Can Linguistic Reasoning Vectors Enhance Multimodal Reasoning Ability?

Ziyi Wang[1] **Li Li**[1] **Aolin Zhou**[1] **Yankun Shen**[1]
Chonghan Liu[2] **Shuxia Lin**[1*] **Xu Yang**[1]

[1] School of Computer Science & Engineering, Southeast University, China
[2] University of California, Los Angeles, USA

220256460@seu.edu.cn shuxia.mumu@gmail.com

Abstract

Most Vision-Language Models (VLMs) are built by extending pretrained Large Language Models (LLMs) with visual modules and multimodal alignment. However, this multimodal scaling often degrades the language-side reasoning ability originally encoded in the base LLM. While the base LLM retains usable reasoning after scaling, the aligned VLM itself cannot reliably access this ability. Therefore, recovering the degraded reasoning capability in VLMs would benefit more from seeking help from the base LLM than from the VLM alone. Motivated by this, we propose **LIFT** (**L**anguage-side reason**i**ng **F**acilitation and **T**ransfer), a lightweight vector-intervention method that transfers reasoning capability from the base LLM to the VLM without retraining the backbone. LIFT defines **Reasoning Vectors** as answer-token hidden-state differences between a Reasoner path with an explicit reasoning trace and a Solver path without it, and injects these vectors into language-side activations of the target VLM. LIFT further supports learnable vector adaptation while keeping the VLM backbone frozen. We evaluate LIFT on two VLMs across six reasoning benchmarks, comparing Reasoning Vectors extracted from the base LLM and from the aligned VLM under matched protocols. Results show that LLM-derived vectors consistently outperform VLM-derived vectors, confirming that the base LLM is a more effective source for recovering reasoning. LIFT partially recovers degraded reasoning through lightweight language-side interventions. Further analyses show that Reasoning Vectors influence intermediate reasoning behavior rather than merely altering final answers. The source code is publicly available at `https://github.com/zitai-wang/lift`.

1 Introduction

Most Vision-Language Models (VLMs) are built by extending pretrained Large Language Models (LLMs) with visual modules. A typical architecture employs an LLM backbone, a vision encoder, and a cross-modal projector, achieving multimodal alignment through instruction tuning [1, 7, 12, 18, 21, 50]. This modular design is widely adopted because pretrained LLMs already provide strong language and reasoning capabilities. This raises the possibility that part of a VLM's reasoning ability is inherited from the underlying LLM, rather than acquired solely through multimodal alignment. However, multimodal scaling may change the behavior and internal representations of the original language model [16, 29]. This leads to our central question: whether reasoning representations formed in the base LLM can still be used to intervene on the corresponding VLM.

*Shuxia Lin is the corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

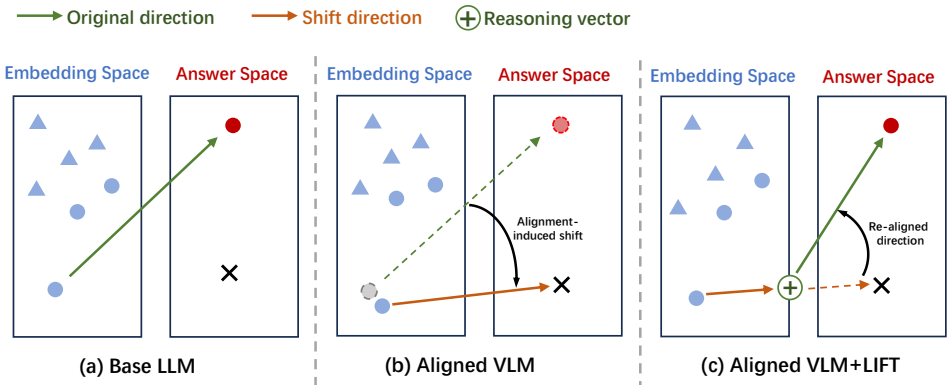


Figure 1: Motivation of LIFT. Multimodal alignment may change language-side reasoning directions in the VLM. LIFT tests whether Reasoning Vectors from the base LLM can be reused as lightweight interventions in the aligned VLM.

As a motivating observation, we compare the base LLM and its corresponding VLM on the same pure-text reasoning benchmarks without visual inputs. We find that the base LLM can sometimes outperform the VLM, even though the VLM shares the same language backbone and is further trained with multimodal data. The gap also appears in the reasoning process. In many failure cases, the VLM starts in the right direction but makes mistakes in later reasoning steps. It may copy a number incorrectly, change an intermediate variable, drop a condition, or produce an answer mismatched with the question target. These failures suggest that the relevant reasoning ability remains usable, but becomes less stable during multi-step generation.

Existing studies have made substantial progress in multimodal reasoning by directly improving the target VLM, including chain-of-thought supervision [38, 39, 47, 49], rationale-rich instruction tuning [10], verification-based reasoning [32], and reasoning-oriented reinforcement learning or post-training [13, 42]. These methods primarily optimize the VLM's external behavior, and do not directly examine whether the reasoning-induced representation shifts learned before multimodal alignment remain recoverable and transferable after alignment. However, since the base LLM already has language-side reasoning ability before multimodal alignment, VLM reasoning may be improved by reusing these representations rather than relying exclusively on heavy VLM retraining. Motivated by this view, we study whether the reasoning-induced hidden-state change in the base LLM can be extracted and reused as a lightweight intervention in the VLM, without updating parameters.

Recent work on task vectors suggests that task-relevant knowledge can be summarized as compact activation or parameter differences [14, 28]. Such vectors can steer model behavior without modifying the full model, providing a natural tool for studying VLM reasoning after multimodal alignment. If alignment changes how the VLM accesses language-side reasoning states, then a reasoning-induced hidden-state shift extracted from the base LLM may serve as a useful intervention direction. Building on this view, we propose **LIFT** (**L**anguage-side reason**i**ng **F**acilitation and **T**ransfer), a lightweight vector-intervention method for VLM reasoning. Given a question, a reasoning trace, and a final answer, we construct two teacher-forced paths. The Reasoner path takes the question–reasoning–answer context, while the Solver path takes only the question–answer context. Because the two paths share the same question and final answer, we use their hidden-state difference on answer tokens as an approximation of the reasoning-induced shift in answer representations. We define this difference as the **Reasoning Vector**. LIFT injects these vectors into the target VLM as lightweight interventions.

We instantiate LIFT with three extraction sources. LLM-derived extracts Reasoning Vectors from the base LLM using text-only reasoning data, capturing reasoning-induced directions before multimodal scaling. VLM-derived extracts vectors from the target VLM using the same text-only data, examining how these directions appear after multimodal alignment. VLM-MM extracts vectors from the target VLM using multimodal reasoning data, where images are included as context but the vectors are still computed from language-side answer representations.

Across Qwen2.5-VL-7B-Instruct [2] and InternVL2.5-8B [5], Reasoning Vectors extracted from the corresponding base LLMs consistently improve VLM performance and outperform vectors extracted from the VLMs. Learnable adaptation further improves performance, especially when initialized from the base-LLM vectors. These results suggest that language-side reasoning representations remain useful after multimodal scaling, but may become less directly accessible in the aligned VLM.

Overall, our main contributions are as follows:

- We introduce LIFT, a lightweight vector-intervention method for VLM reasoning. LIFT provides a framework for extracting and transferring language-side Reasoning Vectors from base LLMs to their corresponding VLMs after multimodal scaling.
- We show that Reasoning Vectors extracted from the base LLM provide stronger interventions than those from the corresponding VLM. This suggests that language-side reasoning ability remains usable after multimodal scaling, and that the corresponding base LLM is a more effective source for recovering degraded reasoning in the aligned VLM. Learnable vector adaptation further recovers access to this reasoning ability in the target VLM.
- We analyze how Reasoning Vectors affect VLM reasoning through source comparisons and layer-wise analysis. These show that Reasoning Vectors influence the reasoning process rather than merely changing the final answer, often helping the model stay aligned with the question target.

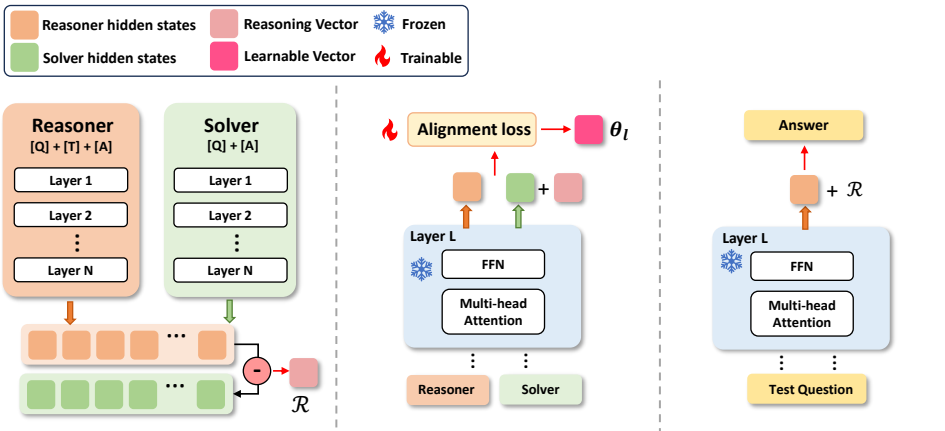
2 Related Work

Task and activation vectors. Task and activation vectors provide compact representations in parameter or representation space for steering model behaviors [16, 28, 31, 36, 51]. Prior work constructs such vectors either from parameter differences between fine-tuned and pretrained models [14, 17, 27] or from activation differences induced by different prompts, contexts, or demonstrations [11, 15, 22, 35]. In language models, these vectors have been used for parameter-efficient adaptation, model editing, in-context learning, and controllable activation steering [41]. More recently, [46] studied latent chain-of-thought vectors, while [19] investigated the transfer and probing of reasoning mechanisms using CoT vectors. These studies show that injecting reasoning-related vectors into hidden states can improve multi-step reasoning. Our work is closest to activation-level steering, but the goal is different. Prior work mainly studies how activation directions represent or control behaviors within language models, whereas LIFT uses reasoning-conditioned activation differences as a probe for multimodal scaling. We study whether Reasoning Vectors extracted from the base LLM remain useful after multimodal scaling, and compare them with vectors extracted directly from the corresponding VLM.

Reasoning enhancement and transfer in VLMs. Prior work has improved VLM reasoning along several directions. At inference time, methods often guide generation through tree-of-thought search [43, 44], question decomposition [45], verification [32], or reward-guided decoding [26]. Regarding training, existing studies introduce multimodal CoT supervision or rationale-rich instruction tuning [10, 30, 34, 40, 47, 48], while recent reasoning-oriented post-training approaches utilize reinforcement learning to further refine performance [4, 13, 42]. Other studies investigate the cross-modal transfer of reasoning capabilities, such as leveraging textual CoT data to compensate for scarce visual annotations [8], steering internal reasoning states [25], or merging LLM and VLM parameters [3]. These advancements demonstrate the vast potential for enhancing VLM reasoning through optimized training and decoding paradigms. Rather than introducing another training or decoding strategy, our focus is to use language-side Reasoning Vectors as probes and interventions for studying whether reasoning representations formed in the base LLM remain recoverable in the aligned VLM after multimodal scaling. We therefore compare vectors extracted from the base LLM and from the corresponding VLM under the same Reasoner–Solver extraction and intervention protocol.

3 Methodology

We propose **LIFT** (**L**anguage-side reason**i**ng **F**acilitation and **T**ransfer), a lightweight vector-intervention method for studying and improving VLM reasoning with language-side Reasoning Vectors. Rather than updating the VLM backbone, LIFT defines a Reasoning Vector as the answer-token hidden-state difference between Reasoner and Solver paths, using it as an approximation of the reasoning-induced shift in answer representations. The resulting vectors are used as interventions in selected language-model layers. We study LIFT in two settings: static vector transfer, where Reasoning Vectors from different sources are directly injected into the target VLM, and learnable vector adaptation, where only activation-level vector parameters are optimized. Figure 2 provides an overview of the overall pipeline. Section 3.1 formulates Reasoning Vectors through a Reasoner–



(a) Reasoner-Solver Framework **(b) Learnable Vector Adaptation** **(c) Reasoning Vector Injection**

Figure 2: Overview of LIFT. (a) For each layer, LIFT computes hidden-state differences between the Reasoner and Solver paths and averages these differences to obtain the Reasoning Vector. (b) Learnable adaptation optimizes only the injected vector parameters while keeping the VLM backbone frozen. (c) At inference, vectors are injected into selected VLM layers.

Solver comparison. Section 3.2 describes vector sources and extraction, while Sections 3.3 and 3.4 detail learnable vector adaptation and inference-time injection, respectively.

