Can LLMs Reason Abstractly Over Math Word Problems Without CoT? Disentangling Abstract Formulation From Arithmetic Computation

Anonymous authors
Paper under double-blind review

Abstract

Final-answer-based metrics are commonly used for evaluating large language models (LLMs) on math word problems, often taken as proxies for reasoning ability. However, such metrics conflate two distinct sub-skills: **abstract formulation** (capturing mathematical relationships using expressions) and **arithmetic computation** (executing the calculations). Through a disentangled evaluation on GSM8K and SVAMP, we find that the final-answer accuracy of Llama-3 and Qwen2.5 (1B-32B) without CoT is overwhelmingly bottlenecked by the arithmetic computation step and not by the abstract formulation step. Contrary to the common belief, we show that CoT primarily aids in computation, with limited impact on abstract formulation. Mechanistically, we show that these two skills are composed conjunctively even in a single forward pass without any reasoning steps via an **abstract-then-compute** mechanism: models first capture problem abstractions, then handle computation. Causal patching confirms these abstractions are present, transferable, composable, and precede computation. These behavioural and mechanistic findings highlight the need for disentangled evaluation to accurately assess LLM reasoning and to guide future improvements.[1]

1 Introduction

Large language models (LLMs) have demonstrated impressive progress on various math problem datasets (Cobbe et al., 2021; Hendrycks et al., 2021b; Patel et al., 2021), often leveraging Chain-of-Thought (CoT) prompting (Wei et al., 2022). Despite the availability of step-by-step reasoning chains, standard evaluation predominantly relies on final-answer accuracy (comparing the model's final numerical output against a gold answer), which reduces model performance to a single metric (Liu et al., 2024; Opedal et al., 2024). This reduction limits the possible insights when diagnosing LLMs' reasoning abilities, especially in zero-shot scenarios without CoT. When an LLM fails to produce the correct answer, is it due to "reasoning deficits", or could it be a calculation error?

To investigate this, we propose a disentangled evaluation framework that separately measures two core skills of mathematical problem-solving (See Figure 1): (1) **abstract formulation** (hereafter, abstraction) — the ability to identify relevant quantities and translate the natural language problem into its underlying mathematical relationships (e.g., $36 + 47$ or $x + y$ in Figure 1); and (2) **arithmetic computation** (hereafter, computation) — the capacity to calculate the final answer from that expression (e.g., evaluate $36 + 47$ to 83).

Using this disentangled evaluation on GSM8K (Cobbe et al., 2021) and SVAMP (Patel et al., 2021) with Llama-3 and Qwen-2.5 models (1B-32B), we find that even without CoT: (i) models surprisingly perform better at abstraction than computation, despite the former's perceived conceptual complexity. (ii) if deriving the final answer in math word problems depends on these two skills conjunctively, final-answer accuracy alone may give a misleading picture of models' reasoning abilities in math word problems. Moreover, we show that CoT

[1]Code and data will be made publicly available upon acceptance.

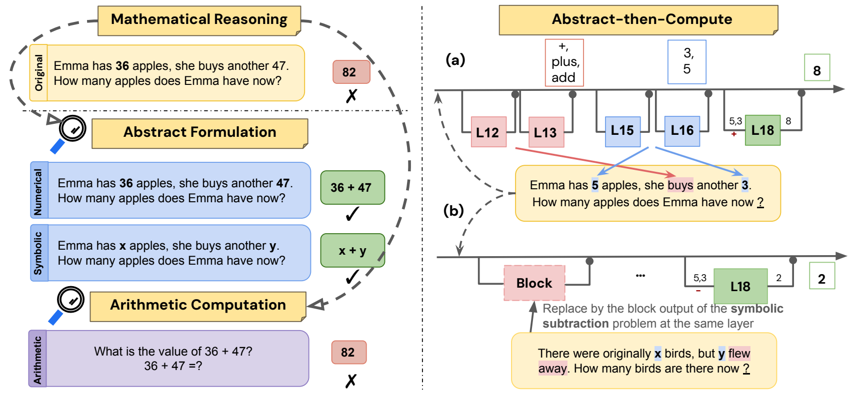


Figure 1: **Left (Disentangled evaluation framework):** Final-answer accuracy obscures reasoning ability due to conflating abstract formulation and arithmetic computation. **Right (Abstract-then-Compute Mechanism in Llama-3 8B):** (a) Residual stream at the last token position shows that models first capture problem abstraction (L13-14), followed by computation (L18). (b) Same as (a), but one critical layer output is patched with a different symbolic abstraction (e.g., $x - y$), causally changing the computation from $5 + 3 = 8$ to $5 - 3 = 2$.

primarily improves computation, with limited gains in abstraction, further demonstrating the value of disentangled evaluation.

While these behavioural findings suggest that models can formulate abstractions without explicit CoT when separately prompted, it remains unclear whether abstraction and computation are composed conjunctively when deriving the final answer during single-pass inference. To explore this, we move beyond outcome-based evaluation, and conduct mechanistic interpretability analyses. Using logit attribution and activation patching, we identify a consistent and sequential **abstract-then-compute** mechanism (see Figure 1a). Moreover, cross-prompt patching provides evidence that models do form abstractions internally independent of the surface form (numerical or symbolic, see Figure 1b): when these symbolic abstractions (e.g. $x - y$) are transferred into a different problem, they are utilized and composed with the subsequent computation stages, altering the final answer.

Contributions: (i) Through disentangled evaluation, we show that without CoT, models exhibit stronger reasoning ability than final-answer accuracy suggests, and that CoT primarily aids calculation. (ii) Using mechanistic interpretability, we uncover an abstract-then-compute mechanism in a single-pass generation, where abstractions are transferrable across problem variants. Collectively, our findings suggest an alternative narrative: poor final-answer accuracy without CoT (Wei et al., 2022; Sprague et al., 2025), or performance declines on problem variants (Zhang et al., 2024a; Shi et al., 2023; Mirzadeh et al., 2025), can stem from arithmetic errors rather than reasoning deficits.

2 Related work

Mathematical reasoning evaluation Existing math problem-solving benchmarks spans elementary word problems (Cobbe et al., 2021; Patel et al., 2021; Amini et al., 2019; Miao et al., 2020; Ling et al., 2017; Koncel-Kedziorski et al., 2016; Shi et al., 2015) to higher levels (Hendrycks et al., 2021b;a; Zhong et al., 2024; Zhang et al., 2023; He et al., 2024). Early datasets paired expressions with answers, but evaluation largely focused on final-answer-based metrics (Patel et al., 2021; Shi et al., 2015). With the rise of LLMs and CoT prompting (Wei et al., 2022), rationale-based formats became common (Hendrycks et al., 2021b; Cobbe et al., 2021), yet standard evaluations still predominantly use final-answer metrics, and occasionally code execution from rationales (Mishra et al., 2022; Gao et al., 2023). In contrast, we move beyond this final-answer-centric paradigm, by decomposing problem-solving into abstract formulation and arithmetic computation, inspired by the cognitive theories (Opedal et al., 2024).

Memorization vs. generalization Variants of math word problems with perturbations were introduced to test generalization beyond memorization (Zhang et al., 2024a; Ye et al., 2025; Gao et al., 2023; Shi et al., 2023; Li et al., 2024; Mirzadeh et al., 2025). While performance drops are often interpreted as reasoning failures, our results suggest they may instead stem mainly from arithmetic errors, pointing to a different improvement strategy.

Mechanistic interpretability Mechanistic interpretability methods, such as logit attribution (nostalgebraist, 2020; Belrose et al., 2023) and causal patching (Goldowsky-Dill et al., 2023; Wang et al., 2023; Meng et al., 2022; Zhang & Nanda, 2023; Merullo et al., 2024; Cheng et al., 2025), have been used to trace model computations. Prior work on math reasoning largely focuses on the mechanisms behind arithmetic computations (Nikankin et al., 2025; Zhang et al., 2024b), while recent work on word problems use probing classifiers to track explicit variable reasoning (Ye et al., 2025). In contrast, we examine implicit reasoning within a single forward pass to uncover abstraction beyond computation.

3 Dataset and experimental design

Task and dataset We study math word problems using GSM-8K (Cobbe et al., 2021) and SVAMP (Patel et al., 2021). GSM-8K spans 2–8 steps without distractors (See Figure 6 in Appendix A.1 for statistics), while SVAMP involves single-step reasoning with distractor variants. To evaluate abstract formulation, we create symbolic variants: SVAMP expressions are templated into variable-based forms; GSM-8K symbolic versions from the test set are generated using `gpt-4o-mini` (OpenAI, 2024) via a two-stage *generate-then-validate* to ensure the correctness. See Appendix A.2 and Table 4 for details and examples. For interpretability, we generate 3,600 simple 1–2 step[2] word problems involving basic operations $(+, -, \times, \div)$ from 1,200 diverse LLM-generated templates, covering varied scenarios, verb choices, entities, names and sentence structures. See Appendix B.1 and Table 7 for details and examples.

Models We evaluate instruction-tuned Llama-3 (1B, 3B, 8B) (Grattafiori et al., 2024) and Qwen 2.5 (Yang et al., 2024) (3B, 7B, 14B, 32B) models. Mechanistic interpretability analyses focus on Llama-3 8B, Qwen 2.5 7B, and Qwen 2.5 14B.

Evaluation All experiments use greedy decoding and FP16 precision on RTX 8000/A100L GPUs. Numeric answers are evaluated via normalized Exact Match. Symbolic expressions are evaluated using `gpt-4o-mini` (94% agreement with humans on 120 samples, prompt and details in Appendix A.3) and with `sympy` for numeric expressions. We report standard accuracy. CoT generations are capped at 512 tokens. See Appendix A.3 for details.

4 Disentangled evaluation

We first introduce the disentangled evaluation framework, then present results without CoT in Sec. 4.1, followed by an analysis of CoT's impact in Sec. 4.2.

