittens—so 7 is the number of adopted kittens, so thrice that is 21. Trixie has had 12.

So, yes, total kittens are 7 + 21 + 12 = 40.

But wait, hold on. Are the 7 kittens in the car considered as adopted? Or are they separate from the house cats' kittens? Because the problem says they adopted 7 from the shelter, and their house cats had kittens. So, the total number is the 7 plus the house cats' kittens.

So, yes, 7 + 21 + 12 = 40.

Alternatively, if "Patchy has had thrice the number of adopted kittens" means that Patchy has had 3 times as many as Trixie, and both together have had 12. But that wouldn't make sense because it says Patchy has had thrice the number of adopted kittens, while Trixie has had 12. So, it's more likely that Patchy has had 3 times 7, which is 21, and Trixie has had 12, so total 21 + 12 = 33, plus the 7 they are driving home, making 40.

Yes, that makes sense.

So, the answer is 40 kittens.

Final Answer The Doubtfire family now has \boxed{40} kittens. </think>

The Doubtfire sisters are driving home with 7 kittens adopted from the local animal shelter. Their mother informs them that their two house cats, Patchy and Trixie, have just had kittens. Patchy has had thrice the number of adopted kittens, while Trixie has had 12.

1. The 7 kittens are adopted from the shelter.
2. Patchy, the first cat, has had thrice the number of these 7 kittens. Therefore, Patchy has had (3 \times 7 = 21) kittens.
3. Trixie, the other cat, has had 12 kittens.

To find the total number of kittens, we add the 7 kittens from the shelter, the 21 kittens from Patchy, and the 12 kittens from Trixie:

[7 + 21 + 12 = 40]

Thus, the Doubtfire family now has (\boxed{40}) kittens.

Figure 9: An example of ***ThinkEdit*** from the GSM8K dataset. The original model provides a short, flawed explanation. After ***ThinkEdit***, the model's reasoning process is more thorough.

At the national curling championships, there are three teams of four players each. After the championships are over, the very courteous participants each shake hands three times with every member of the opposing teams, and once with each member of their own team.

How many handshakes are there in total?

✗ Original Model

<think> First, determine the total number of participants. There are three teams, each with four players, making a total of 12 participants.

Next, calculate the number of handshakes within each team. Each team member shakes hands once with their teammates. For one team, this results in 4 players shaking hands with 3 others each, totaling 6 handshakes. Since there are three teams, the total intra-team handshakes are 18.

Then, calculate the number of handshakes between teams. Each player shakes hands three times with each member of the opposing teams. For one player, this involves 3 players from each of the other two teams, resulting in 6 handshakes per player. With 12 players in total, this gives 72 handshakes across all inter-team interactions.

Finally, add the intra-team and inter-team handshakes together to find the total number of handshakes: 18 + 72 = 90. </think>

Let's break down the problem step by step.

Total Participants: - There are 3 teams, each with 4 players. - Total participants = 3 × 4 = **12 players**.

Handshakes Within Each Team: - Each player shakes hands **once** with each of their teammates. - For one team: - Number of handshakes = (\binom{4}{2} = \frac{4 \times 3}{2} = 6) - Since there are 3 teams: - Total intra-team handshakes = 6 × 3 = **18 handshakes**

Handshakes Between Teams: - Each player shakes hands **three times** with each member of the opposing teams. - For one player: - Number of opponents from other teams = 8 (since there are 12 - 4 = 8 players not in their team) - Total handshakes per player = 8 × 3 = **24 handshakes** - For all 12 players: - Total inter-team handshakes = 12 × 24 = **288 handshakes**

Total Handshakes: - Add intra-team and inter-team handshakes: - Total handshakes = 18 + 288 = **306 handshakes**

[\boxed{306}]

✓ ThinkEdit Model

<think> To determine the total number of handshakes, I need to consider how often each participant shakes hands with others. Each person shakes hands three times with every opponent and once with their own team member.

First, let's think about the interactions within the same team. Each team has four players, and each one shakes hands once with the others. So for one team, that's 4 players each shaking hands once with 3 teammates, which is 4 * 3 = 12 handshakes, but since each handshake is between two people, I need to divide by 2 to avoid double-counting. So within one team, it's 6 handshakes.

But wait, actually, no. If each player shakes hands once with each team member, then for a team of four, each handshake is unique. So it's actually 4 choose 2, which is 6 handshakes. So yeah, 6 handshakes per team.