3.1 Reasoning Vector Formulation

Given a question Q, a reasoning trace T, and a final answer A, we construct two teacher-forced forward paths using the same model. The **Reasoner** path receives the question–reasoning–answer context, denoted as $\mathcal{C}_R = (Q, T, A)$, while the **Solver** path receives the question–answer context, denoted as $\mathcal{C}_S = (Q, A)$. Hidden states are read only from aligned answer-token positions in the final answer span, while tokens from the question and reasoning trace are excluded.

Let $\boldsymbol{h}_l(a; \mathcal{C})$ denote the hidden representation of answer token a at layer l under context $\mathcal{C}$. For each answer token a, we first compute the token-wise Reasoner–Solver hidden-state difference:

$$\boldsymbol{\delta}_l(a) = \boldsymbol{h}_l(a; \mathcal{C}_R) - \boldsymbol{h}_l(a; \mathcal{C}_S), \quad a \in \mathcal{A}. \tag{1}$$

This difference measures how the answer-token representation changes when the reasoning trace is included in the context. Since the two paths share the same question and final answer, the difference provides an approximation of the reasoning-induced shift in answer representations.

For a support set $\mathcal{D} = \{(q_i, t_i, a_i)\}_{i=1}^{N}$, we average the token-wise differences over the final answer span to obtain a task-level Reasoning Vector at layer l:

$$\boldsymbol{r}_l = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{|\mathcal{A}_i|} \sum_{a \in \mathcal{A}_i} \left[\boldsymbol{h}_l(a; \mathcal{C}_{R,i}) - \boldsymbol{h}_l(a; \mathcal{C}_{S,i}) \right], \tag{2}$$

where $\mathcal{C}_{R,i} = (q_i, t_i, a_i)$ and $\mathcal{C}_{S,i} = (q_i, a_i)$. We compute this average independently at each language-model layer. Thus, Eq. (2) defines a task-level Reasoning Vector $\boldsymbol{r}_l$ for layer l, and the full LIFT representation is the layer-wise collection $\mathcal{R} = \{\boldsymbol{r}_l\}_{l=1}^{L}$.

3.2 Reasoning Vector Sources and Extraction

For each extraction source, we build the support set from samples that the source model answers correctly. We then apply the same Reasoner–Solver protocol to compute its Reasoning Vectors. This allows us to compare reasoning directions obtained from each model's own successful behavior, rather than requiring all models to follow the same reasoning format.

We consider three vector sources. LLM-derived vectors are extracted from the base LLM using text-only reasoning support sets, capturing reasoning-induced directions before multimodal scaling.

VLM-derived vectors are extracted from the target VLM's language backbone using text-only reasoning support sets, testing whether useful language-side reasoning directions can still be obtained after multimodal scaling. VLM-MM vectors are extracted from the target VLM using multimodal reasoning support sets, testing whether multimodal inputs provide directions better aligned with the target VLM's reasoning process.

Here, language-side refers to the extraction and intervention location: all Reasoning Vectors are computed from answer-token hidden states and injected into the language backbone, rather than the vision encoder. Thus, LLM-derived and VLM-derived use text-only support sets, whereas VLM-MM uses multimodal support sets.

3.3 Learnable Vector Adaptation

Static vector transfer tests whether an extracted Reasoning Vector can be directly reused by the target VLM. To better align the vector with the target representation space while keeping the VLM backbone frozen, we introduce learnable vector adaptation. We initialize the learnable vector parameter using the extracted Reasoning Vector at the selected adaptation layer, and optimize only this parameter during adaptation.

During adaptation, we use the support set to obtain Reasoner representations and intervened Solver representations. For the Solver path, the learnable vector is added to the hidden states at the selected adaptation layer:

$$\widetilde{\boldsymbol{x}}_l = \boldsymbol{x}_l + \boldsymbol{\theta}_l. \tag{3}$$

We do not use a separate scaling coefficient in learnable adaptation, because the magnitude of $\boldsymbol{\theta}_l$ is learned during optimization.

Let z_l^R denote the support-set averaged answer-token representation of the Reasoner path at layer l. Similarly, let z_l^S denote the support-set averaged answer-token representation of the Solver path after vector injection. The learnable vector is optimized to bring the intervened Solver representation closer to the Reasoner representation:

$$\mathcal{L}_{\mathrm{rec}} = \left\| \boldsymbol{z}_l^{\mathcal{R}} - \boldsymbol{z}_l^{\mathcal{S}} \right\|_2^2, \qquad \mathcal{L} = \lambda_{\mathrm{rec}} \mathcal{L}_{\mathrm{rec}}. \tag{4}$$

Here, λ_{rec} controls the weight of the representation matching loss. During adaptation, the VLM backbone remains frozen, and only the selected-layer learnable vector parameter θ_l is updated.

3.4 Layer Selection and Injection

Because activation intervention is layer-sensitive [35], we select the injection layer using a small held-out subset before final evaluation. Specifically, for each benchmark, we inject the Reasoning Vector into each candidate language-model layer separately and evaluate performance on this subset. The layer with the best held-out performance is then used for full evaluation. We apply the same layer-selection protocol to all vector sources and adaptation settings.

At inference time, LIFT applies vector addition during autoregressive decoding at the selected language-model layer. Let x_l denote the hidden state at the selected layer. Static and learnable interventions are given by:

$$\tilde{x}_l = x_l + \mu r_l, \qquad \tilde{x}_l = x_l + \theta_l, \tag{5}$$

where μ controls the strength of the static Reasoning Vector and θ_l is the optimized learnable vector. LIFT only modifies language-token hidden states, without updating VLM parameters.

4 Experiments

We evaluate LIFT on Qwen2.5-VL-7B-Instruct [2] and InternVL2.5-8B [5], which are built on Qwen2.5 and InternLM2.5 language backbones, across six reasoning benchmarks. Our experiments study how language-side Reasoning Vectors behave after multimodal scaling. We first test whether static vector intervention can improve VLM reasoning, and then compare Reasoning Vectors extracted from different sources: LLM-derived, VLM-derived, and VLM-MM. Finally, we study learnable vector adaptation to examine whether the gains come from the semantic structure of the extracted Reasoning Vectors, rather than simply from introducing additional trainable parameters. We describe

Table 1: Main results on six reasoning benchmarks across two VLMs. We abbreviate MathVista/MathVision/CommonsenseQA/StrategyQA/ScienceQA as MVista/MVision/CSQA/SQA/SciQA. LLM/VLM-derived denote static vector transfer using Reasoning Vectors extracted from the base LLM and the corresponding VLM, respectively; LLM/VLM-learnable are learnable vectors initialized from these corresponding static vectors. Bold and underlined numbers indicate the best and second-best results for each model and benchmark. All results are reported as accuracy (%), and Avg. denotes the average accuracy across benchmarks.

Model	Method	GSM8K	MVista	MVision	CSQA	SQA	SciQA	Avg.
Qwen2.5-VL	Baseline	82.2	68.5	25.0	77.3	58.6	87.8	66.6
	1-shot CoT	82.7	68.4	25.3	75.1	61.2	87.9	66.8
	2-shot CoT	82.9	68.4	25.3	76.8	61.3	87.9	67.1
	4-shot CoT	82.9	68.4	25.3	<u>78.1</u>	61.6	87.9	67.4
	LLM-derived	<u>84.2</u>	69.1	<u>26.6</u>	78.0	**63.5**	<u>88.1</u>	<u>68.3</u>
	VLM-derived	83.0	<u>70.7</u>	25.7	77.7	60.3	87.8	67.5
	LLM-learnable	**85.6**	**71.8**	**27.3**	**78.3**	<u>62.4</u>	**88.9**	**69.1**
	VLM-learnable	83.2	69.5	24.7	77.7	60.3	87.9	67.2
InternVL2.5	Baseline	77.1	62.8	22.7	82.3	65.6	96.5	67.8
	1-shot CoT	77.1	60.3	23.4	82.4	65.7	96.7	67.6
	2-shot CoT	77.2	63.8	23.4	82.7	63.6	96.7	67.9
	4-shot CoT	77.4	63.8	23.4	<u>83.1</u>	66.7	96.7	68.5
	LLM-derived	<u>78.1</u>	<u>64.0</u>	<u>25.0</u>	<u>83.1</u>	<u>67.1</u>	<u>96.8</u>	<u>69.0</u>
	VLM-derived	77.9	63.0	23.4	82.6	65.9	96.7	68.3
	LLM-learnable	78.0	**64.6**	**25.3**	**83.2**	**67.6**	**97.4**	**69.4**
	VLM-learnable	**78.4**	63.4	20.1	79.0	62.4	96.0	66.6

the experimental setup in Section 4.1 and present the main results in Section 4.2, followed by representation-level and layer-wise analyses in Section 4.3.

4.1 Setup and Implementation Details

Models and datasets. We evaluate LIFT on two VLMs across six reasoning benchmarks that span text-only and multimodal settings. The text-only benchmarks are GSM8K [6], CommonsenseQA [33], and StrategyQA [9], while the multimodal benchmarks are MathVista [24], MathVision [37], and ScienceQA [23]. This benchmark suite allows us to examine whether language-side Reasoning Vectors remain effective not only in pure-text reasoning, but also in multimodal reasoning tasks with VLMs. More details on the datasets and evaluation protocols are provided in Appendix B.1.

Implementation details. For a fair comparison, all LIFT variants use the same zero-shot CoT inference setup as the baseline VLM; they differ only by the injected Reasoning Vector and the selected intervention layer. Vector extraction follows the source definitions in Section 3.2. For each evaluation benchmark, we extract the Reasoning Vector from a corresponding support set: GSM8K, MathVista, and MathVision use a GSM8K support set; CommonsenseQA and StrategyQA use support sets from their own training splits; and ScienceQA uses a CommonsenseQA support set.

For static vector transfer, the intervention strength μ is fixed to 1.0. For learnable vector adaptation, only the injected vectors are optimized, and the learning rate is set to 1×10^{-4}. For the layer-selection procedure described in Section 3.4, we randomly sample 100 training examples as a development subset to choose the best injection layer, and then use the selected layer for full evaluation.

4.2 Main Results

4.2.1 Reasoning Vectors Transfer Effectively Across Modalities

Both LLM-derived and VLM-derived vectors improve the average score over the baseline, with LLM-derived vectors giving stronger and more consistent gains. As shown in Table 1, LLM-derived vectors improve the average score from 66.6/67.8 to 68.3/69.0 on Qwen2.5-VL/InternVL2.5. Notably, LLM-derived vectors outperform 4-shot CoT in average score on both VLMs, while keeping the

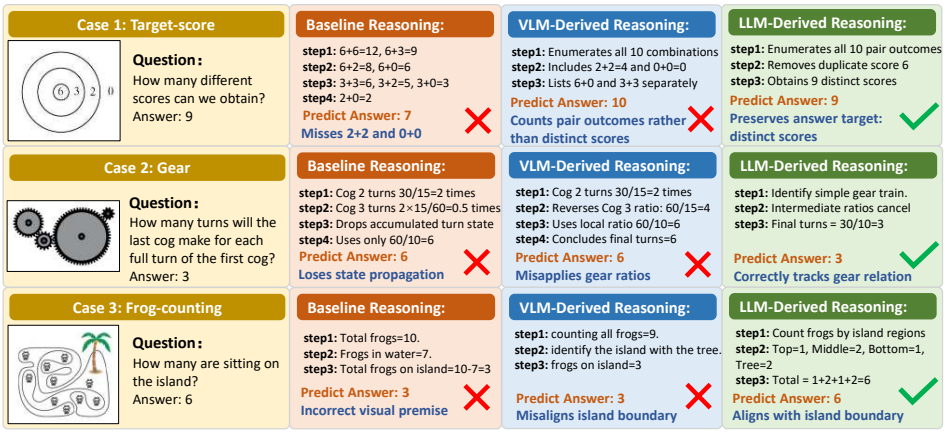


Figure 3: Representative Case Studies of Reasoning Repairs.

zero-shot CoT inference setup unchanged except for vector injection. These results show that static Reasoning Vector intervention can improve VLM reasoning without updating model parameters. Appendix C provides split sizes and bootstrap confidence intervals for the main accuracy gains. Appendix D.2 uses same-norm random vectors to test whether the extracted direction matters, beyond simply adding a vector with the same norm.

We further observe cross-modal transfer. The Reasoning Vectors evaluated on multimodal benchmarks in Table 1 are extracted from text-only support sets. Specifically, MathVista and MathVision use vectors extracted from GSM8K, while ScienceQA uses vectors extracted from CommonsenseQA. Despite this text-only extraction, LLM-derived vectors improve all three multimodal benchmarks on both VLMs. On MathVista/MathVision, they raise scores from 68.5/25.0 to 69.1/26.6 on Qwen2.5-VL-7B-Instruct and from 62.8/22.7 to 64.0/25.0 on InternVL2.5-8B. These results indicate that language-side reasoning vectors extracted from text-only data can still improve multimodal reasoning performance.