Framework Suppose a task T can be decomposed into a set of sub-skills $\{s_1, s_2, \ldots, s_n\}$, such that solving T requires executing these skills conjunctively (i.e., $T = s_1 \cap s_2 \cap \cdots \cap s_n$). Disentangled evaluation aims to assess each sub-skill s_i independently via a corresponding subtask t_i, designed to isolate and test that specific skill. Let $\text{Eval}(T)$ denote the evaluation metric on the full task, and $\text{Eval}(t_i)$ the metric for subtask t_i. Measuring $\text{Eval}(t_1), \ldots, \text{Eval}(t_n)$ enables finer-grained attribution of performance, identifying failure of specific skills. In math word problems, let (Q, E, A) be the question, expression and answer triplets, we decompose mathematical problem-solving into **abstract formulation** (translating Q to mathematical relationships E) and **arithmetic computation** (executing the

[2]We focus on 1–2 step problems, as models often fail simple word problems involving multi-step computations in a single forward pass.

Setting	Skills Tested	Question Form and Example	Answer Form and Example
Original	Abstraction +Computation	Numerical: Weng earns $12 for every hour she works. If she worked for 50 minutes, how much did she earn?	Number: 10
Arithmetic Computation	Computation	Numerical: What is the value of $12 \times \left(\frac{50}{60}\right)$? or $12 \times \left(\frac{50}{60}\right) =$?	Number: 10
Numerical Abstraction	Abstraction	Numerical: Weng earns $12 for every hour she works. If she worked for 50 minutes, how much did she earn?	Expression: $12 \times \left(\frac{50}{60}\right)$
Symbolic Abstraction	Abstraction	Symbolic: Weng earns $x for every hour she works. If she worked for y minutes, how much did she earn?	Expression: $x \times \left(\frac{y}{60}\right)$

Table 1: Disentangled evaluation in math word problems with tested skills, varying by question and answer forms. Instructions in Appendix Table 5.

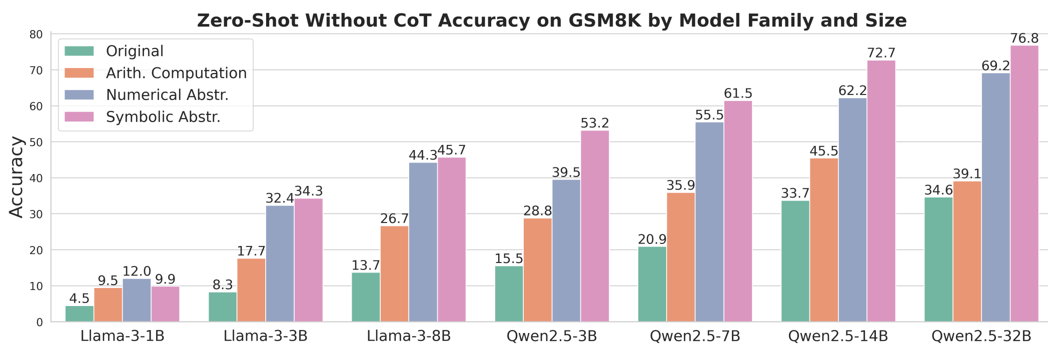


Figure 2: Model zero-shot **without CoT** performance on GSM8K. (i) Models exhibit much better abstraction performance (*Symbolic* and *Numerical*) than in actually computing the expressions (*Arithmetic Computation*). (ii) Final-answer accuracy in the *Original* setting may provide a misleading picture of models' reasoning ability, possibly due to arithmetic limitations.

calculation from E to produce A). Besides the standard *Original* setting (requiring both abstraction and computation), we design three targeted subtasks: *Symbolic Abstraction*, which assess abstraction using symbolic variables; *Numerical Abstraction*, evaluating abstraction with concrete numbers but without computation; and *Arithmetic Computation*, which directly tests execution of fully specified expressions from Q. See Table 1 and Appendix A.3 for details.

4.1 Understanding model failures: reasoning or arithmetic error?

We first apply disentangled evaluation **zero-shot without CoT** across multiple model sizes of Llama-3 and Qwen2.5 families. As shown in Figure 2, the error rates are consistently lower for abstract formulation (both *Numerical* and *Symbolic Abstraction*) compared to arithmetic computation. This suggests that if final-answer accuracy in the *Original* setting depends on both competencies conjunctively, poor performance observed in the *Original* setting could stem from arithmetic computation failures, rather than reasoning deficits. Consequently, this indicates that final-answer accuracy alone from the *Original* setting may substantially mislead a model's underlying reasoning ability. See additional results in Appendix A.4. To assess the reliability and external validity of the symbolic abstraction evaluation, we perform ablations over symbol order and symbol choice in Appendix A.6.

4.2 Disentangling CoT gains

We now apply disentangled evaluation with CoT to disentangle CoT gains (Table 2). We show that CoT yields the largest gains in computation (e.g., +62.8%), confirming its effectiveness in multi-step arithmetic. In contrast, abstraction shows limited improvement (e.g., +6.7% for *Symbolic abstraction* and +17.6% for *Numerical abstraction*), even with extended generation budgets (512 tokens), suggesting CoT is less helpful for abstraction. Gains in the *Original* setting (e.g., +62.8%) likely reflect a mix of benefits from both components and possible data leakage. See additional results in Appendix A.5.

Δ Accuracy	8B	7B	14B	32B	Avg.
Original	64.8	68.5	58.4	59.7	62.8
Arith. Comp.	64.8	60.5	51.2	58.2	58.7
Numerical Abstr.	15.8	21.6	21.6	11.6	17.6
Symbolic Abstr.	11.0	13.2	1.1	1.3	6.7

Table 2: Accuracy difference (%) with and without CoT. Results are shown for Llama 3 (8B) and Qwen2.5 models (7B, 14B, 32B).

Summary: These findings challenge the view that poor final-answer accuracy in math reasoning benchmarks always implies 'poor reasoning'. Instead, our disentangled design reveals that many models do possess a level of abstract formulation capabilities, which are often obscured in standard evaluations due to their limited arithmetic competence. Crucially, while abstraction variants indicate far higher performance than the *Original* setting, models are still not perfect — performance in *Symbolic Abstraction* remains far from 100% (45.7% for Llama-8B, 76.8% for Qwen-32B), but the gap is significantly narrower than previously assumed, calling for more precise definition and evaluation of reasoning.

5 Inside the model: probing abstraction and computation

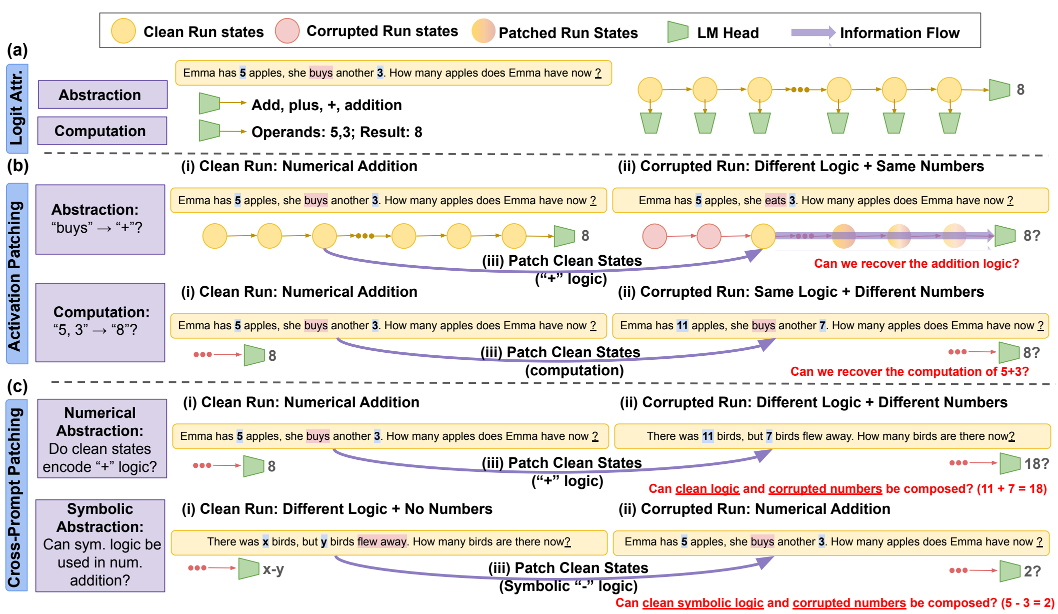


Figure 3: Overview of interpretability methods probing the abstract-then-compute mechanism in simple math problems, focusing on hidden states at the last token position across layers.

To investigate whether abstraction and computation are composed conjunctively when producing a final numerical answer in a single forward pass, we move beyond outcome-based evaluation and apply mechanistic interpretability. We hypothesize an **abstract-then-compute** process: first inferring the abstraction (e.g., '+' from "buys"), then performing the computation (e.g., $5 + 3$). Section 5.1 identifies key layers for each stage; Section 5.2 validate these layers and tests abstraction transferability across forms (symbolic/concrete) and logic.