But there are three teams, so that's 3 * 6 = 18 handshakes within the same teams.

Now, between teams, each person shakes hands three times with every member of the opposing teams. So let's figure out how many inter-team handshakes there are.

Each team has four players, and there are three teams, so each player has 4 * 3 = 12 opponents. Each handshake is three times, so that's 12 * 3 = 36 handshakes per player. But since each handshake involves two people, we need to divide by 2 to avoid double-counting. So 36 * 3 / 2 = 54 handshakes between teams.

Wait, hold on. Let me think again.

If each of the 4 players on one team shakes hands three times with each of the 4 players on another team, that's 4 * 4 * 3 = 48 handshakes between two teams. But since there are three teams, each pair of teams has 48 handshakes. How many pairs of teams are there? It's 3 choose 2, which is 3. So total inter-team handshakes would be 48 * 3 = 144.

But that seems high. Maybe I'm overcounting. Let me verify.

Alternatively, think of it as each pair of opposing players shakes hands three times. There are 3 teams, so each player is on one team, and there are two other teams. Each player has 4 opponents on each of the other two teams, so 8 opponents in total. Each of these 8 pairs shakes hands three times, so per player, that's 8 * 3 = 24 handshakes. Since there are 12 players in total (4 per team * 3 teams), that's 12 * 24 / 2 = 144 handshakes. So that matches.

So inter-team handshakes are 144.

Now, adding the intra-team handshakes: 6 per team * 3 teams = 18.

Total handshakes: 144 + 18 = 162.

Wait, but that seems a bit high. Let me check another way.

Total number of participants is 12. Each person shakes hands with 11 others. But the number of handshakes isn't 12 * 11 because each handshake is counted twice. So total handshakes would be (12 * 11) / 2 = 66, but that's if each pair shakes hands once. However, in this case, some pairs shake hands three times and some once.

So perhaps I need to calculate the total number of handshakes as follows:

Within the same team: Each team has 4 players, each shaking hands once with each of the other 3, so 4 * 3 / 2 = 6 handshakes per team. Three teams, so 18.

Between teams: For each pair of teams, each player on one team shakes hands three times with each player on the other team. So for two teams, that's 4 * 4 * 3 = 48 handshakes. There are 3 pairs of teams, so 48 * 3 = 144.

Total handshakes: 18 + 144 = 162.

Alternatively, let's think about each player's total handshakes. Each player shakes hands once with their 3 teammates and three times with each of the 8 opponents (since there are 4 opponents on each of the other two teams). So that's 3 + (4 * 3 * 2) = 3 + 24 = 27 handshakes per player. Wait, 4 opponents on each of two teams, so 4 * 2 = 8 opponents, each with 3 handshakes, so 8 * 3 = 24, plus 3 teammates each with 1 handshake: 24 + 3 = 27.

So each player has 27 handshakes, and there are 12 players, so total handshakes counted this way would be 12 * 27 = 324. But since each handshake is counted twice, we divide by 2: 324 / 2 = 162. So that confirms it.

Therefore, the total number of handshakes is 162.

Final Answer The total number of handshakes is \boxed{162}. </think>

To determine the total number of handshakes at the national curling championships, we need to consider both intra-team and inter-team handshakes.

1. **Intra-team handshakes**:
   - Each team has 4 players, and each player shakes hands once with each of the other 3 teammates.
   - For one team, the number of handshakes is calculated as (\binom{4}{2} = 6).
   - There are 3 teams, so the total number of intra-team handshakes is (3 \times 6 = 18).
2. **Inter-team handshakes**:
   - Each player shakes hands three times with each member of the opposing teams.
   - Each player has 4 opponents on each of the other two teams, resulting in 8 opponents in total.
   - Each pair of opposing players shakes hands three times, so for each player, the number of handshakes is (8 \times 3 = 24).
   - Since there are 12 players in total, the total number of handshakes counted this way is (12 \times 24 = 288).
   - However, each handshake is counted twice, so we divide by 2: (288 / 2 = 144).
3. **Total handshakes**:
   - Adding the intra-team and inter-team handshakes gives (18 + 144 = 162).

Thus, the total number of handshakes is (\boxed{162}).

Figure 10: An example of ***ThinkEdit*** from MATH-level5. While both models initially make a wrong assumption, the model after applying ***ThinkEdit*** corrects itself and arrives at the correct final reasoning.