4.2.2 LLM-derived Vectors Provide More Reliable Interventions than VLM-derived Vectors

Vector source clearly affects performance. LLM-derived vectors outperform VLM-derived vectors on both VLMs, reaching 68.3/69.0 versus 67.5/68.3 on Qwen2.5-VL/InternVL2.5, and achieve higher scores on most datasets. This suggests that VLM-derived vectors are still useful but less reliable under our extraction protocol. A possible explanation is that multimodal alignment changes the organization of language-side reasoning representations in the VLM, making VLM-extracted directions less stable as interventions than reasoning-conditioned directions extracted from the base LLM.

We further examine representative MathVision cases to understand how vector source affects multimodal reasoning. As shown in Figure 3, LLM-derived vectors often repair intermediate reasoning errors rather than merely changing the final answer. In *target-score*, the baseline misses repeated-hit cases such as $2+2$ and $0+0$, while the VLM-derived vector counts pair outcomes instead of distinct scores. The LLM-derived vector preserves both repeated-hit enumeration and score deduplication, giving the correct answer 9. In *gear*, the baseline and VLM-derived vector use local gear ratios but fail to propagate the full chain, whereas the LLM-derived vector preserves the end-to-end relation. In *frog-counting*, the baseline relies on a subtractive count from frogs assumed to be in the water, while the VLM-derived vector narrows the island to the palm-tree region. The LLM-derived vector instead follows the island boundary and sums frogs by region. These cases suggest that LLM-derived vectors can influence intermediate reasoning behavior and help keep the response aligned with the question target. Additional examples with full reasoning traces are provided in Appendix F.

Overall, the quantitative results and case studies suggest that LLM-derived vectors provide **more reliable interventions** than VLM-derived vectors. Rather than merely changing final predictions, they appear to help preserve reasoning steps that remain aligned with the target question.

4.2.3 LLM-derived Vectors Outperform VLM-MM Vectors

Table 2 compares VLM-side extraction with text-only and multimodal support sets. Compared with VLM-derived vectors, VLM-MM improves the average score from 56.8/60.1 to 57.0/60.6 on Qwen2.5-VL/InternVL2.5. The gain mainly comes from MathVision, where scores increase from 25.7/23.4 to 26.0/24.7. This suggests that, with the source model fixed to the VLM, vectors derived from the same multimodal benchmark are more effective on average than those extracted from text-only support sets.

However, LLM-derived vectors still achieve the best average performance. Across MathVision and ScienceQA, the averages are 57.4/60.9 for LLM-derived, 57.0/60.6 for VLM-MM, and 56.8/60.1 for VLM-derived on Qwen2.5-VL/InternVL2.5. This comparison gives VLM-side extraction a favorable setting, since VLM-MM extracts vectors from the target VLM using support sets from the same multimodal benchmark used for evaluation. The lower performance suggests that the advantage of LLM-derived vectors comes from the source model, not just the extraction data.

This ordering, LLM-derived $>$ VLM-MM $>$ VLM-derived, suggests both extraction data and source model matter. VLM-MM is more effective than VLM-derived on the corresponding multimodal benchmarks, but still remains below LLM-derived on average. One possible explanation is that multimodal alignment changes the organization of language-side reasoning representations in the VLM. As a result, even language-side vectors extracted from the VLM may already reflect multimodal alignment effects, making VLM-MM useful for the corresponding multimodal benchmark but less stable as a general steering direction than pre-alignment LLM-derived vectors.

Table 2: Accuracy (%) of VLM multimodal extraction on multimodal reasoning benchmarks.

Model	Method	MVision	SciQA	Avg.
Qwen2.5-VL	Baseline	25.0	87.8	56.4
	LLM-derived	**26.6**	**88.1**	**57.4**
	VLM-derived	25.7	87.8	56.8
	VLM-MM	<u>26.0</u>	<u>87.9</u>	<u>57.0</u>
InternVL2.5	Baseline	22.7	96.5	59.6
	LLM-derived	**25.0**	**96.8**	**60.9**
	VLM-derived	23.4	<u>96.7</u>	60.1
	VLM-MM	<u>24.7</u>	96.4	<u>60.6</u>

4.2.4 Meaningful Initialization Matters for Learnable Vector Adaptation

Learnable vector adaptation is most effective when initialized from LLM-derived vectors. As shown in Table 1, LLM-learnable achieves the best average score on both VLMs, improving static LLM-derived vectors from 68.3/69.0 to 69.1/69.4 on Qwen2.5-VL/InternVL2.5. This suggests that LLM-derived vectors already provide useful reasoning directions, and lightweight training can further adapt them to the target VLM.

The comparison between LLM-learnable and VLM-learnable shows that the initialization source matters. Although both settings optimize injected vectors with the VLM backbone fixed, LLM-learnable consistently outperforms VLM-learnable, reaching 69.1/69.4 versus 67.2/66.6 on Qwen2.5-VL/InternVL2.5. On InternVL2.5-8B, VLM-learnable even falls below the baseline, indicating that learnable adaptation is sensitive to initialization quality.

Table 3 further compares LLM-derived initialization with random initialization under the same adaptation protocol. Replacing LLM-derived initialization with random initialization reduces the average score from 69.1/69.4 to 66.7/68.3 on Qwen2.5-VL/InternVL2.5. This shows that the gains of LLM-learnable are not merely due to adding trainable activation-level parameters, but also rely on a useful language-side reasoning-direction initialization.

Table 3: Initialization ablation for learnable vector adaptation. Random init replaces the extracted Reasoning Vector with a random vector under the same training setting.

Model	Method	GSM8K	MVista	MVision	CSQA	SQA	SciQA	Avg.
Qwen2.5-VL	LLM-learnable	**85.6**	**71.8**	**27.3**	**78.3**	**62.4**	**88.9**	**69.1**
	Random init	83.2	68.6	24.7	77.4	58.8	87.9	66.7
	VLM-learnable	**83.2**	**69.5**	**24.7**	**77.7**	**60.3**	**87.9**	**67.2**
	Random init	83.0	68.6	**24.7**	77.4	58.8	**87.9**	66.7
InternVL2.5	LLM-learnable	**78.0**	**64.6**	**25.3**	**83.2**	**67.6**	**97.4**	**69.4**
	Random init	77.4	63.3	24.1	83.1	65.4	96.7	68.3
	VLM-learnable	78.4	**63.4**	**20.1**	79.0	**62.4**	**96.0**	**66.6**
	Random init	**78.8**	62.5	19.7	**79.4**	59.7	95.8	66.0

A possible explanation is that VLM-derived vectors are affected by multimodal alignment even when computed from language-side answer states. They may therefore be less stable as initialization. In contrast, LLM-derived vectors

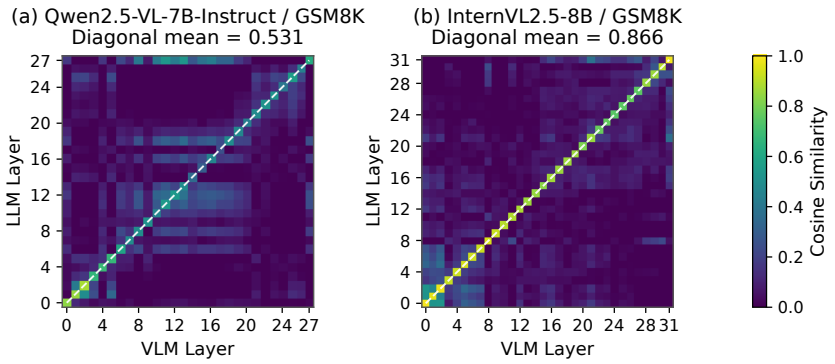


Figure 4: Cross-layer cosine similarity between LLM- and VLM-derived vectors on GSM8K.

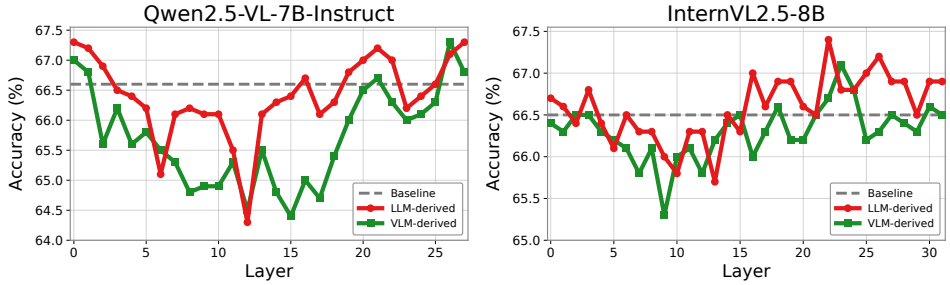


Figure 5: Layer-wise performance of LLM- and VLM-derived vectors averaged over six datasets.

provide a cleaner language-side starting point, so adaptation mainly aligns an existing reasoning direction to the target VLM rather than discovering one from scratch.

4.3 Representation Alignment and Layer-wise Analysis

Cross-layer vector alignment. The cosine similarity between LLM-derived and VLM-derived vectors in Fig. 4 shows diagonal concentration in both heatmaps, indicating that vectors from similar depths align better between the base LLM and the corresponding VLM. This suggests that multimodal scaling partially preserves the layer-wise organization of language-side reasoning representations.

However, the strength of this correspondence differs across models. InternVL2.5-8B exhibits a strong diagonal pattern on GSM8K, with an average diagonal similarity of 0.866, suggesting close layer-wise LLM–VLM alignment. In contrast, Qwen2.5-VL-7B-Instruct shows a more diffuse pattern, with an average diagonal similarity of 0.531 and stronger off-diagonal similarities in middle layers. This suggests that reasoning-related structures are partially preserved but may be redistributed or less cleanly aligned after multimodal scaling. Overall, this layer-wise correspondence helps explain why LLM-derived Reasoning Vectors can transfer to VLMs, while the weaker and more diffuse patterns also motivate layer-specific intervention and layer selection.

Layer-wise sensitivity. Fig. 5 shows Reasoning Vector injection is strongly layer-sensitive. Both LLM-derived and VLM-derived vectors fluctuate across layers, suggesting their effect depends on layer-specific VLM organization rather than applying uniformly at all depths. Despite this variation, the source-level trend remains clear: LLM-derived vectors generally outperform VLM-derived vectors across layers. This is consistent with Table 1, where VLM-derived vectors remain useful but LLM-derived vectors provide stronger and more reliable interventions after multimodal scaling.

5 Conclusion and Limitations

We presented LIFT, a lightweight vector-intervention method that transfers language-side Reasoning Vectors from a base LLM to its corresponding VLM. Across text-only and multimodal reasoning benchmarks, LIFT improves VLM reasoning without updating the VLM backbone. These results

suggest that reasoning-related directions formed in the base LLM can remain usable after multimodal scaling, even though multimodal alignment may partially weaken or reorganize them in the corresponding VLM.

LIFT also has several limitations. Our experiments cover two VLM families, and future work should evaluate whether the findings generalize to more architectures, alignment procedures, and multimodal tasks. LIFT requires support examples to extract Reasoning Vectors, and its effectiveness is sensitive to the selected intervention layer. In addition, because LIFT operates on language-side hidden states, it cannot directly correct errors caused by inaccurate visual perception or grounding. A promising direction is to combine reasoning-vector intervention with stronger visual grounding methods.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (62576091), Jiangsu Province Carbon Peak Carbon Neutrality Science and Technology Innovation Special Fund Project (Grant No. BT2025029), the Southeast University Big Data Computing Center, and the Southeast University Kunpeng & Ascend Center of Cultivation. The authors declare no competing interests.

References

[1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35: 23716–23736, 2022.

[2] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report, 2025. URL `https://arxiv.org/abs/2502.13923`.

[3] Shiqi Chen, Jinghan Zhang, Tongyao Zhu, Wei Liu, Siyang Gao, Miao Xiong, Manling Li, and Junxian He. Bring reason to vision: Understanding perception and reasoning through model merging. *arXiv preprint arXiv:2505.05464*, 2025.

[4] Yu Chen, Caorui Li, Ziyu Xiong, Yidong Wang, Mingqi Gao, Shuman Liu, Biao Liu, Chunfeng Yang, Anxiang Zeng, Haibo Zhang, and Chaofan Chen. Omnireasoner: Thinking with long audio-video via native tool use, 2026. URL `https://arxiv.org/abs/2607.19339`.

[5] Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, et al. Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling. *arXiv preprint arXiv:2412.05271*, 2024.

[6] 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. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[7] Wenliang Dai, Junnan Li, Dongxu Li, Anthony Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale N Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning. *Advances in neural information processing systems*, 36:49250–49267, 2023.