5.1 Uncovering the abstract-then-compute mechanism in one forward pass

5.1.1 Methods

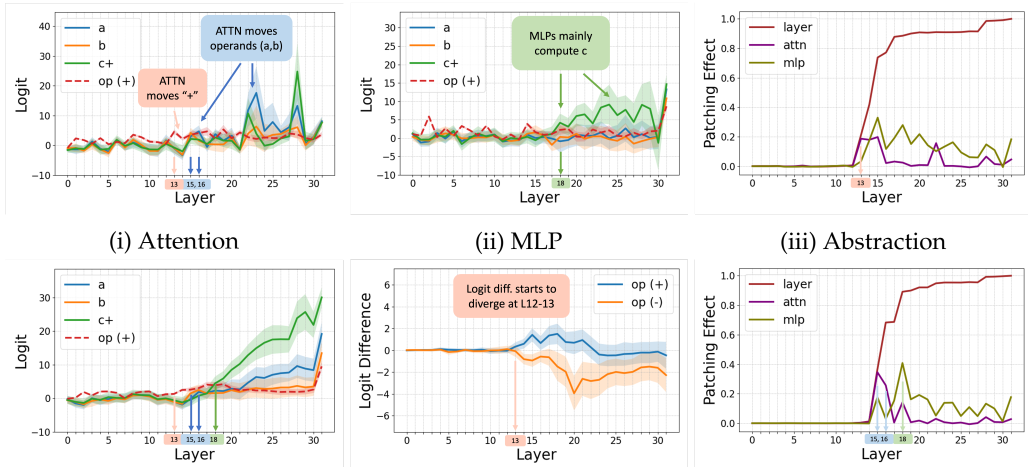
We use logit attribution (nostalgebraist, 2020; Belrose et al., 2023) and activation patching (Ghandeharioun et al., 2024; Zhang & Nanda, 2023; Meng et al., 2022) to probe whether abstraction and computation occur during single-step generation. As summarized in Figure 3, we seek evidence of abstraction and computation.

Logit attribution We use logit attribution to examine specific information (e.g., operator or answer tokens) at each layer (See Figure 3a for illustration). Specifically, we compute *direct logit attribution* (nostalgebraist, 2020; Belrose et al., 2023) of a target token t by projecting hidden states at various points in each layer onto the vocabulary space: $\text{logit}(t) = \langle W_U[t], \text{LN}(h) \rangle$, where h is the hidden state, LN is LayerNorm, and $W_U[t]$ is the unembedding vector. We probe four points within each layer at the last token position: the attention output, MLP output, and the residual stream immediately after merging the attention output (*resid mid*) and after merging the MLP output (*resid final*). As summarized in Figure 3a, we track abstraction via the logits of operator tokens (e.g., "+", "add", "addition") and computation via the logits of operand and answer tokens across layers.

Algorithm 1 Activation Patching

Input: Set Ω of clean and corrupted sample pairs (X_{cl}, X_{cor}), model $\mathcal{M}$ with hidden states $\mathcal{S}$.
Output: Patching effects for $\mathcal{S}$: $E_{\mathcal{S}}$.
for $(X_{cl}^{(i)}, X_{cor}^{(i)}) \in \Omega$ **do**
  $logit_o, A_{cl} \leftarrow \mathcal{M}(X_{cl}^{(i)}, A_{cl})$ # Clean run: get clean logits and store all layer activations A_{cl}
  $logit_c, A_{cor} \leftarrow \mathcal{M}(X_{cor}^{(i)}, A_{cor})$ # Corrupted run: get corrupted logits and store all layer activations A_{cor}
  for $s \in \mathcal{S}$ **do**
    $A'_{cor}(s) \leftarrow A_{cl}(s)$ # Patched run: replace hidden state s in A_{cor} by A_{cl}
    $logit_p \leftarrow \mathcal{M}(X_{cor}^{(i)}, A'_{cor})$ # get patched logits
    $e_s^{(i)} \leftarrow \frac{logit_p - logit_c}{logit_o - logit_c}$ # patching effect
  end for
end for
Return: $E_s \leftarrow \frac{1}{|\Omega|} \sum_{i=1}^{|\Omega|} e_s^{(i)}$

Activation patching To identify components causally responsible for abstraction and computation, we apply activation patching (Algorithm 1, see Figure 3b for visualization)) (Ghandeharioun et al., 2024; Zhang & Nanda, 2023; Meng et al., 2022). To quantify the contribution of each component across layers, this method replaces a single intermediate hidden state in the corrupted forward pass with the corresponding hidden state from the clean run and measures how much this single hidden state injected in corrupted forward pass can restore the prediction of the clean answer. This patching effect per state per layer is a normalized score from 0 (no recovery) to 1 (full recovery to clean performance), with higher indicating more contribution. We patch attention, MLP and final layer outputs across layers at the last position. Formally, we quantify causal impact using the logit difference



(i) Attention (ii) MLP (iii) Abstraction

(iv) Resid Final (v) Resid Mid (vi) Computation

Figure 4: Visualizations of internal computations at last token position in Llama-3 8B for addition math word problems: (i,ii, iv, v) for logit attribution results where a, b are operands and c is the result; (iii, vi) for activation patching results. We label the starting layer of abstraction, operand moving and computation in pink, blue and green, respectively.

between clean $a_{cl}^{(i)}$ and corrupted answers $a_{cor}^{(i)}$ in Eq. 2.

$$\text{LD}_*(i) = \text{logit}_*(a_{\text{cl}}^{(i)}) - \text{logit}_*(a_{\text{cor}}^{(i)}) \tag{1}$$

$$e_s^{(i)} = \frac{\text{LD}_p(i) - \text{LD}_c(i)}{\text{LD}_o(i) - \text{LD}_c(i)} \tag{2}$$

To probe **abstraction** (Figure 3b), we construct minimally different clean/corrupted pairs that vary in their underlying logic (e.g., "buys" for addition vs. "eats" for subtraction) but have the same numbers (e.g., "5,3"). In Figure 3b, the clean input implies $5 + 3 = 8$, while the corrupted input implies $5 - 3 = 2$. We patch individual clean states to the corrupted run to identify critical layers for restoring the addition logic and recovering the clean answer '8'. For **computation** (Figure 3b), we use pairs with the same logic (e.g., addition), but different numbers (e.g., "5,3" vs. "11,7"). Here, we seek to identify layers whose states when patched individually from the clean run to the corrupted run, are critical to perform the clean-run-specific computation with clean operands 5, 3 and output "8".

5.1.2 *Abstract-then-compute hypothesis*

As shown in Figure 4, we observe distinct stages for abstraction and computation, supporting the *abstract-then-compute* hypothesis. Logit attribution reveals that around L13–14, attention begins moving the inferred operator (e.g., '+', plus', add') to the last position (Figure 4i, iv). This coincides with a divergence in logit differences between target operators (+' vs. '−') in addition and subtraction problems (Figure 4v), suggesting that while earlier layers encode generic operator features, problem-specific abstraction emerges here. Subsequently, around L15–16, Figure 4i,iv shows operands transfer to the last position; Following abstraction, the computation phase appears to begin at L18, primarily through MLPs layers (Figure 4ii, iv). Activation patching confirms the distinct stages: abstraction starts at around L13, with rising attention and layer patching effects (Figure 4iii); The rise of attention and layer patching effects in L15,16 in Figure 4vi aligns with our previous observation that operands are being moved to the last position. Finally, the peak patching effect of MLP at L18 highlight their crucial role in calculating the answer. These combined results support our hypothesis that the model follows an *abstract-then-compute* mechanism within a single forward pass. Additional logit attribution and activation patching results for other models and two-operator problems are in Appendix B.2.

5.2 Validation and abstraction transfer with cross-prompt patching

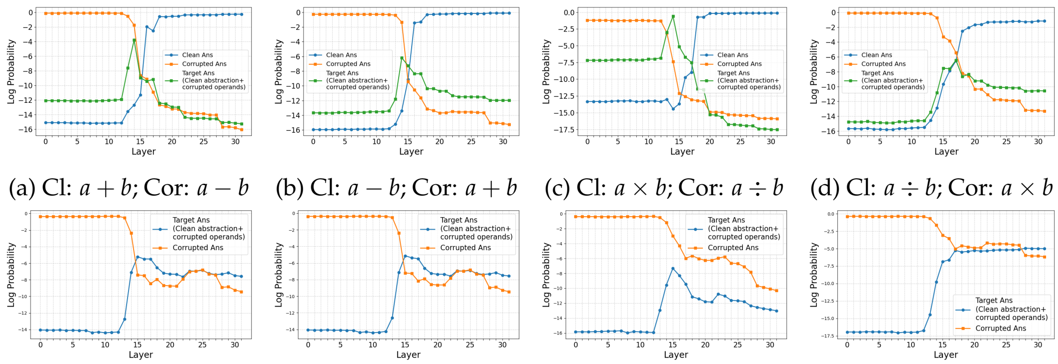
We now validate the causal role of the critical layers for abstraction (L13,14) and computation (L15,16 for operands, L18 for execution). We also investigate if the abstraction representations formed at around L13,14 are transferable across problem forms (symbolic/concrete) and templates, and can be composed with subsequent computation stage.

Method Cross-prompt patching also uses Algorithm 1, but instead of computing patching effects, we track the log-probability of specific tokens across layers in each patched run. This acts as a form of "knock-out" intervention: we overwrite a single layer's activations in the corrupted run with clean activations that are hypothesized to contain specific information (e.g., abstraction, operands, computation), and observe whether this information is reflected in the output level.

To validate the critical layers for each stage, we cross-patch for **numerical abstraction** (Figure 3c), where both the clean and corrupted inputs are numerical problems, but differ in both underlying logic and operands. As shown in Figure 3c, the clean run corresponds to $5+3=8$ and the corrupted to $11-7=4$. We patch hidden states from the clean run into the corrupted run and validate our hypothesis: (i) *Abstraction (L13-14)*: At these layers, operands have not been transferred yet, so patching should only transfer the clean logic. If these layers encode addition logic, the model should apply the clean addition operator to the corrupted operands, computing $11+7=18$ in the remaining forward pass. We expect the log-probability of this *target answer* ('18') to rise. (ii) *Operand Transfer (L15–16)*: L15 begins operand transfer and already contains both clean run logic and operand information. Patching them should increase the log-prob of the clean answer (e.g., '8'), while reducing probability of the corrupted ('4') and target answers ('18'). (iii) *Computation (Layer 18)*: By this point, the full ingredients (abstraction and computation) are available. Patching here should fully recover the clean answer ('8'). We expect the log-prob close to 0. To evaluate these effects, we track log-probabilities across layers for the **target answer** ($18=11+7$, clean logic + corrupted operands) – testing numerical abstraction transfer, the **clean answer** ($8=5+3$) – testing operand alignment and execution, and the **corrupted answer** ($4=11-7$).

To investigate if the abstraction representations can be transferred across problem forms (symbolic/numerical) and templates, and if they can be composed with subsequent computation stage, we cross-patch for **symbolic abstraction** (See Figure 3c) – patching symbolic clean states to numerical corrupted run. Here, clean inputs are symbolic (no concrete numbers), and corrupted inputs are numerical problems with a different underlying logic. This ensures that only abstraction (no operands or computation) is transferred from the clean run, unlike numerical abstraction cross-patching. In the example in Figure 3c, clean run predicts $x-y$, while the corrupted run corresponds to $5+3=8$. By patching clean states from the symbolic problem to the numerical corrupted run, we examine (i) if *symbolic abstractions are also formed* at around L13-14, despite predicting 'x' as the first token, and (ii) if this abstraction ($x-y$), when transferred into numerical corrupted run, can be *used and composed* with corrupted operands $(5,3)$ to compute $5-3=2$. To assess this, we track the per-layer log-probabilities of the **target answer** (clean logic + corrupted operands, $2=5-3$) and **corrupted answer** ($8=5+3$), and omit the clean answer 'x'. If symbolic abstraction transfer occurs, we expect an increase in the target answer log-prob, and a corresponding decrease in the corrupted answer starting around L13. Note that since the symbolic clean states across layers are predicting 'x', we expect both answer log-probs to drop.

Results Figure 5a shows results for **numerical abstraction** cross-patching results corresponding to the example illustrated in Figure 3. As expected, the target answer log-probability ('18') begins rising at L13 (abstraction onset), peaks at L14 (abstraction formed), and drops when clean operands are introduced (L15). The clean answer ('8') log-probability keeps rising from L13 (abstraction) and continue at 15 (operand integration), stabilizing by L18 (computation). The corrupted answer ('4') log-probability drops after L13. These trends hold across underlying logic (Figure 5b-d), confirming the roles of these critical layers as identified earlier. In **symbolic abstraction** cross-patching (Figure 5e-h), we observe consistent behaviour: from L13 onward, the target answer probability increases while the clean answer decreases, eventually flipping. This indicates that (i) abstractions injected via patch-



(a) Cl: $a + b$; Cor: $a - b$ (b) Cl: $a - b$; Cor: $a + b$ (c) Cl: $a \times b$; Cor: $a \div b$ (d) Cl: $a \div b$; Cor: $a \times b$

(e) P. Cl:$x - y$; Cor: $a + b$ (f) Cl: $x - y$; Cor: $a + b$ (g) Cl: $x \times y$; Cor: $a + b$ (h) Cl: $x \div y$; Cor: $a + b$

Figure 5: Cross-patching results for Llama-3 8B with corresponding clean and corrupted run. a, b indicate concrete numerical problems, while x, y indicate symbolic problems. **Top (Numerical Abstraction):** Patching concrete problems with different abstractions shows target log-prob rising at 13 (abstraction onset), peaking at 14 (abstraction formed), then falling as clean operands are introduced. Meanwhile, the clean answer's log-prob rises from 13 (abstraction) and 15 (operand integration), stabilizing at layer 18 (computation). **Bottom (Symbolic Abstraction):** Patching symbolic problems into concrete addition shows target log-probability rising at layer 13, peaking at 15 (where predictions flip), then declining.

ing are composed with corrupted operands to produce valid outputs, and (ii) abstraction representations at L13–14 are invariant to surface form and problem template. Concretely, comparing Figure 5e and Figure 5f, where minimally different templates are used in (e) and random templates in (f), we observe near-identical effects in both cases —suggesting abstraction transfer is template-invariant. Furthermore, (g) and (h) show that injecting symbolic *multiplication* and *division* abstractions into concrete *addition* problems still flips the model's prediction—demonstrating the generality of abstraction transfer. Cross-patching results for other models, and two-operator problems are in Appendix B.3.

Together, these results provide strong support for the abstract-then-compute hypothesis with critical layers for abstraction (L13,14) and computation (L15,16 for operands and L18 for computation), and further demonstrate that: (i) abstraction can be transferred and composed with subsequent computation across surface forms (symbolic/concrete) and templates, and (ii) even at the last position in symbolic problems, when predicting the first output token 'x', middle layers already encode abstraction (e.g., the correct operator), indicating that next-token prediction reflects not just immediate token prediction, but also anticipates future outputs.

6 Conclusion

Disentangled evaluation reveals that, without CoT, models perform better at abstraction than computation, with the latter bottlenecking final-answer accuracy — challenging the view that poor performance always imply reasoning failure. Mechanistic interpretability uncovers an abstract-then-compute mechanism with transferable abstractions. We argue for disentangled evaluation to more precisely assess model abilities and inform architectural design.

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *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)*, pp.

2357–2367, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1245. URL https://aclanthology.org/N19-1245/.

Nora Belrose, Zach Furman, Logan Smith, Danny Halawi, Igor Ostrovsky, Lev McKinney, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*, 2023.

Ziling Cheng, Meng Cao, Marc-Antoine Rondeau, and Jackie Chi Kit Cheung. Stochastic chameleons: Irrelevant context hallucinations reveal class-based (mis)generalization in llms, 2025. URL https://arxiv.org/abs/2505.22630.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023.

Asma Ghandeharioun, Avi Caciularu, Adam Pearce, Lucas Dixon, and Mor Geva. Patchscopes: A unifying framework for inspecting hidden representations of language models. *arXiv preprint arXiv:2401.06102*, 2024.

Nicholas Goldowsky-Dill, Chris MacLeod, Lucas Sato, and Aryaman Arora. Localizing model behavior with path patching. *arXiv preprint arXiv:2304.05969*, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.211. URL https://aclanthology.org/2024.acl-long.211/.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

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

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. MAWPS: A math word problem repository. In Kevin Knight, Ani Nenkova, and Owen Rambow (eds.), *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1152–1157, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1136. URL https://aclanthology.org/N16-1136/.

Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. Gsm-plus: A comprehensive benchmark for evaluating the robustness of llms as mathematical problem solvers. *arXiv preprint arXiv:2402.19255*, 2024.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 158–167, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1015. URL https://aclanthology.org/P17-1015/.

Yu Lu Liu, Su Lin Blodgett, Jackie Cheung, Q. Vera Liao, Alexandra Olteanu, and Ziang Xiao. ECBD: Evidence-centered benchmark design for NLP. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16349–16365, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.861. URL https://aclanthology.org/2024.acl-long.861/.

Kevin Meng, David Bau, Alex J Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=-h6WAS6eE4.

Jack Merullo, Carsten Eickhoff, and Ellie Pavlick. Language models implement simple Word2Vec-style vector arithmetic. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5030–5047, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.281. URL https://aclanthology.org/2024.naacl-long.281/.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. A diverse corpus for evaluating and developing english math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 975–984, 2020.

Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=AjXkRZIvjB.

Swaroop Mishra, Matthew Finlayson, Pan Lu, Leonard Tang, Sean Welleck, Chitta Baral, Tanmay Rajpurohit, Oyvind Tafjord, Ashish Sabharwal, Peter Clark, and Ashwin Kalyan. LILA: A unified benchmark for mathematical reasoning. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5807–5832, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.392. URL https://aclanthology.org/2022.emnlp-main.392/.

Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. Arithmetic without algorithms: Language models solve math with a bag of heuristics. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=O9YTt26r2P.

nostalgebraist. interpreting gpt: the logit lens, 2020. URL https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens.

Andreas Opedal, Alessandro Stolfo, Haruki Shirakami, Ying Jiao, Ryan Cotterell, Bernhard Schölkopf, Abulhair Saparov, and Mrinmaya Sachan. Do language models exhibit the same cognitive biases in problem solving as human learners? *arXiv preprint arXiv:2401.18070*, 2024.

OpenAI. Gpt-4o mini: Advancing cost-efficient intelligence. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/, 2024.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2080–2094, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.168. URL https://aclanthology.org/2021.naacl-main.168/.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, pp. 31210–31227. PMLR, 2023.

Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pp. 1132–1142, 2015.

Zayne Rea Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. To cot or not to cot? chain-of-thought helps mainly on math and symbolic reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=w6nlcS8Kkn.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=NpsVSN6o4ul.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yunyang Wan, Yuqi Liu, Zeyu Cui, Zhenru Zhang, Zihan Qiu, Shanghaoran Quan, and Zekun Wang. Qwen2.5 technical report. *ArXiv*, abs/2412.15115, 2024. URL https://api.semanticscholar.org/CorpusID:274859421.

Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of language models: Part 2.1, grade-school math and the hidden reasoning process. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Tn5B6Udq3E.

Fred Zhang and Neel Nanda. Towards best practices of activation patching in language models: Metrics and methods. *arXiv preprint arXiv:2309.16042*, 2023.

Hugh Zhang, Jeff Da, Dean Lee, Vaughn Robinson, Catherine Wu, William Song, Tiffany Zhao, Pranav Vishnu Raja, Charlotte Zhuang, Dylan Z Slack, Qin Lyu, Sean M. Hendryx, Russell Kaplan, Michele Lunati, and Summer Yue. A careful examination of large language model performance on grade school arithmetic. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024a. URL https://openreview.net/forum?id=RJZRhMzZzH.

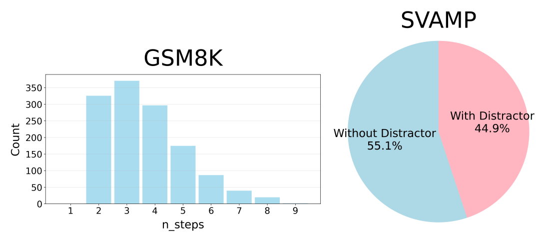
Wei Zhang, Chaoqun Wan, Yonggang Zhang, Yiu-ming Cheung, Xinmei Tian, Xu Shen, and Jieping Ye. Interpreting and improving large language models in arithmetic calculation. *arXiv preprint arXiv:2409.01659*, 2024b.