[8] Yifan Du, Zikang Liu, Yifan Li, Wayne Xin Zhao, Yuqi Huo, Bingning Wang, Weipeng Chen, Zheng Liu, Zhongyuan Wang, and Ji-Rong Wen. Virgo: A preliminary exploration on reproducing o1-like mllm. *arXiv preprint arXiv:2501.01904*, 2025.

[9] Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361, 2021.

[10] Jiawei Guo, Tianyu Zheng, Yizhi Li, Yuelin Bai, Bo Li, Yubo Wang, King Zhu, Graham Neubig, Wenhu Chen, and Xiang Yue. Mammoth-vl: Eliciting multimodal reasoning with instruction tuning at scale. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13869–13920, 2025.

[11] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

[12] Shaohan Huang, Li Dong, Wenhui Wang, Yaru Hao, Saksham Singhal, Shuming Ma, Tengchao Lv, Lei Cui, Owais Khan Mohammed, Barun Patra, et al. Language is not all you need: Aligning perception with language models. *Advances in Neural Information Processing Systems*, 36: 72096–72109, 2023.

[13] Wenxuan Huang, Bohan Jia, Zijie Zhai, Shaosheng Cao, Zheyu Ye, Fei Zhao, Zhe Xu, Xu Tang, Yao Hu, and Shaohui Lin. Vision-r1: Incentivizing reasoning capability in multimodal large language models. *arXiv preprint arXiv:2503.06749*, 2025.

[14] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022.

[15] Yuchu Jiang, Jiale Fu, Chenduo Hao, Xinting Hu, Yingzhe Peng, Xin Geng, and Xu Yang. Mimic in-context learning for multimodal tasks, 2025. URL https://arxiv.org/abs/2504.08851.

[16] Pegah Khayatan, Mustafa Shukor, Jayneel Parekh, Arnaud Dapogny, and Matthieu Cord. Analyzing finetuning representation shift for multimodal llms steering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2206–2216, 2025.

[17] Hongkang Li, Yihua Zhang, Shuai Zhang, Meng Wang, Sijia Liu, and Pin-Yu Chen. When is task vector provably effective for model editing? a generalization analysis of nonlinear transformers. *arXiv preprint arXiv:2504.10957*, 2025.

[18] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pages 19730–19742. PMLR, 2023.

[19] Li Li, Ziyi Wang, Yongliang Wu, Jianfei Cai, and Xu Yang. Cot vectors: Transferring and probing the reasoning mechanisms of llms, 2025. URL https://arxiv.org/abs/2510.00579.

[20] Nuo-Jing Liang, Xiao-Tong Yuan, and Pan Zhou. Understanding the out-of-distribution generalization of chain-of-thought reasoning in llms, 2026. URL https://arxiv.org/abs/2611.06438.

[21] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023.

[22] Sheng Liu, Haotian Ye, Lei Xing, and James Zou. In-context vectors: Making in context learning more effective and controllable through latent space steering. *arXiv preprint arXiv:2311.06668*, 2023.

[23] Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in neural information processing systems*, 35: 2507–2521, 2022.

[24] Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*, 2023.

[25] Ruilin Luo, Zhuofan Zheng, Yifan Wang, Yiyao Yu, Xinzhe Ni, Zicheng Lin, Jin Zeng, and Yujiu Yang. Ursa: Understanding and verifying chain-of-thought reasoning in multimodal mathematics. *arXiv e-prints*, pages arXiv–2501, 2025.

[26] Oscar Mañas, Pierluca D'Oro, Koustuv Sinha, Adriana Romero-Soriano, Michal Drozdzal, and Aishwarya Agrawal. Controlling multimodal llms via reward-guided decoding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1391–1401, 2025.

[27] Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. Task arithmetic in the tangent space: Improved editing of pre-trained models. *Advances in Neural Information Processing Systems*, 36:66727–66754, 2023.

[28] Nina Panickssery, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Matt Turner. Steering llama 2 via contrastive activation addition. *arXiv preprint arXiv:2312.06681*, 2023.

[29] Neale Ratzlaff, Man Luo, Xin Su, Vasudev Lal, and Phillip Howard. Training-free mitigation of language reasoning degradation after multimodal instruction tuning. In *Proceedings of the AAAI Symposium Series*, volume 5, pages 384–388, 2025.

[30] Hao Shao, Shengju Qian, Han Xiao, Guanglu Song, Zhuofan Zong, Letian Wang, Yu Liu, and Hongsheng Li. Visual cot: Advancing multi-modal language models with a comprehensive dataset and benchmark for chain-of-thought reasoning. *Advances in Neural Information Processing Systems*, 37:8612–8642, 2024.

[31] Nishant Subramani, Nivedita Suresh, and Matthew E Peters. Extracting latent steering vectors from pretrained language models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 566–581, 2022.

[32] Linzhuang Sun, Hao Liang, Jingxuan Wei, Bihui Yu, Tianpeng Li, Fan Yang, Zenan Zhou, and Wentao Zhang. Mm-verify: Enhancing multimodal reasoning with chain-of-thought verification. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14100–14115, 2025.

[33] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, 2019.

[34] Omkar Thawakar, Dinura Dissanayake, Ketan Pravin More, Ritesh Thawkar, Ahmed Heakl, Noor Ahsan, Yuhao Li, Ilmuz Zaman Mohammed Zumri, Jean Lahoud, Rao Muhammad Anwer, et al. Llamav-o1: Rethinking step-by-step visual reasoning in llms. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24290–24315, 2025.

[35] Eric Todd, Millicent L Li, Arnab Sen Sharma, Aaron Mueller, Byron C Wallace, and David Bau. Function vectors in large language models. *arXiv preprint arXiv:2310.15213*, 2023.

[36] Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering. *arXiv preprint arXiv:2308.10248*, 2023.

[37] Ke Wang, Junting Pan, Weikang Shi, Zimu Lu, Houxing Ren, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Measuring multimodal mathematical reasoning with math-vision dataset. *Advances in Neural Information Processing Systems*, 37:95095–95169, 2024.

[38] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[39] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[40] Guowei Xu, Peng Jin, Ziang Wu, Hao Li, Yibing Song, Lichao Sun, and Li Yuan. Llava-cot: Let vision language models reason step-by-step. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2087–2098, 2025.

[41] Haolin Yang, Hakaze Cho, Yiqiao Zhong, and Naoya Inoue. Unifying attention heads and task vectors via hidden state geometry in in-context learning. *arXiv preprint arXiv:2505.18752*, 2025.

[42] Yi Yang, Xiaoxuan He, Hongkun Pan, Xiyan Jiang, Yan Deng, Xingtao Yang, Haoyu Lu, Dacheng Yin, Fengyun Rao, Minfeng Zhu, et al. R1-onevision: Advancing generalized multimodal reasoning through cross-modal formalization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2376–2385, 2025.

[43] Huanjin Yao, Jiaxing Huang, Wenhao Wu, Jingyi Zhang, Yibo Wang, Shunyu Liu, Yingjie Wang, Yuxin Song, Haocheng Feng, Li Shen, et al. Mulberry: Empowering mllm with o1-like reasoning and reflection via collective monte carlo tree search. *arXiv preprint arXiv:2412.18319*, 2024.

[44] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

[45] Haowei Zhang, Jianzhe Liu, Zhen Han, Shuo Chen, Bailan He, Volker Tresp, Zhiqiang Xu, and Jindong Gu. Visual question decomposition on multimodal large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1926–1949, 2024.

[46] Jason Zhang and Scott W Viteri. Uncovering latent chain of thought vectors in large language models. In *Workshop on Neural Network Weights as a New Data Modality*, 2025.

[47] Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. Multimodal chain-of-thought reasoning in language models. *arXiv preprint arXiv:2302.00923*, 2023.

[48] Kesen Zhao, Beier Zhu, Junbao Zhou, Xingyu Zhu, Zhongqi Yue, and Hanwang Zhang. Thinking with images as continuous actions: Numerical visual chain-of-thought. *arXiv preprint arXiv:2602.23959*, 2026.

[49] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

[50] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*, 2023.

[51] Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, et al. Representation engineering: A top-down approach to ai transparency. *arXiv preprint arXiv:2310.01405*, 2023.

A Derivation of Reasoning Vector

When the model reasons with a reasoning trace, the generation of an answer token $a \in A$ depends not only on the question tokens but also on the intermediate reasoning tokens. Let Q, T, and A denote the question, reasoning trace, and answer tokens, respectively. For a simplified single-head self-attention module, we denote the corresponding key and value matrices as (K_Q, V_Q), (K_T, V_T), and (K_A, V_A). The self-attention output for answer token a under the reasoning-conditioned input (Q, T, A) can be written as

$$\begin{aligned}
&\mathrm{SA}\left(a, [K_Q, K_T, K_A], [V_Q, V_T, V_A]\right) \\
&= \frac{\exp(aK_Q^\top)}{Z_{\text{total}}} V_Q + \frac{\exp(aK_T^\top)}{Z_{\text{total}}} V_T + \frac{\exp(aK_A^\top)}{Z_{\text{total}}} V_A \\
&= \frac{Z_Q}{Z_{\text{total}}} \cdot \underbrace{\frac{\exp(aK_Q^\top)}{Z_Q} V_Q}_{\mathrm{SA}(a,[K_Q],[V_Q])} + \frac{Z_T}{Z_{\text{total}}} \cdot \underbrace{\frac{\exp(aK_T^\top)}{Z_T} V_T}_{\mathrm{SA}(a,[K_T],[V_T])} + \frac{Z_A}{Z_{\text{total}}} \cdot \underbrace{\frac{\exp(aK_A^\top)}{Z_A} V_A}_{\mathrm{SA}(a,[K_A],[V_A])}.
\end{aligned} \tag{6}$$

where $Z_Q = \sum \exp(aK_Q^\top)$, $Z_T = \sum \exp(aK_T^\top)$, $Z_A = \sum \exp(aK_A^\top)$, and $Z_{\text{total}} = Z_Q + Z_T + Z_A$.

In contrast, without the reasoning trace, the answer token only attends to the question and answer tokens. The self-attention output under the input (Q, A) is

$$\mathrm{SA}\left(a, [K_Q, K_A], [V_Q, V_A]\right) = \frac{Z_Q}{Z_Q + Z_A} \cdot \mathrm{SA}(a, [K_Q], [V_Q]) + \frac{Z_A}{Z_Q + Z_A} \cdot \mathrm{SA}(a, [K_A], [V_A]). \tag{7}$$

By comparing Eq. 6 and Eq. 7, the reasoning-conditioned attention output can be written as a standard attention term plus an additive reasoning-conditioned shift:

$$\begin{aligned}
&\mathrm{SA}\left(a, [K_Q, K_T, K_A], [V_Q, V_T, V_A]\right) \\
&= \underbrace{\mathrm{SA}\left(a, [K_Q, K_A], [V_Q, V_A]\right)}_{\text{Standard Attention}} + \mu \cdot \underbrace{\left(\mathrm{SA}(a, [K_T], [V_T]) - \mathrm{SA}\left(a, [K_Q, K_A], [V_Q, V_A]\right)\right)}_{\text{Reasoning Vector}},
\end{aligned} \tag{8}$$

where $\mu = \frac{Z_T}{Z_{\text{total}}}$.

Equation 8 shows that, in this simplified setting, the reasoning trace acts as an additive term in the attention output of the answer tokens. In other words, the contribution of the reasoning to answer generation can be summarized as an additive change in the model's internal states. LIFT operationalizes this view through answer-token hidden-state differences under the Reasoner–Solver formulation. Recent theoretical work [20] characterizes Chain-of-Thought reasoning as compositional out-of-distribution generalization, where reusable single-step reasoning transitions are composed to solve longer tasks. This perspective is complementary to our derivation: while prior work studies the generalization mechanism of multi-step reasoning, our analysis focuses on the representation-level changes induced by explicit reasoning traces, motivating the construction of Reasoning Vectors through the Reasoner–Solver difference.

B Implementation Details

B.1 Datasets and Evaluation Protocols

We use Qwen2.5-7B-Instruct as the base LLM for Qwen2.5-VL-7B-Instruct, and InternLM2.5-7B-Chat as the base LLM for InternVL2.5-8B. We evaluate LIFT on six reasoning benchmarks: GSM8K, MathVista, MathVision, CommonsenseQA, StrategyQA, and ScienceQA. These benchmarks cover both text-only and multimodal reasoning scenarios. For each benchmark, we use a held-out evaluation split for final testing. Unless otherwise specified, the main evaluation follows the zero-shot CoT setting used in the main table. All methods are evaluated under the same zero-shot CoT prompt, decoding setup, evaluation split, and answer parser. Support sets are used only before evaluation to extract Reasoning Vectors or adapt learnable vectors, and are never included in the test set.

GSM8K. GSM8K is a text-only mathematical reasoning benchmark. We use the training split to construct support examples and evaluate on the test split. The final numerical answer is parsed from the model response, with `\boxed{}` used as the preferred answer format whenever applicable.