Xiaotian Zhang, Chunyang Li, Yi Zong, Zhengyu Ying, Liang He, and Xipeng Qiu. Evaluating the performance of large language models on gaokao benchmark. *arXiv preprint arXiv:2305.12474*, 2023.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. AGIEval: A human-centric benchmark for evaluating foundation models. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 2299–2314, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-naacl.149. URL https://aclanthology.org/2024.findings-naacl.149/.

A Disentangled evaluation details and additional results

A.1 Dataset statistics



(a) N-steps (b) Distractor

Figure 6: Distribution of problem characteristics by number of reasoning steps (GSM8K) and presence of distractors (SVAMP).

A.2 Symbolic variant creation for GSM8K and SVAMP

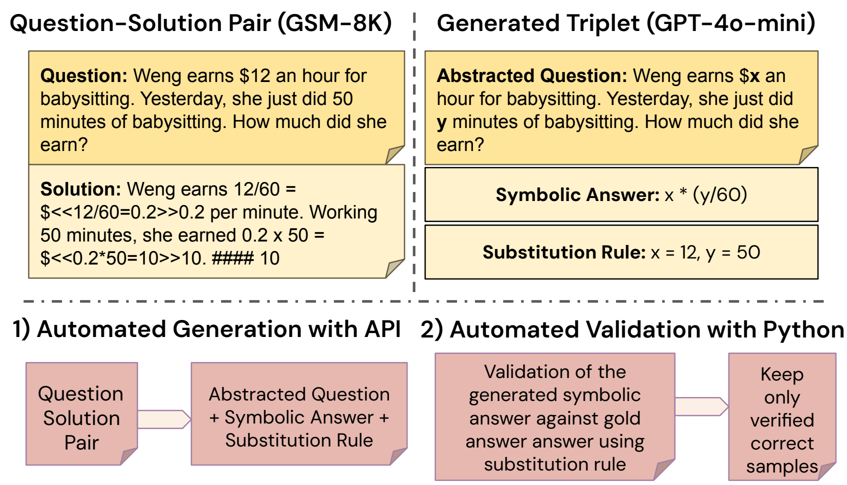


Figure 7: **Generate-then-validate pipeline**: We use API calls to obtain abstract question-answer-substitution triplets from the concrete question-solution pair from GSM-8K, then validate them against gold answer using `sympy`. Triplets that fail this check are manully reviewer and corrected.

All our evaluations are conducted on the GSM8K test set and the full SVAMP dataset. To support our evaluation of abstract formulation and arithmetic computation in Section 4, we construct symbolic question and expression answer variants for both SVAMP and GSM-8K. Examples are shown in Table 4.

For SVAMP, which already includes both the expression (e.g., 20×10) and the final numerical answer (e.g., 200), we create symbolic abstraction variants by replacing all numeric values with symbolic variables (e.g., x, y) in both the question and the corresponding expression. This preserves the structure and semantics of the original problem while abstracting away from the concrete numbers. For arithmetic computation variant, we use the paired expression.

For GSM-8K, which lacks such annotations, we generate both the symbolic abstraction variant and the numerical expressions using a two-stage generate-then-validate pipeline (Figure 7). In the generation stage, we use `GPT-4o-mini` (OpenAI, 2024) to produce triplets from original question–solution pairs. Each triplet consists of: (1) a symbolic version of the question, where relevant numbers are replaced with variables while maintaining the semantic content; (2) a symbolic expression that represents the solution in closed-form using those variables; and (3) a substitution rule that maps each variable to its original numeric value. In the validation stage, we verify the correctness of each generated sample. We apply the substitution rule to the symbolic expression, obtaining a numerical expression, then using `sympy` to evaluate the expression, and compare the resulting numeric answer to the gold answer from GSM-8K. Triplets that fail this check are manually reviewed and corrected.

Box 1: Symbolic evaluation prompt

Determine whether the following two mathematical expressions are equivalent. The expressions may be written in simplified or unsimplified symbolic form (e.g., `1/2x + 3`), natural language (e.g., "Susan made `1/2x + 3` buttons") or in LaTeX notation. Consider expressions equivalent if they represent the same mathematical value, even if written differently (e.g., different notation, simplification, or variable order when valid). Respond only with: `True` or `False`.

Example:
1. $z - (y - x)$
2. `z - y + x`
Answer: `True`

1. `Susan made 1/2*x buttons`
2. `0.5x`
Answer: `True`

1. `2(y + x)`
2. $M = 2(y + x)$
Answer: `True`

1. `xz * ((1 - y)/100)`
2. `x × z - (y/100) × (x × z)`
Answer: `True`

Now evaluate:
1. {`symbolic_gold_answer`}
2. {`abstract_generated_answer`}
Answer:

Gold Answer	Model Generation	Our Eval	GPT-4o-mini Eval
$u * (x + y + z)$	$xu + yu + zu$	True	True
$x + x * (1/y)$	$x + (x/y)$	True	True
$0.5(x + yz)$	$z * (y + 1) * x/2$	False	False
$(y + z)/x$	$xz - y = xy$	False	False
$xz * ((1 - y)/100)$	$(x * (1 - y/100) * z)$	False	True
$(12/x) * y$	$y * 12$	False	True

Table 3: Comparison of gold answers, model generations, our annotated correctness, and GPT-4o-mini evaluation on a held-out set of 120 samples.

A.3 Evaluation details

In this section, we detail the evaluation of the four settings. First, we show the instructions used in each settings in Table 5 with and without CoT. The prompt used in each setting is then a concatenation of the instruction and the question.

For the *Original* and *Arithmetic Computation* settings, where the expected output is a final integer answer, we extract the answer following the token ####, remove any accompanying

Dataset	Symbolic Question	Answer	Substitution
GSM8K	I have x liters of orange drink that are y% water and I wish to add it to z liters of pineapple drink that is u% water. But as I pour it, I spill v liters of the orange drink. How much water is in the remaining w liters?	$(y \cdot (x - v) + u \cdot z)/100$	$x = 10$, $y = \frac{2}{3}$, $z = 15$, $u = \frac{3}{5}$, $v = 1$, $w = 24$
GSM8K	Jerry has a flock of chickens. The red chickens produce x eggs a day, and the white chickens produce y eggs a day. Every day Jerry collects z eggs. If he has u more white chickens than red chickens, how many red chickens does he have?	$(z - u \cdot y)/(x + y)$	$x = 3$, $y = 5$, $z = 42$, $u = 2$
GSM8K	Adrian's age is x times the age of Harriet, and Harriet is y the age of Zack. Calculate the average age of the three in three years if Harriet is z years old now.	$(x * z + z + (z/y) + 9)/3$	$x = 3$, $y = \frac{1}{2}$, $z = 21$
SVAMP	Each pack of DVDs costs x dollars. If there is a discount of y dollars on each pack	$x - y$	$x = 76$, $y = 25$
SVAMP	An industrial machine worked for x minutes. It can make y shirts a minute.	$x \cdot y$	$x = 4$, $y = 5$
SVAMP	Paco had x salty cookies and y sweet cookies. He ate z sweet cookies and u salty cookies. How many salty cookies did Paco have left?	$x - u$	$x = 26$, $y = 17$, $z = 14$, $u = 9$

Table 4: Constructed symbolic examples from GSM8K and SVAMP datasets.

units, and normalize formatting (e.g., removing commas, dollar signs, percentage symbols, and units like 'g') before comparing it with the gold answer.

For the *Numerical Abstraction* setting, where answers are expected to be *numerical expressions*, we first convert LaTeX-style expressions to Python syntax (when written in Markdown form), then evaluate them using `sympy` to check equivalence with the gold expression.

In the *Symbolic Abstraction* setting, where outputs are *symbolic expressions*, we use `gpt-4o-mini` as an automated evaluator. The prompting to `gpt-4o-mini` is shown in Box 1, and responses are generated with temperature set to 0. To validate this method, we annotated a held-out set of 120 samples manually for correctness, and compared our annotations with the `gpt-4o-mini` evaluator's decisions. We find that `gpt-4o-mini` achieves **94% agreement** with our judgment in identifying symbolic expression equivalence. Example comparisons are shown in Table 3.

A.4 Additional result of disentangled evaluation without CoT

We report zero-shot, no-CoT performance on SVAMP in Figure 8. Compared to GSM8K, SVAMP is a significantly simpler benchmark consisting of math word problems that require only a single reasoning step — namely, a single arithmetic operation. As with GSM8K, models perform better on the abstraction variants than in the original setting, though the performance gap is smaller due to the task's simplicity.

Interestingly, we observe a notable difference from GSM8K: across all model sizes, even small models such as LLAMA 1B and 3B perform well on the *Arithmetic Computation* variant, often outperforming both the abstraction variants and the original setting. This suggests

Table 5: Prompting Strategies, Problem Variants and Instructions

Setting	Strategy	Instruction	Question	Answer
Original	No CoT	Please answer the question directly WITHOUT showing the reasoning process, you MUST write the answer as **an integer** after '####', without including the equation or units.	Weng earns $12 an hour for babysitting. Yesterday, she just did 50 minutes of babysitting. How much did she earn?	10
Original	CoT	**Let's think step by step**, you MUST write the answer as **an integer** after '####' without including the units. Write the answer at the end.	Weng earns $12 an hour for babysitting. Yesterday, she just did 50 minutes of babysitting. How much did she earn?	10
Arithmetic Computation	No CoT	Please answer the question directly WITHOUT showing the reasoning process, you MUST write the answer as **an integer** after '####'	What is the value of 12 * (50/60)?	10
Arithmetic Computation	CoT	**Let's think step by step**, you MUST write the answer as **an integer** after '####' . Write the answer at the end.	What is the value of 12 * (50/60)?	10
Numerical Abstraction	No CoT	Please answer the question directly without showing the reasoning process, you MUST write the **expression** with appropriate round brackets after '####', without including the units, and you DO NOT need to simplify the expression.	Weng earns $12 an hour for babysitting. Yesterday, she just did 50 minutes of babysitting. How much did she earn?	12 * (50/60)
Numerical Abstraction	CoT	**Let's think step by step**, at the end, you MUST write the **expression** with appropriate parenthesis after '####', without including the units, but you DO NOT need to simplify the expression.	Weng earns $12 an hour for babysitting. Yesterday, she just did 50 minutes of babysitting. How much did she earn?	12 * (50/60)
Symbolic Abstraction	No CoT	Please answer the question directly WITHOUT showing the reasoning process, you MUST write the **expression** with appropriate round brackets after '####' without including the units, and you DO NOT need to simplify the expression.	Weng earns $x an hour for babysitting. Yesterday, she just did y minutes of babysitting. How much did she earn?	$x * (y/60)$
Symbolic Abstraction	CoT	**Let's think step by step**, at the end, you MUST write the **expression** with appropriate round brackets after '####' without including the units, but you DO NOT need to simplify the expression.	Weng earns $x an hour for babysitting. Yesterday, she just did y minutes of babysitting. How much did she earn?	$x * (y/60)$

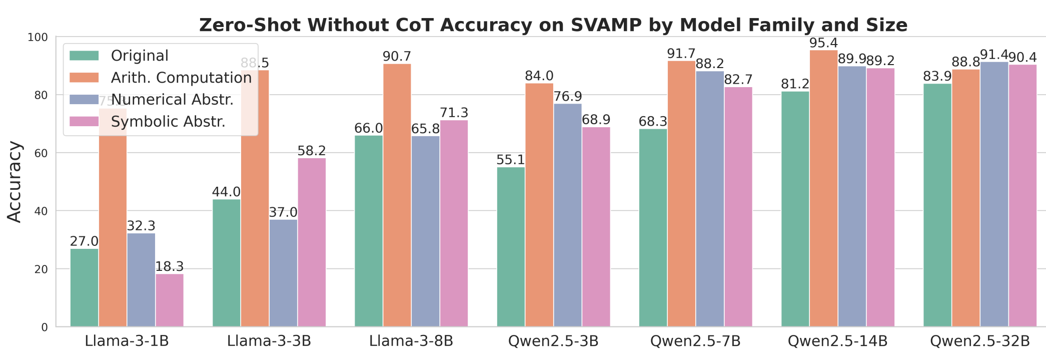


Figure 8: Model zero-shot **without CoT** performance on SVAMP.

that computing one-step expressions (e.g., $5 - 3$) is less challenging than deriving an abstract formulation with only one step. However, in tasks involving multiple steps, abstraction becomes comparatively easier than executing the full computation correctly, as shown in the case of GSM8K. This highlights how model capabilities depend not just on the skill type but also on the complexity of the required operation.

A.5 Additional resuls of disentangled evaluation with CoT

We present the full results on GSM8K for Llama family and Qwen family in Figure 9, and full results on SVAMP for Llama family and Qwen family in Figure 10.

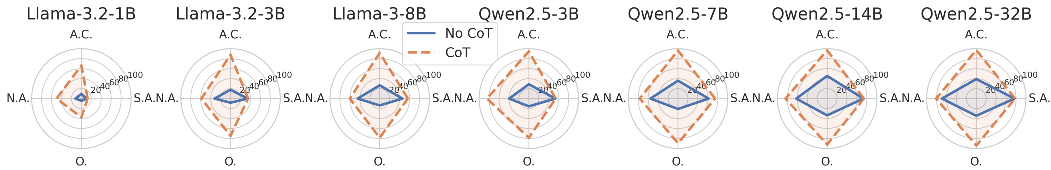


Figure 9: Model zero-shot **with and without CoT** performance on GSM8K. A.C.: Arithmetic Computation; N.A.: Numerical Abstraction; O.: Original; S.A.: Symbolic Abstraction.

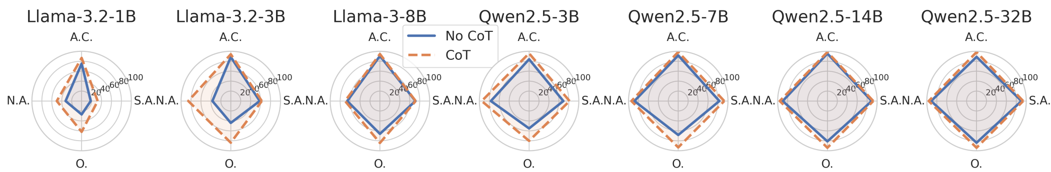


Figure 10: Model zero-shot **with and without CoT** performance on SVAMP. A.C.: Arithmetic Computation; N.A.: Numerical Abstraction; O.: Original; S.A.: Symbolic Abstraction.

A.6 Ablation on symbolic abstraction Variant

To assess the reliability and external validity of the symbolic abstraction evaluation, we perform ablations over symbol order and symbol choice. As illustrated in Figure 11, we compare three settings:

- **Original Symbols**: Variables are consistently represented using a fixed set of letters in order—$x, y, z, u, v, w, p, q, r, s, t$—e.g., $x \times (y/60)$.
- **Reversed Symbols**: The same set of symbols is used, but the order is reversed (e.g., $y \times (x/60)$), preserving the semantic and structural content of the problem while changing the superficial presentation.
- **Random Symbols**: Each original symbol is replaced with a randomly sampled letter from the alphabet, unique to each dataset. This preserves the structure of the expression while removing any consistent identity cues. The mappings are as follows: {`'a': 'h', 'd': 'i', 'm': 's', 'n': 'r', 'p': 'e', 'q': 'l', 'r': 'c', 's': 'v', 't': 'j', 'u': 'm', 'v': 't', 'w': 'o', 'x': 'u', 'y': 'p', 'z': 'b', 'Z': 'f'`}

In Table 6, we observe mild performance degradation with symbol perturbations on both models, (e.g., three-point drop with Reversed and another two points with Random), but models retain strong accuracy compared to the Original setting. This suggests that Symbolic Abstraction is relatively robust to surface-level symbol changes.

Setting	Llama 8B		Qwen 7B	
	No CoT	CoT	No CoT	CoT
original	45.7	56.7	61.5	74.7
reverse	42.8	51.8	61.9	74.8
random	41.0	53.1	58.0	71.9

Table 6: Results of ablation study on symbol choices and symbol order, with and without CoT under zero-shot setting on GSM8K.

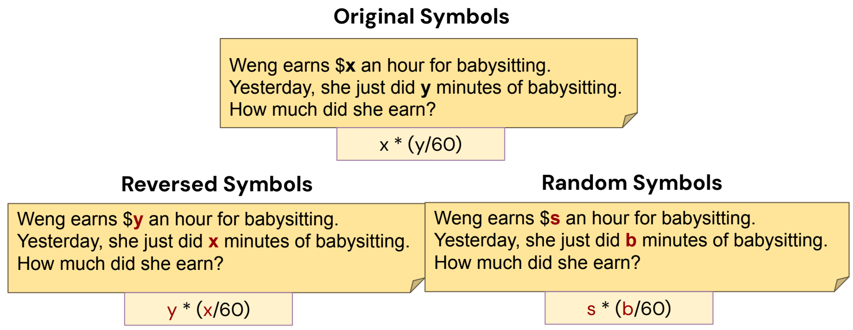


Figure 11: Experiment configurations for the ablation study on symbol choices and symbol order.

B Mechanistic intepretability additional details and results

B.1 Interpretability data construction

To construct a dataset suitable for mechanistic interpretability, we focus on simpler math word problems that require only one or two reasoning steps with one or two basic arithmetic operations (addition, subtraction, multiplication, or division). We deliberately avoid more complex multi-step problems, as model performance on such tasks tends to be poor, potentially confounding interpretability analyses.

For each pair of arithmetic operations—$(x + y, x - y)$ and $(x \times y, x \div y)$ and $(x + y + z, x + y - z, x - y + z, x - y - z)$ — we use a proprietary model to generate 150 template pairs, totaling 1200 templates. These templates are minimally different in semantics but vary across a broad range of topics, verb choices, names, and syntactic structures. Examples are presented in Table 7. For instance, a representative pair might include:

- `[name]` has $\{x\}$ apples. They get $\{y\}$ more apples. How many apples does `[name]` have now? (corresponding to $x + y$)
- `[name]` has $\{x\}$ apples. They give away $\{y\}$ apples. How many apples does `[name]` have now? (corresponding to $x - y$)

Each template is instantiated by replacing the `[name]` placeholder with a randomly selected name from a curated list of 30 English first names, shown below:

`James, Emma, William, Olivia, Benjamin, Charlotte, Henry, Amelia, Alexander, Ava, Samuel, Sophia, Jacob, Mia, Daniel, Lily, Michael, Grace, Ethan, Ella, Jack, Chloe, Lucas, Harper, Thomas, Zoe, Matthew, Nora, Nathan, Isla.`

The numerical placeholders $\{x\}$ and $\{y\}$ are populated with integers ≤ 50, to avoid detokenization issues during model processing.

B.2 Logit attribution and activation patching additional results

Other models We observe a similar *abstract-then-compute* mechanism in other models, including Qwen 2.5 7B and Qwen 2.5 14B. In Qwen 2.5 7B, the abstraction stage occurs around layers 18–20, with the computation stage beginning around layers 22–23. In Qwen 2.5 14B, abstraction takes place around layers 29–32, followed by computation starting at layer 36.

For additional interpretability results using logit lens and activation patching:

- See Figure 12, Figure 13, and Figure 14 for Llama-3 8B on subtraction, multiplication, and division.