MathVista. MathVista is a multimodal mathematical reasoning benchmark. We evaluate on the testmini split, which contains 1,000 examples.

MathVision. MathVision is a visually grounded mathematical reasoning benchmark. It contains problems that require understanding diagrams, charts, geometric structures, or other visual contexts before performing mathematical reasoning. We follow the evaluation split and answer parsing protocol used in the main experiments, and parse the final answer from the model response.

CommonsenseQA. CommonsenseQA is a text-only multiple-choice commonsense reasoning benchmark. We evaluate on 1,221 examples.

StrategyQA. StrategyQA is a text-only binary reasoning benchmark that requires implicit multi-hop reasoning. The model is required to produce a final answer in a constrained yes-or-no format. Accuracy is computed by matching the parsed answer with the ground-truth label.

ScienceQA. ScienceQA is a multimodal science reasoning benchmark. It contains questions that may require scientific knowledge, visual understanding, or both. We evaluate on the subset used in the main experiments and follow the same answer parsing protocol.

Scope of comparisons. LIFT is designed as a lightweight intervention and analysis tool for studying whether language-side Reasoning Vectors from a base LLM remain usable inside the corresponding VLM after multimodal scaling. Our main comparison therefore focuses on different vector sources under the same protocol. We do not aim to replace training-based VLM adaptation methods or reinforcement learning.

The average score reported in the main tables is computed as the arithmetic mean over the evaluated benchmarks. Table 4 summarizes the evaluation sizes and support sources used for each benchmark.

Table 4: Evaluation split sizes and support sources used in our experiments. Support examples are used only for offline Reasoning Vector extraction or learnable adaptation and are never appended to evaluation prompts.

Benchmark	Modality	#Eval. examples	Support source
GSM8K	Text	1319	GSM8K
MathVista	Image-text	1000	GSM8K
MathVision	Image-text	304	GSM8K
CommonsenseQA	Text	1221	CommonsenseQA
StrategyQA	Text	687	StrategyQA
ScienceQA	Image-text	4241	CommonsenseQA

B.2 Prompting and Generation Settings

During evaluation, the baseline and all LIFT variants use the same zero-shot CoT setting, decoding configuration, and answer parser. Support examples and extracted traces are used only before evaluation for vector extraction or learnable adaptation; they are not included in the evaluation set or used as in-context demonstrations. Therefore, the reported performance differences are not caused by prompt changes.

For Reasoning Vector extraction, we construct two teacher-forced paths from each support set. The Reasoner path receives the question, a model-generated reasoning trace, and the final answer, while the Solver path receives only the question and the final answer. This with-CoT versus without-CoT comparison is used only offline to compute the answer-token hidden-state difference that defines the Reasoning Vector.

For mathematical tasks, we ask the model to place the final answer within `\boxed{}` whenever applicable. For multiple-choice tasks, we parse the predicted option. For yes-or-no tasks, we parse

the final yes-or-no answer. For all LIFT evaluations, we use the same evaluation prompt as the corresponding non-intervened baseline, so that performance changes can be attributed to the internal intervention rather than to prompt changes. Table 5 lists the prompt templates used in our experiments.

Table 5: Prompt templates used in our experiments.

Setting	Prompt Template
Direct Answer	`Please answer the question directly. Put your final answer within \boxed{}.`
Reasoning Trace	`Please reason step by step. Put your final answer within \boxed{}.`
Multiple Choice	`Please choose the correct option. Put your final answer within \boxed{}.`
Yes/No	`Please answer the question with yes or no. Put your final answer within \boxed{}.`

For all baseline and LIFT evaluations, we use deterministic decoding with beam size 1 and sampling disabled. The maximum number of new tokens is set to 512 for text-only benchmarks and 4096 for multimodal benchmarks, as summarized in Table 6.

Table 6: Generation configuration used for evaluation.

Parameter	Value
Number of beams	1
Maximum new tokens	512 / 4096
Length penalty	0.0
Random seed	42

B.3 Support Set Construction and Vector Extraction

Unless otherwise specified, we sample 500 support examples for each support source. Each support example contains a question Q, a model-generated reasoning trace T, and a final answer A. The reasoning trace T is generated by the corresponding source model under the step-by-step CoT prompt.

For each vector source, we keep only support examples that are answered correctly by that source model. Thus, LLM-derived vectors are extracted from examples correctly answered by the corresponding base LLM, while VLM-derived vectors are extracted from examples correctly answered by the target VLM. As a result, LLM-derived and VLM-derived vectors may be extracted from different correct-example subsets, but they follow the same Reasoner–Solver extraction protocol.

For GSM8K, MathVista, and MathVision, we use GSM8K support examples to extract mathematical reasoning vectors. For CommonsenseQA and StrategyQA, we use support examples from their corresponding training splits. For ScienceQA, we use CommonsenseQA support examples to evaluate whether language-side commonsense reasoning vectors transfer to multimodal science reasoning. These support-source choices are summarized together with the evaluation sizes in Table 4.

B.4 Learnable Vector Adaptation

In the learnable adaptation setting, the target VLM backbone remains frozen and only the injected vector parameters θ_l are optimized. The learnable vectors are initialized from the corresponding extracted Reasoning Vectors unless otherwise specified. We use the same offline support set as vector extraction and optimize only the alignment objective described in the main text; no language-model cross-entropy loss is used for vector adaptation.

We train the vector parameters for 6 epochs using AdamW with learning rate 1×10^{-4}. The batch size is 1, with gradient accumulation over 4 steps. We use weight decay 1×10^{-3}, warmup ratio 0.1, gradient clipping with maximum norm 1.0, and 16-bit mixed precision. The VLM backbone and vision encoder remain frozen throughout adaptation. Table 7 summarizes the hyperparameters used for learnable vector adaptation.

Table 7: Hyperparameters for learnable vector adaptation.

Hyperparameter	Value
Trainable parameters	Injected vectors θ_l only
Backbone parameters	Frozen
Optimizer	AdamW
Learning rate	1×10^{-4}
Batch size	1
Gradient accumulation	4
Training epochs	6
Weight decay	1×10^{-3}
Warmup ratio	0.1
Gradient clipping	1.0
Training objective	Alignment loss only
Precision	16-mixed
Initialization	LLM-derived or VLM-derived vector

B.5 Reproducibility Details

We use random seed 42 for support-set sampling, layer-selection subset sampling, and learnable adaptation unless otherwise specified. All baseline and LIFT evaluations use deterministic decoding with sampling disabled. Experiments are implemented in PyTorch and run with `torch.float16` for static evaluation and 16-bit mixed precision for learnable adaptation. We run experiments on NVIDIA RTX 3090 GPUs with 24GB memory and NVIDIA RTX A6000 GPUs with 48GB memory. All experiments fit on a single GPU. The wall-clock time varies across benchmarks depending on the evaluation-set size and maximum generation length: text-only benchmarks with a 512-token limit are faster, while multimodal benchmarks with a 4096-token limit require longer evaluation.

C Statistical Uncertainty of Main Accuracy Differences

We report paired bootstrap confidence intervals to quantify the statistical uncertainty of the main accuracy differences. For each comparison, we use per-example correctness pairs from the same evaluation split and compute the paired accuracy difference under bootstrap resampling. We use 10,000 bootstrap resamples for each comparison.

Table 8: Evaluation split sizes used in our experiments.

Benchmark	#Examples
GSM8K	1319
MathVista	1000
MathVision	304
CommonsenseQA	1221
StrategyQA	687
ScienceQA	4241

For macro-average comparisons, we first compute the paired accuracy difference within each model–benchmark pair and then average the differences equally across all evaluated model–benchmark pairs. All deltas are reported in percentage points.

Table 9: Paired bootstrap confidence intervals for macro-average accuracy differences.

Comparison	Avg. Delta	95% CI
LLM-derived − Baseline	+1.45	[0.45, 2.45]
VLM-derived − Baseline	+0.70	[0.12, 1.28]
LLM-derived − VLM-derived	+0.75	[0.15, 1.35]

Table 10: Additional-seed robustness check for frozen Reasoning Vector transfer. All results are reported as accuracy (%).

Model	Method	GSM8K	MVista	MVision	CSQA	SQA	SciQA	Avg.
Qwen2.5-VL	Baseline	82.2	68.5	25.0	77.3	58.6	87.8	66.6
	LLM-derived	84.4	69.5	26.7	78.3	63.4	88.0	68.4
	VLM-derived	83.2	70.6	25.7	77.8	60.3	87.6	67.5
InternVL2.5	Baseline	77.1	62.8	22.7	82.3	65.6	96.5	67.8
	LLM-derived	78.5	63.9	25.0	83.5	67.2	96.8	69.2
	VLM-derived	77.5	63.1	24.3	82.9	65.8	96.7	68.4

Table 11: Three-seed robustness evaluation on Qwen2.5-VL-7B-Instruct. Results are reported as mean $\pm$ standard deviation of accuracy (%).

Model	Method	GSM8K	MVista	MVision	CSQA	SQA	SciQA
Qwen2.5-VL	LLM-derived	84.3 ± 0.8	69.9 ± 0.7	26.5 ± 0.7	78.4 ± 0.4	62.5 ± 0.9	88.4 ± 0.3
	VLM-derived	82.5 ± 0.5	69.6 ± 1.0	25.0 ± 0.7	77.8 ± 0.4	60.3 ± 0.5	87.5 ± 0.5

Table 9 shows that both LLM-derived and VLM-derived Reasoning Vectors improve macro-average accuracy over the baseline. LLM-derived vectors also achieve higher macro-average accuracy than VLM-derived vectors under this paired bootstrap analysis.

D Robustness Checks and Random-vector Control

D.1 Additional-seed Robustness Check

We additionally repeat the experiment with one different random seed for support-set construction and layer-selection development examples, while keeping the evaluation split fixed. Table 10 shows that the main trend remains unchanged: LLM-derived Reasoning Vectors improve over the baseline and achieve higher average accuracy than VLM-derived vectors on both VLM backbones.

We further evaluate the robustness of LIFT on Qwen2.5-VL-7B-Instruct using three random seeds (42, 347, and 125). In each run, the support examples and held-out subsets used for layer selection are independently sampled. We report the mean and standard deviation across the three runs in Table 11.

Across the three runs, LLM-derived vectors consistently outperform VLM-derived vectors on all six benchmarks. The relatively small standard deviations further demonstrate that the performance improvements are robust to variations in support-set sampling and layer selection.

D.2 Same-norm Random-vector Control

We further conduct a same-norm random-vector control to test whether the gains of LIFT can be explained by arbitrary activation perturbations. For each selected intervention layer l, we sample a Gaussian random vector $\epsilon_l \sim \mathcal{N}(0, I)$ and rescale it to have the same ℓ_2 norm as the corresponding LLM-derived Reasoning Vector r_l:

$$r_l^{\text{rand}} = \frac{\epsilon_l}{\|\epsilon_l\|_2} \cdot \|r_l\|_2. \tag{9}$$

We then inject the same-norm random vector into the target VLM using the same selected layer, scaling coefficient, decoding setting, and evaluation protocol as the corresponding LIFT intervention:

$$\widetilde{x}_l = x_l + \mu r_l^{\text{rand}}. \tag{10}$$

This control preserves the perturbation magnitude while replacing the extracted Reasoning Vector direction with a random direction. It therefore tests whether the observed improvements are caused merely by adding a vector of similar scale to the hidden states.

As shown in Table 12, same-norm random vectors do not improve over the baseline and fail to reproduce the gains of LLM-derived Reasoning Vectors. On Qwen2.5-VL, the random-vector control

obtains an average accuracy of 66.4, compared with 66.6 for the baseline and 68.3 for LLM-derived vectors. On InternVL2.5, it obtains 66.1, below both the baseline average of 67.8 and the LLM-derived average of 69.0. These results suggest that LIFT's gains are not explained by perturbation magnitude alone; the Reasoning Vectors provide useful task-relevant information for intervention.

Table 12: Same-norm random-vector control. All results are reported as accuracy (%).

Model	Method	GSM8K	MathVista	MathVision	CSQA	SQA	SciQA	Avg.
Qwen2.5-VL	Baseline	82.2	68.5	25.0	77.3	58.6	87.8	66.6
	Same-norm random	82.1	68.2	25.0	77.3	58.2	87.8	66.4
	LLM-derived	**84.2**	**69.1**	**26.6**	**78.0**	**63.5**	**88.1**	**68.3**
InternVL2.5	Baseline	77.1	62.8	22.7	82.3	65.6	96.5	67.8
	Same-norm random	77.1	62.6	22.8	79.7	58.4	96.3	66.1
	LLM-derived	**78.1**	**64.0**	**25.0**	**83.1**	**67.1**	**96.8**	**69.0**

D.3 Reliability of Held-out Layer Selection

To evaluate the reliability of our layer-selection strategy, we compare the layer selected using a 100-example held-out subset with the oracle layer identified through an exhaustive sweep over candidate intervention layers. As shown in Table 13, the performance gap between the selected layer and the oracle layer is at most 0.3 percentage points across all four settings. These results demonstrate that our lightweight layer-selection procedure provides a close approximation to exhaustive layer search, without requiring evaluation on all candidate layers.