Subset	Example Data
$(+,-)$	$(+)$ [name] owns x stuffed animals. A relative sends them y more stuffed animals. How many stuffed animals does [name] have now? $(-)$ [name] owns x stuffed animals. They give y stuffed animals to a younger sibling. How many stuffed animals does [name] have now?
$(+,-)$	$(+)$ [name] finds x seashells at the beach. The next day they find y more seashells. How many seashells does [name] have now? $(-)$ [name] finds x seashells at the beach. The tide washes away y seashells. How many seashells does [name] have now?
$(+,-)$	$(+)$ The storage has x gigabytes free. [name] saves y gigabytes of photos. How much space remains? $(-)$ The storage has x gigabytes free. Cloud storage adds y gigabytes. What is the new capacity?
$(\times,\div)$	$(\times)$ The glacier recedes x inches daily. How much will it shrink after y days? $(\div)$ The glacier retreated x inches over y days. What was the average daily recession?
$(\times,\div)$	$(\times)$ Each server rack uses x kilowatts. What's the total power for y racks? $(\div)$ The data center used x kilowatts across y racks. What was the average per rack?
$(\times,\div)$	$(\times)$ The spaceship's shield blocks x radiation units hourly. How much radiation can it block in y hours? $(\div)$ The shield blocked x units over y hours. What was its average protection rate?
Two operations	$(x+y+z)$ [name] collects x stamps, buys y more, and inherits z. Total stamps? $(x+y-z)$ [name] has x stamps, acquires y more, but loses z. How many left? $(x-y+z)$ [name] owns x stamps, sells y, but trades for z. How many now? $(x-y-z)$ [name] has x stamps, donates y, and ruins z. How many remain?

Table 7: Interpretability dataset examples.

- See Figure 15, Figure 16, Figure 17, and Figure 18 for Qwen 2.5 7B on all four operations: addition, subtraction, multiplication, and division.
- See Figure 19, Figure 20, Figure 21, and Figure 22 for Qwen 2.5 14B on the same set of arithmetic tasks.

Two-operator dataset For two operator dataset, we only report results for Qwen 2.5 7B and Qwen 2.5 14B, because Llama-3 8B only achieve 16.5% accuracy on this dataset.

See Figure 27 and Figure 28 for logit attribution results for Qwen 2.5 7B and Qwen 2.5 14B, respectively.

B.3 Cross-prompt patching additional results

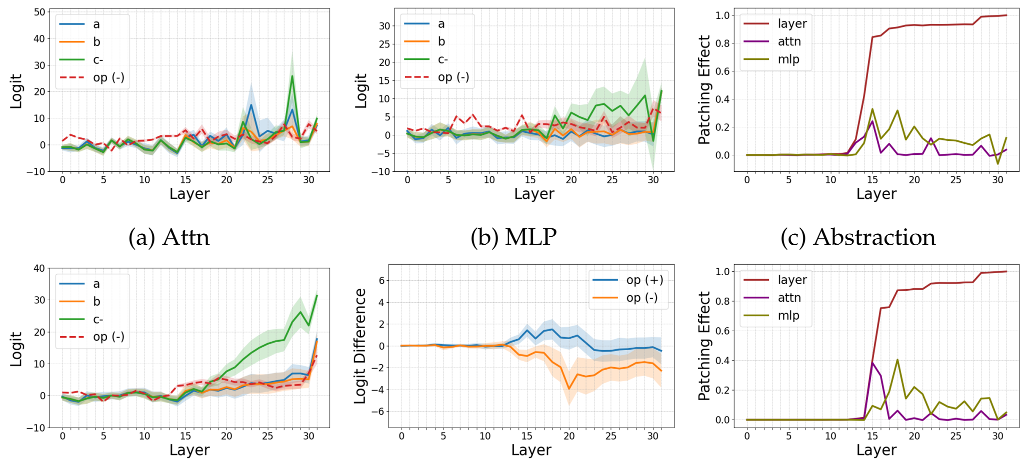
Other models See Figure 23, Figure 24, and Figure 25 for symbolic abstraction cross-prompt patching results (for single operators: $+, -, \times, \div$) on Llama3 8B, Qwen 2.5 7B, and Qwen 2.5 14B, respectively. The results are consistent across models: the likelihood of the target answer peaks at the abstraction stage, while the likelihood of the corrupted answer drops significantly starting from the same stage.

See Figure 26 for numerical abstraction cross-prompt patching results on Llama3 8B, Qwen 2.5 7B, and Qwen 2.5 14B. We observe consistent trends across all models: the probability of the target answer begins to rise at the onset of the abstraction stage and peaks by its end.

Meanwhile, the clean answer probability increases steadily throughout the abstraction stage, reaching a log-probability of 0 at the start of the computation stage.

Two-operator dataset For the two-operator dataset, we report results only for Qwen 2.5 7B and Qwen 2.5 14B, as Llama-3 8B performs poorly on this setting, achieving only 16.5% accuracy.

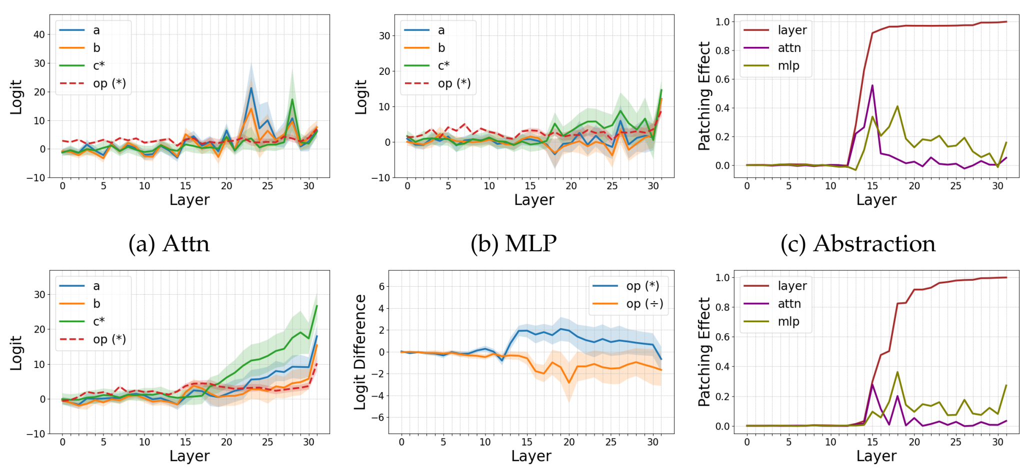
See Figure 29 for symbolic abstraction cross-prompt patching results on Qwen 2.5 7B and Qwen 2.5 14B.



(a) Attn (b) MLP (c) Abstraction

(d) Block (e) Resid Mid (f) Computation

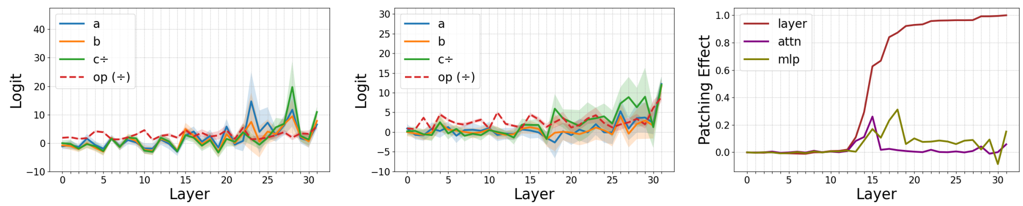
Figure 12: Visualizations of internal computations at last token position in **Llama-3 8B** for **subtraction** math word problems: (a, b, d, e) for logit attribution results, (c, d) activation patching for results.



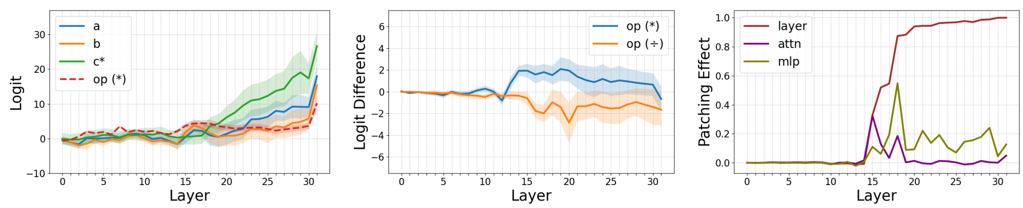
(a) Attn (b) MLP (c) Abstraction

(d) Block (e) Resid Mid (f) Computation

Figure 13: Visualizations of internal computations at last token position in **Llama-3 8B** for **multiplication** math word problems: (a, b, d, e) for logit attribution results, (c, d) activation patching for results.

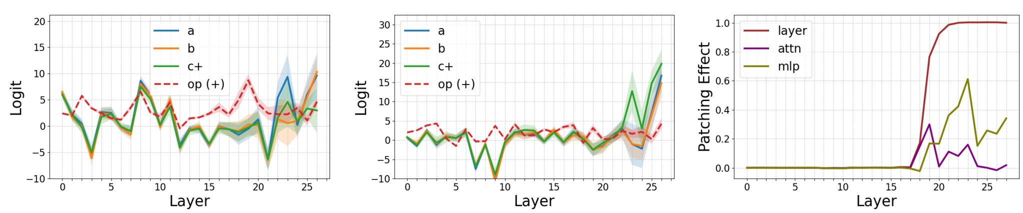


(a) Attn (b) MLP (c) Abstraction

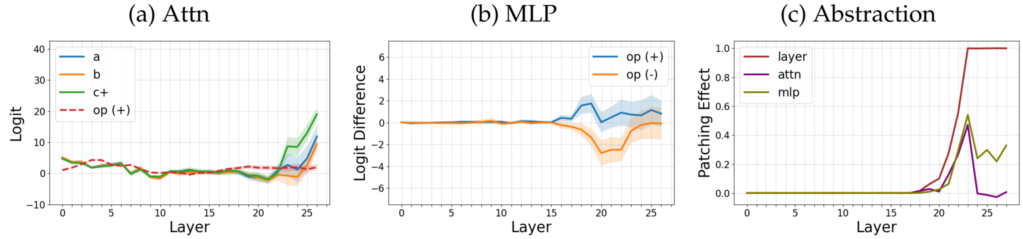


(d) Block (e) Resid Mid (f) Computation

Figure 14: Visualizations of internal computations at last token position in **Llama-3 8B** for **division** math word problems: (a, b, d, e) for logit attribution results, (c, d) activation patching for results.