D.4 Sensitivity to Support-set Size

We further investigate the sensitivity of LIFT to the number of support examples by evaluating support-set sizes of 100, 200, and 500. As shown in Table 14, LIFT remains effective with substantially fewer support examples. Using only 100 examples already improves performance over the baseline across all reported settings. These results suggest that LIFT can extract useful Reasoning Vectors from relatively small support sets.

E Additional Attention-level Injection Results

To test whether LIFT can also work at attention-head outputs, we additionally evaluate attention-level injection. This variant extracts head-wise Reasoning Vectors from attention outputs and injects them into the corresponding attention-head outputs during inference. The target VLM backbone remains frozen, and the vision encoder is not modified. For each dataset, the injection layer is selected on a held-out development subset of 100 training examples and then fixed for final evaluation. No evaluation examples are used for layer selection.

For each attention head m at layer l, the attention-level vector is computed as

$$v_{\text{Attn}}^{(l,m)} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|A_i|}\sum_{a\in A_i}\left(o_{R,i}^{(l,m)}(a) - o_{S,i}^{(l,m)}(a)\right), \tag{11}$$

where $o_{R,i}^{(l,m)}(a)$ and $o_{S,i}^{(l,m)}(a)$ denote the output of attention head m for answer token a under the Reasoner and Solver inputs, respectively.

During inference, the intervention is applied to the attention-head output at each generated language-token position t:

$$\widetilde{o}^{(l,m)}(t) = o^{(l,m)}(t) + \mu \cdot v_{\text{Attn}}^{(l,m)}. \tag{12}$$

Here, $o^{(l,m)}(t)$ is the original output of attention head m at layer l, $\widetilde{o}^{(l,m)}(t)$ is the intervened output, and μ is the intervention strength. We set $\mu = 1.0$ for all attention-level injection experiments.

Table 15 reports the results of attention-level injection. Attention-level vectors improve over the baseline on average for both VLM backbones. LLM-derived vectors again achieve higher average performance than VLM-derived vectors. This provides additional evidence that the advantage of

Table 13: Comparison between held-out selected layers and oracle layers. Results are averaged over six benchmarks.

Model	Method	Selected	Oracle	Gap
Qwen2.5-VL	LLM-derived	68.3	68.3	0.0
	VLM-derived	67.5	67.8	0.3
InternVL2.5	LLM-derived	69.0	69.1	0.1
	VLM-derived	68.3	68.4	0.1

Table 14: Sensitivity to support-set size. Results are averaged over CSQA, SQA, and SciQA.

Model	Method	Baseline	100	200	500
Qwen2.5-VL	LLM-derived	74.6	75.7	75.9	76.5
	VLM-derived	74.6	75.1	75.3	75.3
InternVL2.5	LLM-derived	81.5	81.9	82.1	82.3
	VLM-derived	81.5	81.9	81.8	81.7

LLM-derived vectors is not specific to the hidden-state injection setting used in the main experiments, but can also appear when the intervention is applied to attention-head outputs.

Table 15: Additional results of Attention-level Injection. For each dataset, the injection layer is selected on a held-out development subset and then fixed for evaluation.

Model	Method	GSM8K	MVista	MVision	CSQA	SQA	SciQA	Avg.
Qwen2.5-VL	Baseline	82.2	68.5	25.0	77.3	58.6	87.8	66.6
	LLM-derived	**83.2**	**69.6**	**26.6**	**77.8**	**59.7**	**88.1**	**67.5**
	VLM-derived	82.9	69.5	26.3	77.6	59.4	88.0	67.3
InternVL2.5	Baseline	77.1	62.8	22.7	82.3	65.6	96.5	67.8
	LLM-derived	**78.1**	**63.9**	**25.3**	**83.2**	**67.4**	**96.7**	**69.1**
	VLM-derived	77.9	63.2	25.0	83.1	67.0	96.6	68.8

F Additional Case Studies with Full Reasoning Traces

We provide additional case studies with full reasoning traces. These examples complement the summarized cases in the main text by showing the outputs of the baseline, VLM-derived Reasoning Vector intervention, and LLM-derived Reasoning Vector intervention under the same zero-shot CoT prompt.

Case 1: Target-score

Question: By shooting two arrows at the shown target on the wall, how many different scores can we obtain?
Gold Answer: 9.

Baseline **Prediction: 7** ×

To determine the number of different scores that can be obtained by shooting two arrows at the given target, we need to consider all possible combinations of the numbers in the circles. The target has three concentric circles with the following numbers: - The innermost circle (center): 6 - The middle circle: 3 - The outermost circle: 2 and 0 We will calculate the score for each possible combination of two arrows: 1. Both arrows hit the center (6): - Score = $6 + 6 = 12$ 2. One arrow hits the center (6) and the other hits the middle (3): - Scores = $6 + 3 = 9$ or $3 + 6 = 9$ - Total unique score = 9 3. One arrow hits the center (6) and the other hits the outer circle (2 or 0): - Scores = $6 + 2 = 8$ or $6 + 0 = 6$ - Total unique scores = 8 and 6 4. Both arrows hit the middle circle (3): - Score = $3 + 3 = 6$ 5. One arrow hits the middle circle (3) and the other hits the outer circle (2 or 0): - Scores = $3 + 2 = 5$ or $3 + 0 = 3$ - Total unique scores = 5 and 3 6. Both arrows hit the outer circle (2 and 0): - Scores = $2 + 0 = 2$ or $0 + 2 = 2$ - Total unique score = 2 Now,

let's list all the unique scores: - 12 - 9 - 8 - 6 - 5 - 3 - 2 Counting these unique scores, we have: 12, 9, 8, 6, 5, 3, 2. There are 7 unique scores. Thus, the number of different scores that can be obtained is: $\boxed{7}$

VLM-derived **Prediction: 10 ×**

To determine the number of different scores that can be obtained by shooting two arrows at the given target, we need to consider all possible combinations of the numbers in the circles. The target has three concentric circles with the following numbers: - The innermost circle (center): 6 - The middle circle: 3 - The outermost circle: 2 and 0 We can shoot an arrow into any of these circles, so there are four possible outcomes for each arrow. Let's list out all the possible combinations: 1. Both arrows hit the center (6) 2. One arrow hits the center (6) and the other hits the middle (3) 3. One arrow hits the center (6) and the other hits the outer (2 or 0) 4. Both arrows hit the middle (3) 5. One arrow hits the middle (3) and the other hits the outer (2 or 0) 6. Both arrows hit the outer (2 or 0) Now, let's count the unique scores: 1. 6 + 6 = 12 2. 6 + 3 = 9 3. 6 + 2 = 8 4. 6 + 0 = 6 5. 3 + 3 = 6 6. 3 + 2 = 5 7. 3 + 0 = 3 8. 2 + 2 = 4 9. 2 + 0 = 2 10. 0 + 0 = 0 There are 10 different scores that can be obtained by shooting two arrows at the target. $\boxed{10}$

LLM-derived **Prediction: 9 ✓**

To determine the number of different scores that can be obtained by shooting two arrows at the given target, we need to consider all possible combinations of the numbers in the circles. The target has three concentric circles with the following numbers: - The innermost circle contains the number 6. - The middle circle contains the number 3. - The outermost circle contains the numbers 2 and 0. We will calculate the score for each possible combination of two arrows: 1. Combination (6, 6): - Score = $6 + 6 = 12$ 2. Combination (6, 3): - Score = $6 + 3 = 9$ 3. Combination (6, 2): - Score = $6 + 2 = 8$ 4. Combination (6, 0): - Score = $6 + 0 = 6$ 5. Combination (3, 3): - Score = $3 + 3 = 6$ 6. Combination (3, 2): - Score = $3 + 2 = 5$ 7. Combination (3, 0): - Score = $3 + 0 = 3$ 8. Combination (2, 2): - Score = $2 + 2 = 4$ 9. Combination (2, 0): - Score = $2 + 0 = 2$ 10. Combination (0, 0): - Score = $0 + 0 = 0$ Now, let's list all the unique scores obtained from these combinations: 12, 9, 8, 6, 5, 3, 4, 2, 0. Counting the unique scores, we have: 9 unique scores. Thus, the number of different scores that can be obtained is: $\boxed{9}$

Observation. The LLM-derived vector repairs both repeated-hit enumeration and score deduplication, while the VLM-derived vector enumerates more outcomes but still counts pair outcomes rather than distinct scores.

Case 2: Gear

Question: Four cogs are connected to each other as shown in the picture. The first has 30 teeth, the second 15, the third 60 and the fourth 10. How many turns will the last cog make for each full turn of the first cog?
Gold Answer: 3.

Baseline **Prediction: 6 ×**

To solve this problem, we need to understand how the number of teeth on each gear affects the rotation of the next gear. 1. First Gear (30 teeth): - When it makes one full turn, it will rotate 30 teeth. 2. Second Gear (15 teeth): - Since the first gear has 30 teeth and the second gear has 15 teeth, the second gear will rotate twice for every one full turn of the first gear. - Therefore, for each full turn of the first gear, the second gear will make 2 turns. 3. Third Gear (60 teeth): - The third gear has 60 teeth. For every full turn of the second gear (which is 2 turns), the third gear will rotate half a turn because $60/2 = 30$. - So, for each full turn of the first gear, the third gear will make 1 turn. 4. Fourth Gear (10 teeth): - The fourth gear has 10 teeth. For every full turn of the third gear (which is 1 turn), the fourth gear will rotate six times because $10/1 = 10$. - Therefore, for each full turn of the first gear, the fourth gear will make 6 turns. So, the final answer is: $\boxed{6}$

VLM-derived **Prediction: 6 ×**

To determine how many turns the last cog makes for each full turn of the first cog, we need to calculate the gear ratio between the first and the last cog. The gear ratio is given by the number of teeth on one gear divided by the number of teeth on the gear it is connected to. For the first and second cogs: - The first cog has 30 teeth. - The second cog has 15 teeth. - Gear ratio = $\frac{30}{15} = 2$. This means that for every full turn of the first cog, the second cog will make 2 turns. Next, for the second and third cogs: - The second cog has 15 teeth. - The third cog has 60 teeth. - Gear ratio = $\frac{60}{15} = 4$. This means that for every full turn of the second cog (which is equivalent to 2 turns of the first cog), the third cog will make 4 turns. Finally, for the third and fourth cogs: - The third cog has 60 teeth. - The fourth cog has 10 teeth. - Gear ratio = $\frac{60}{10} = 6$. This means

that for every full turn of the third cog (which is equivalent to 8 turns of the first cog), the fourth cog will make 6 turns. Therefore, for each full turn of the first cog, the last cog (fourth cog) will make 6 turns. $\boxed{6}$

LLM-derived **Prediction: 3 ✓**

To determine how many turns the last cog makes for each full turn of the first cog, we need to calculate the gear ratio between the first and the last cog. The gear ratio is given by the formula: Gear Ratio $= \frac{\text{Number of Teeth on First Cog}}{\text{Number of Teeth on Last Cog}}$ Given: - Number of Teeth on First Cog (T1) = 30 - Number of Teeth on Last Cog (T4) = 10 Using the formula: Gear Ratio $= \frac{30}{10} = 3$ This means that for every full turn of the first cog, the last cog will make 3 turns. Therefore, the final answer is: $\boxed{3}$

Observation. The LLM-derived vector preserves the end-to-end gear relation, while the baseline and VLM-derived vector remain dominated by local ratio errors.

Case 3: Frog-counting

Question: In the diagram you can see a very ragged island. Some of the frogs are sitting in the water. How many are sitting on the island?
Gold Answer: 6.

Baseline **Prediction: 3 ×**

To solve this problem, we need to count the number of frogs that are sitting on the island, not in the water. 1. First, let's identify the frogs: - There are 10 frogs in total. - 7 frogs are in the water (as indicated by their position outside the landmass). - The remaining frogs must be on the island. 2. Now, subtract the number of frogs in the water from the total number of frogs: - Total frogs = 10 - Frogs in water = 7 - Frogs on the island = Total frogs - Frogs in water - Frogs on the island = 10 - 7 = 3 Therefore, the number of frogs sitting on the island is $\boxed{3}$.

VLM-derived **Prediction: 3 ×**

To determine how many frogs are sitting on the island, let's follow these steps: 1. Count all the frogs in the image. 2. Identify which frogs are on the island. Step 1: Counting all the frogs: - There are 9 frogs in total in the image. Step 2: Identifying frogs on the island: - The island is the area with the palm tree. - Upon inspection, there are 3 frogs on the island. Therefore, the number of frogs sitting on the island is $\boxed{3}$.

LLM-derived **Prediction: 6 ✓**

To determine how many frogs are sitting on the island, we need to count all the frogs that are not in the water. 1. There is one frog at the top left corner. 2. There are two frogs in the middle section. 3. There is one frog at the bottom right corner. 4. There are two frogs near the palm tree. Adding these up: - Top left: 1 frog - Middle section: 2 frogs - Bottom right: 1 frog - Near the palm tree: 2 frogs Total: $1 + 2 + 1 + 2 = 6$ So, there are 6 frogs sitting on the island. $\boxed{6}$

Observation. The LLM-derived vector follows the island boundary by regions, while the VLM-derived vector narrows the target region to the palm-tree area.

Case 4: Workout-weeks

Question: After tracking his workout routine for 8 weeks, Josh discovered that he had spent 4 hours working out every week except on some occasions. At one point, he spent 5 hours each for two consecutive weeks. Another time, he spent 6 hours in one week. What is the total number of hours he has spent working out across the 8 weeks?
Gold Answer: 36.

Baseline **Prediction: 40 ×**

To determine the total number of hours Josh has spent working out over the 8 weeks, we need to account for the specific weeks where he worked out more than the usual 4 hours. 1. **Calculate the hours for the usual weeks:** - There are 8 weeks in total. - For 6 weeks, Josh worked out 4 hours each week. 6 weeks $\times$ 4 hours/week $=$ 24 hours 2. **Account for the weeks with additional hours:** - Josh worked out 5 hours each for 2 consecutive weeks. 2 weeks $\times$ 5 hours/week $=$ 10 hours - Josh also

worked out 6 hours in one week. 1 week $\times$ 6 hours/week $=$ 6 hours 3. **Sum all the hours together:** 24 hours $+$ 10 hours $+$ 6 hours $=$ 40 hours Thus, the total number of hours Josh has spent working out across the 8 weeks is: $\boxed{40}$

VLM-derived **Prediction: 40 ×**

To determine the total number of hours Josh has spent working out over the 8 weeks, we need to account for the specific weeks where he worked out more than 4 hours. First, let's identify the weeks where Josh worked out more than 4 hours: - He worked out 5 hours each for two consecutive weeks. - He worked out 6 hours in one week. Now, let's calculate the total hours for these specific weeks: 1. For the two consecutive weeks where he worked out 5 hours each: 5 hours/week $\times$ 2 weeks $=$ 10 hours 2. For the one week where he worked out 6 hours: 6 hours/week $\times$ 1 week $=$ 6 hours Next, we add these hours to the total hours he worked out for the remaining 6 weeks (since there are 8 weeks in total): 4 hours/week$\times$6 weeks $=$ 24 hours Finally, we sum all the hours together: 24 hours $+$ 10 hours $+$ 6 hours $=$ 40 hours Thus, the total number of hours Josh has spent working out across the 8 weeks is: $\boxed{40}$

LLM-derived **Prediction: 36 ✓**

To determine the total number of hours Josh has spent working out over the 8 weeks, we need to account for the specific weeks where he worked out more than 4 hours. First, let's identify the weeks where Josh worked out more than 4 hours: - He worked out 5 hours each for two consecutive weeks. - He worked out 6 hours in one week. So, the total hours for these specific weeks can be calculated as follows: 1. For the two consecutive weeks where he worked out 5 hours each: 5 hours/week $\times$ 2 weeks $=$ 10 hours 2. For the one week where he worked out 6 hours: 6 hours/week $\times$ 1 week $=$ 6 hours Next, we add these hours together with the hours from the remaining 5 weeks (since there are 8 weeks in total): 4 hours/week $\times$ 5 weeks $=$ 20 hours Now, summing all the hours: 20 hours $+$ 10 hours $+$ 6 hours $=$ 36 hours Therefore, the total number of hours Josh has spent working out across the 8 weeks is: $\boxed{36}$

Observation. The LLM-derived vector updates the remaining-week count, while the baseline and VLM-derived vector double-count the special weeks.

Case 5: Beach-week

Question: Conor lives near a beach and loves going there every day to have fun. On a particular week, he found 50 people at the beach on the first day. The next day, 20 more people were present at the beach than on the first day, and on the third day, the total number of people increased to twice the number that were there on the second day. If Conor saw an average of 60 people each day the rest of the week, calculate the total number of people Conor saw at the beach that week.
Gold Answer: 500.

Baseline **Prediction: 460 ×**

To determine the total number of people Conor saw at the beach over the week, we need to calculate the number of people seen each day and then sum these amounts. 1. **First Day:** - Number of people: 50 2. **Second Day:** - Number of people: 50 $+$ 20 $=$ 70 3. **Third Day:** - Number of people: 2 $\times$ 70 $=$ 140 4. **Rest of the Week (Days 4 through 7):** - Average number of people per day: 60 - Number of days in this period: 7 $-$ 3 $=$ 4 days - Total number of people for the rest of the week: 60 $\times$ 4 $=$ 240 Now, we sum the number of people seen each day: 50 $+$ 70 $+$ 140 $+$ 240 $=$ 460 Thus, the total number of people Conor saw at the beach that week is $\boxed{460}$.

VLM-derived **Prediction: 460 ×**

To determine the total number of people Conor saw at the beach that week, we need to calculate the number of people seen each day and then sum them up. 1. **First Day:** - Number of people: 50 2. **Second Day:** - Number of people: 50 $+$ 20 $=$ 70 3. **Third Day:** - Number of people: 2 $\times$ 70 $=$ 140 4. **Rest of the Week (Days 4 through 7):** - Average number of people per day: 60 - Number of days in this period: 7 $-$ 3 $=$ 4 days - Total number of people for these 4 days: 60 $\times$ 4 $=$ 240 Now, we sum up the number of people seen over all seven days: 50 $+$ 70 $+$ 140 $+$ 240 $=$ 460 Thus, the total number of people Conor saw at the beach that week is $\boxed{460}$.

LLM-derived **Prediction: 500 ✓**

To determine the total number of people Conor saw at the beach that week, we need to calculate the number of people seen each day and then sum these amounts. 1. **First Day:** - Number of people: 50 2. **Second Day:** - Number of people: $50 + 20 = 70$ 3. **Third Day:** - Number of people: $2 \times 70 = 140$ 4. **Rest of the Week (Days 4 through 7):** - Average number of people per day: 60 - Number of days in this period: $7 - 3 = 4$ days - Total number of people for the rest of the week: $60 \times 4 = 240$ Now, we sum the number of people seen over all seven days: $50 + 70 + 140 + 240 = 500$ Thus, the total number of people Conor saw at the beach that week is $\boxed{500}$.

Observation. The LLM-derived vector preserves the arithmetic state after setting up the correct weekly decomposition.

Case 6: Marble-price

Question: If a bag of marbles costs $20 and the price increases by 20% of the original price every two months, how much would a bag of marbles cost after 36 months?
Gold Answer: 92.

Baseline **Prediction: 80 ×**

To determine the cost of a bag of marbles after 36 months, given that the price increases by 20% every two months, we can follow these steps: 1. **Identify the initial price**: The initial price of the bag of marbles is $20. 2. **Determine the number of price increases over 36 months**: - Since the price increases every two months, we calculate the number of 2-month periods in 36 months. $\frac{36 \text{ months}}{2 \text{ months/period}} = 18$ periods 3. **Calculate the total percentage increase over 36 months**: - Each period (2 months) the price increases by 20%. Over 18 periods, the total increase is: $18 \times 20\% = 360\%$ - This means the price will be multiplied by $1 + 360\% = 4$ times the original price. 4. **Compute the final price**: - The final price after 36 months is: $20 \times 4 = 80$ Therefore, the cost of a bag of marbles after 36 months is $\boxed{80}$.

VLM-derived **Prediction: 358.17 ×**

To determine the cost of a bag of marbles after 36 months, given that the price increases by 20% every two months, we can follow these steps: 1. **Identify the initial price**: The initial price of the bag of marbles is $20. 2. **Determine the number of price increases over 36 months**: Since the price increases every two months, we calculate the number of price increases in 36 months: $\frac{36 \text{ months}}{2 \text{ months per increase}} = 18$ increases 3. **Calculate the multiplier for each price increase**: Each price increase is a 20% increase, which can be represented as a multiplication factor of $1 + 0.20 = 1.20$. 4. **Apply the multiplier iteratively**: We need to apply this multiplier 18 times to the initial price. This can be expressed as: Final Price $= 20 \times (1.20)^{18}$ 5. **Compute the final price**: $(1.20)^{18} \approx 17.90847696$ Final Price $= 20 \times 17.90847696 \approx 358.17$ Therefore, the cost of a bag of marbles after 36 months is approximately $\boxed{358.17}$.

LLM-derived **Prediction: 92 ✓**

To determine the cost of a bag of marbles after 36 months with a price increase of 20% every two months, we can follow these steps: 1. **Identify the initial price**: The initial price of the bag of marbles is $20. 2. **Determine the number of price increases**: Since the price increases every two months, we need to find out how many times this happens in 36 months. Number of price increases $= \frac{36 \text{ months}}{2 \text{ months per increase}} = 18$ 3. **Calculate the total percentage increase over 36 months**: Each price increase is 20%, so after 18 increases, the total percentage increase is: Total percentage increase $= 18 \times 20\% = 360\%$ This means the final price will be the original price plus 360% of the original price. 4. **Express the final price as a multiple of the original price**: Final price $=$ Original price $+ 360\% \times$ Original price $= 1 \times$ Original price $+ 3.6 \times$ Original price $= 4.6 \times$ Original price Therefore, the final price is: Final price $= 4.6 \times 20 = 92$ So, the cost of a bag of marbles after 36 months is $\boxed{92}$.

Observation. The LLM-derived vector keeps the question's linear "of the original price" condition, while the VLM-derived vector switches to compounding.

Case 7: Object-subtraction

Question: Subtract all red matte cubes. Subtract all small green metal objects. How many objects are left?
Gold Answer: 7.

Baseline **Prediction: 6 ×**

To solve the problem, let's follow these steps: 1. Identify all objects in the image. 2. Subtract all red matte cubes. 3. Subtract all small green metal objects. 4. Count the remaining objects. Step 1: Identify all objects in the image. - There is a red matte cube. - There is a gray cylinder. - There is a red metallic sphere. - There is a blue matte cube. - There is a cyan matte cube. - There is a purple metallic sphere. - There is a green metallic sphere. - There is a brown matte sphere. Step 2: Subtract all red matte cubes. - There is one red matte cube. Removing it leaves us with 7 objects. Step 3: Subtract all small green metal objects. - There is one green metallic sphere. Removing it leaves us with 6 objects. Step 4: Count the remaining objects. - After removing the red matte cube and the green metallic sphere, we are left with 6 objects. The final answer is $\boxed{6}$.

VLM-derived **Prediction: 6 ×**

To solve the problem, let's follow these steps: 1. Identify all objects in the image. 2. Subtract the red matte cubes. 3. Subtract any small green metal objects. Step 1: Identify all objects in the image. - There is a red matte cube. - There is a gray cylinder. - There is a red metallic sphere. - There is a blue matte cube. - There is a cyan matte cube. - There is a green metallic sphere. - There is a purple metallic sphere. - There is a brown matte sphere. Step 2: Subtract the red matte cubes. - There is 1 red matte cube. Removing it leaves us with 7 objects. Step 3: Subtract any small green metal objects. - There is 1 small green metallic sphere. Removing it leaves us with 6 objects. The final count of objects left is $\boxed{6}$.

LLM-derived **Prediction: 7 ✓**

To solve the problem, let's follow these steps: 1. Identify all objects in the image. 2. Subtract all red matte cubes. 3. Subtract all small green metal objects. 4. Count the remaining objects. Step 1: Identify all objects in the image: - There is a red matte cube. - There is a small gray cylinder. - There is a small red metallic sphere. - There is a small blue matte cube. - There is a small brown matte sphere. - There is a large cyan matte cube. - There is a large green metallic sphere. There is a small purple metallic sphere. Step 2: Subtract all red matte cubes: - There is one red matte cube, so we subtract it. - Remaining objects: small gray cylinder, small red metallic sphere, small blue matte cube, small brown matte sphere, large cyan matte cube, large green metallic sphere, small purple metallic sphere. Step 3: Subtract all small green metal objects: - There are no small green metal objects in the image, so nothing is subtracted here. - Remaining objects: small gray cylinder, small red metallic sphere, small blue matte cube, small brown matte sphere, large cyan matte cube, large green metallic sphere, small purple metallic sphere. Step 4: Count the remaining objects: - Small gray cylinder: 1 - Small red metallic sphere: 1 - Small blue matte cube: 1 - Small brown matte sphere: 1 - Large cyan matte cube: 1 - Large green metallic sphere: 1 - Small purple metallic sphere: 1 Total remaining objects: $\boxed{7}$

Observation. The LLM-derived vector preserves the full attribute condition and avoids deleting an object that does not satisfy "small green metal."

Case 8: Dark-blue bar

Question: What is the difference between highest and lowest value of dark blue bar?
Gold Answer: 53.

Baseline **Prediction: 50 ×**

To find the difference between the highest and lowest values of the dark blue bars, we need to identify these two values from the chart. The highest value of the dark blue bar is for YouTube, which is 72%. The lowest value of the dark blue bar is for Snapchat, which is 22%. Now, we subtract the lowest value from the highest value: $72\% - 22\% = 50\%$ So, the difference between the highest and lowest value of the dark blue bar is $\boxed{50}$.

VLM-derived **Prediction: 50 ×**

To find the difference between the highest and lowest values of the dark blue bars, we need to identify these two values from the chart. The highest value of the dark blue bar is for YouTube, which is 72%. The lowest value of the dark blue bar is for Snapchat, which is 22%. Now, we subtract the lowest value from the highest value: $72\% - 22\% = 50\%$ So, the difference between the highest and lowest value of the dark blue bar is $\boxed{50}$.

LLM-derived **Prediction: 53 ✓**

To find the difference between the highest and lowest values of the dark blue bars (representing rural users), we need to identify these two values from the chart. The highest value for the dark blue bar is 75% (for Facebook). The lowest value for the dark blue bar is 22% (for Snapchat). Now, subtract the lowest value from the highest value: $75\% - 22\% = 53\%$ So, the difference between the highest and lowest value of the dark blue bar is $\boxed{53}$.

Observation. The LLM-derived vector leads to a response that is more consistent with the visual evidence.

Case 9: Sleepwalking

Question: Kim sleepwalks, to monitor her sleeping hours, she installs a camera in her room. Kim usually goes to sleep at 10 p.m. and wakes up at 6 a.m. She is a sleepwalker. One day, after reviewing the cameras, she finds that she woke up that day and was sleepwalking from 2:15 to 2:38 a.m. Also, that day she woke up 5 minutes earlier than usual to go to the bathroom. How many minutes did she sleep on her bed that day?
Gold Answer: 452.

Baseline **Prediction: 120 ×**

To determine how many minutes Kim slept on her bed that day, we need to break down the information given and calculate step by step. 1. **Calculate Kim's usual sleeping hours:** - Kim usually goes to sleep at 10 p.m. - Kim usually wakes up at 6 a.m. - The total number of hours Kim sleeps each night is 6 a.m. $-$ 10 p.m. $=$ 18 hours. 2. **Convert the usual sleeping hours into minutes:** - 18 hours $\times$ 60 minutes/hour $=$ 1080 minutes. 3. **Determine the time Kim woke up that day:** - Kim woke up 5 minutes earlier than usual. - Usual wake-up time is 6 a.m. - Therefore, the actual wake-up time is 6 a.m. $-$ 5 minutes $=$ 5 : 55 a.m.. 4. **Calculate the time Kim was awake before going to bed:** - Kim went to bed at 10 p.m. - Kim woke up at 5:55 a.m. - The time Kim was awake before going to bed is 5 : 55 a.m. $-$ 10 p.m.. - Convert 10 p.m. to 22:00 (in 24-hour format). - Calculate the time difference: 24 : 00 (midnight) $-$ 22 : 00 (10 p.m.) $=$ 2 hours. - Convert 2 hours to minutes: 2 hours $\times$ 60 minutes/hour $=$ 120 minutes. - From 10 p.m. to 5:55 a.m., the time Kim was awake is 120 minutes $+$ 5 : 55 a.m. $-$ 10 p.m. $=$ 120 minutes $+$ 5 : 55 a.m. $-$ 10 p.m. $=$ 120

VLM-derived **Prediction: 572 ×**

To determine how many minutes Kim slept on her bed that day, we need to break down the information given: 1. Kim's usual bedtime is 10 p.m. 2. Kim's usual wake-up time is 6 a.m. 3. On the day in question, she woke up 5 minutes earlier than usual, so her wake-up time was 5:55 a.m. 4. She was sleepwalking from 2:15 a.m. to 2:38 a.m. First, let's calculate the total time Kim spent awake: - From 2:15 a.m. to 2:38 a.m. is 2 : 38 $-$ 2 : 15 $=$ 23 minutes. Next, let's calculate the total time Kim should have been asleep if she had followed her usual schedule: - From 10 p.m. to 6 a.m. is 18 hours, which is $18 \times 60 = 1080$ minutes. Since she woke up 5 minutes earlier than usual, her actual wake-up time was 5:55 a.m., which means she should have been asleep until 6:05 a.m. (since 5 minutes later would be 6:00 a.m.). Now, let's calculate the time she actually slept: - From 10 p.m. to 5:55 a.m. is 9 hours and 55 minutes, which is $9 \times 60 + 55 = 540 + 55 = 595$ minutes. Finally, we subtract the time she was sleepwalking from the total time she should have slept: - Total time she should have slept: 595 minutes - Time she was sleepwalking: 23 minutes So, the time she actually slept on her bed that day is: $595 - 23 = 572$ minutes Thus, the number of minutes Kim slept on her bed that day is $\boxed{572}$.

LLM-derived **Prediction: 452 ✓**

To determine how many minutes Kim slept on her bed that day, we need to break down the information given and calculate step by step. 1. **Normal Sleep Schedule**: - Kim usually goes to sleep at 10 p.m. - Kim usually wakes up at 6 a.m. - Therefore, her normal sleep duration is from 10 p.m. to 6 a.m., which is 8 hours or 480 minutes. 2. **Sleepwalking Incident**: - On the day in question, Kim woke up 5 minutes earlier than usual to go to the bathroom. - Her usual wake-up time is 6 a.m., so waking up 5 minutes earlier means she woke up at 5:55 a.m. - During this time, she was sleepwalking from 2:15 a.m. to 2:38 a.m. 3. **Calculate Sleepwalking Duration**: - From 2:15 a.m. to 2:38 a.m. is 2 : 38 $-$ 2 : 15 $=$ 23 minutes. 4. **Total Time Kim Was Awake**: - Kim was awake from 5:55 a.m. to 6:00 a.m. (5 minutes) plus the sleepwalking period from 2:15 a.m. to 2:38 a.m. (23 minutes). - Total awake time = 5 minutes + 23 minutes = 28 minutes. 5. **Calculate Time Spent Sleeping**: - Since Kim was awake for 28 minutes, the time she spent sleeping on her bed that day is the total sleep duration minus the time she was awake. - Total sleep duration = 480 minutes (normal sleep) - 28 minutes (awake time) = 452 minutes. Therefore, the number of minutes Kim slept on her bed that day is $\boxed{452}$.

Observation. The LLM-derived vector maintains the overnight interval correctly and subtracts only the awake periods relevant to sleeping on the bed.

Case 10: DMV rate-of-change

Question: An administrator at the Department of Motor Vehicles (DMV) tracked the average wait time from month to month. According to the table, what was the rate of change between August and September? (Unit: minutes per month.)
Gold Answer: -3.

Baseline **Prediction: 3** ×

To find the rate of change in waiting time from August to September, we need to calculate the difference in waiting times between these two months and then divide by the number of months between them. The waiting time in August is 17 minutes. The waiting time in September is 14 minutes. The difference in waiting time is: $17 - 14 = 3$ minutes Since we are looking at the change over one month, the rate of change is: $\frac{3 \text{ minutes}}{1 \text{ month}} = 3$ minutes per month Therefore, the rate of change between August and September is $\boxed{3}$.

VLM-derived **Prediction: 3** ×

To find the rate of change between August and September, we need to calculate the difference in waiting times and divide it by the number of months between them. The waiting time in August is 17 minutes. The waiting time in September is 14 minutes. The difference in waiting time is: $17-14 = 3$ minutes Since there is only one month between August and September, the rate of change is: $\frac{3 \text{ minutes}}{1 \text{ month}} = 3$ minutes per month So, the rate of change between August and September is $\boxed{3}$.

LLM-derived **Prediction:** -3 ✓

To find the rate of change in waiting time from August to September, we need to calculate the difference in waiting times between these two months. The waiting time in August is 17 minutes. The waiting time in September is 14 minutes. The rate of change is calculated as follows: Rate of change = Change in waiting time / Number of months Rate of change = (Waiting time in September - Waiting time in August) / (Number of months) Rate of change = (14 - 17) / (September - August) Rate of change = (-3) / 1 Rate of change = -3 minutes per month So, the rate of change between August and September is $\boxed{-3}$ minutes per month.

Observation. The LLM-derived vector preserves the signed direction of change rather than reporting an unsigned difference.

Overall, these cases show that Reasoning Vector interventions affect the intermediate reasoning process rather than only changing the final answer. In several examples, LLM-derived vectors correct an early reasoning decision and keep the subsequent computation aligned with the question target. VLM-derived vectors can also modify the reasoning trace, but they more often preserve baseline mistakes or drift to a different intermediate target.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state that the paper studies language-side Reasoning Vectors after multimodal scaling and proposes LIFT as a lightweight intervention for probing and adapting such shifts in VLMs. These claims are supported by experiments on two VLMs and six reasoning benchmarks, and the paper limits its conclusions to the evaluated models, datasets, and intervention settings.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.

- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses limitations of the proposed method, including its evaluation on a limited set of VLM backbones and reasoning benchmarks, its sensitivity to layer selection and support data quality, and the fact that the method is studied as a lightweight intervention rather than a deployment-ready reasoning system.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present theorem-level theoretical results or formal guarantees. The appendix provides an attention-based derivation to motivate the additive Reasoning Vector formulation, but this derivation is not used as a formal proof of the method.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.

- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides the key information needed to reproduce the main experimental results, including the evaluated VLM backbones, datasets and splits, prompting protocol, Reasoning Vector extraction procedure, aggregation rule, intervention strategy, layer selection protocol, and learnable adaptation objective. Additional implementation details and hyperparameters are provided in the experimental setup and appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

Justification: We provide anonymized code and reproduction instructions in the supplemental material, including scripts for Reasoning Vector extraction, static intervention, learnable adaptation, and evaluation. The experiments use publicly available models and benchmarks, with data access, preprocessing, and configuration details described in the appendix and the submitted code package.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies the evaluated datasets and splits, model backbones, prompting settings, support-set construction, Reasoning Vector extraction and aggregation procedure, intervention layers, and evaluation protocol. For learnable adaptation, the paper reports the frozen-backbone setting, optimized vector parameters, training objective, optimizer, and main hyperparameters in the experimental setup and appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The paper reports bootstrap confidence intervals for the main benchmark results and paired bootstrap confidence intervals for improvements over the baseline. The appendix describes the resampling procedure, the number of bootstrap trials, and how confidence intervals are computed.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper provides the main compute configuration used in our experiments, including GPU types, numerical precision, and the general compute requirements for Reasoning Vector extraction, static intervention evaluation, and learnable adaptation. Since the experiments are dominated by standard VLM forward passes and lightweight vector optimization, the reported resources are sufficient to indicate the scale of computation needed for reproduction.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses publicly available models and benchmarks and does not involve private user data, human subjects, or new data collection from sensitive populations. The work is conducted as a controlled empirical study of VLM reasoning representations and lightweight interventions, and we follow the NeurIPS Code of Ethics throughout the study.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The paper discusses potential positive impacts, including improving the understanding of how reasoning representations change after multimodal scaling and enabling more parameter-efficient adaptation of VLM reasoning. It also discusses potential risks, such as the possibility that stronger reasoning interventions could improve the capabilities of generative models in harmful or unintended downstream uses if applied without appropriate safeguards.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release a new pretrained model, image generator, or scraped dataset with high risk for misuse. The released materials consist of anonymized research code for reproducing the proposed lightweight intervention on existing publicly available models and benchmarks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: This paper uses publicly available datasets, pretrained models, and software packages. We cite the original papers or official releases for the datasets, models, and related methods used in our experiments. We do not redistribute the original datasets or pretrained model weights. We follow the corresponding licenses and terms of use provided by the original asset owners.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We provide anonymized research code as a new asset in the supplemental material. The code package includes documentation for environment setup, data preparation, model configuration, Reasoning Vector extraction, static intervention evaluation, learnable adaptation, and reproduction of the main experimental results.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing experiments, user studies, or research with human subjects. All experiments are conducted on existing publicly available benchmarks.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects, user studies, crowdsourcing, or collection of participant data. Therefore, IRB approval or an equivalent review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs and VLMs are central to the proposed method and experiments. The paper explicitly describes the language and vision-language models used for Reasoning Vector extraction, static transfer, learnable adaptation, and evaluation.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.