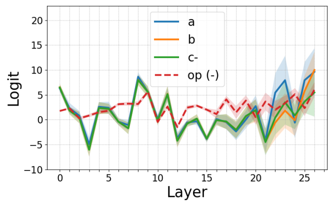


(a) Attn (b) MLP (c) Abstraction

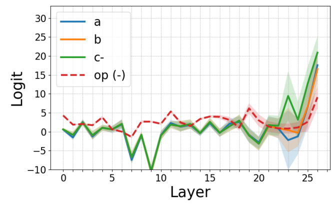


(d) Block (e) Resid Mid (f) Computation

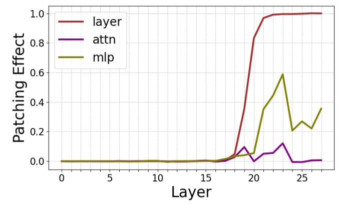
Figure 15: Visualizations of internal computations at last token position in **Qwen 2.5 7B** for **addition** math word problems: (a, b, d, e) for logit attribution results, (c, d) activation patching for results.



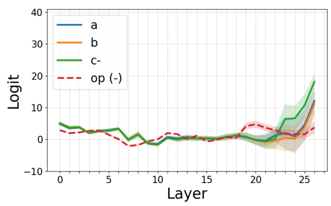
(a) Attn



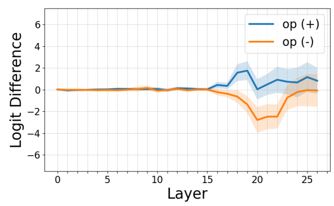
(b) MLP



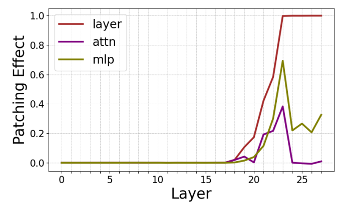
(c) Abstraction



(d) Block

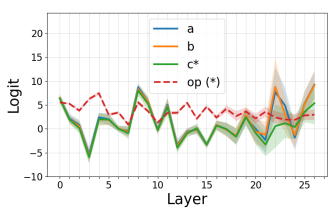


(e) Resid Mid

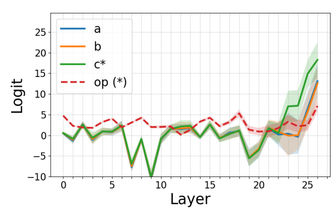


(f) Computation

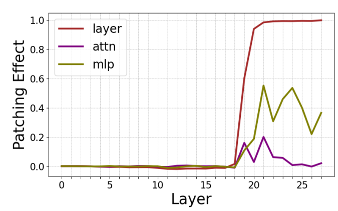
Figure 16: Visualizations of internal computations at last token position in **Qwen 2.5 7B** for **subtraction** math word problems: (a, b, d, e) for logit attribution results, (c, d) activation patching for results.



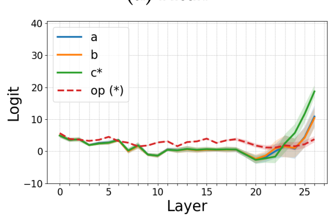
(a) Attn



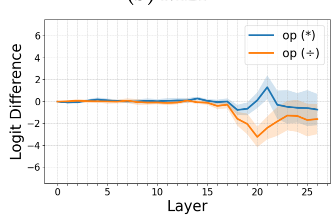
(b) MLP



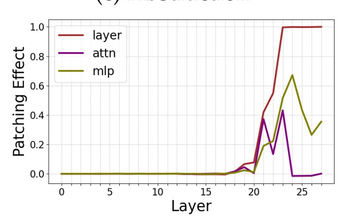
(c) Abstraction



(d) Block

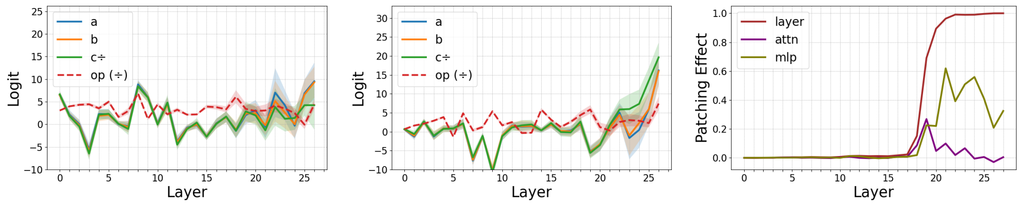


(e) Resid Mid

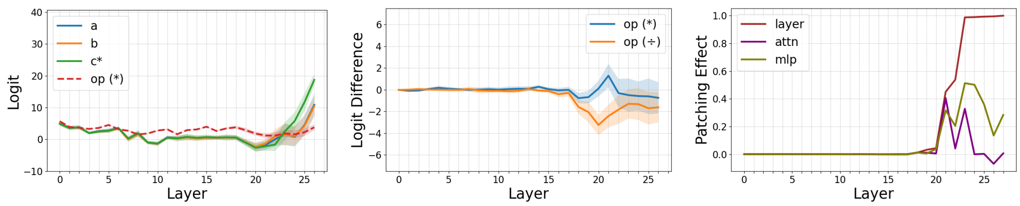


(f) Computation

Figure 17: Visualizations of internal computations at last token position in **Qwen 2.5 7B** for **multiplication** math word problems: (a, b, d, e) for logit attribution results, (c, d) activation patching for results.

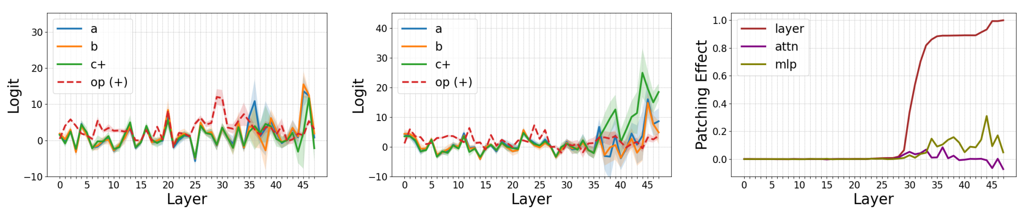


(a) Attn (b) MLP (c) Abstraction

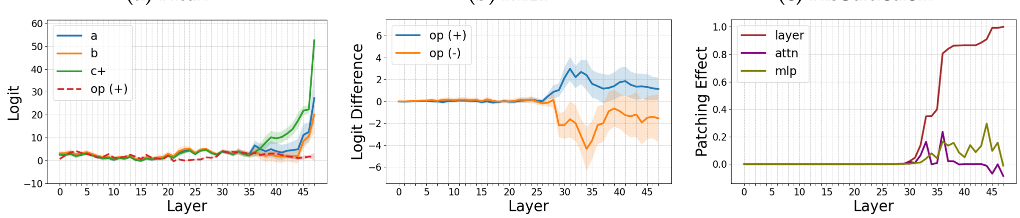


(d) Block (e) Resid Mid (f) Computation

Figure 18: Visualizations of internal computations at last token position in **Qwen 2.5 7B** for **division** math word problems: (a, b, d, e) for logit attribution results, (c, d) activation patching for results.

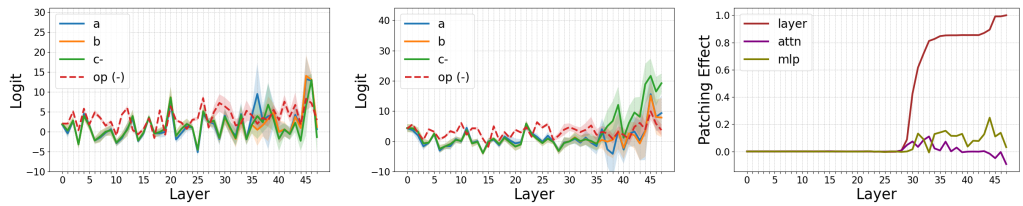


(a) Attn (b) MLP (c) Abstraction

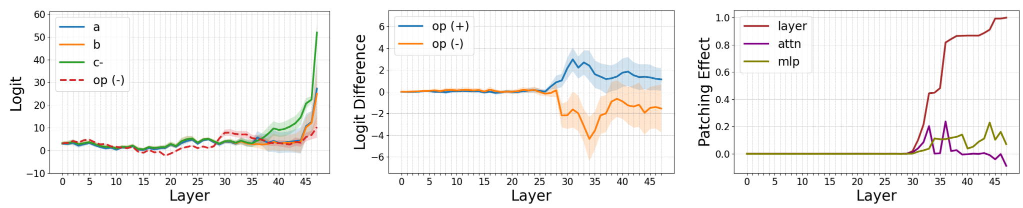


(d) Block (e) Resid Mid (f) Computation

Figure 19: Visualizations of internal computations at last token position in **Qwen 2.5 14B** for **addition** math word problems: (a, b, d, e) for logit attribution results, (c, d) activation patching for results.

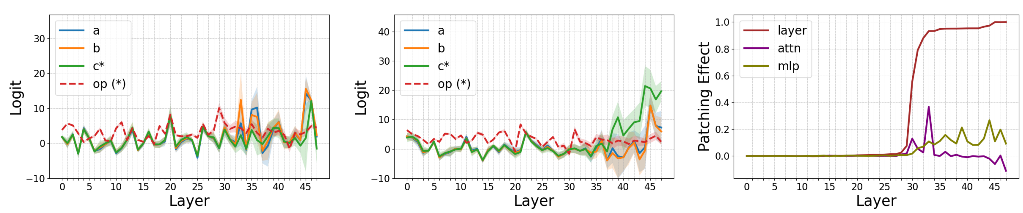


(a) Attn (b) MLP (c) Abstraction

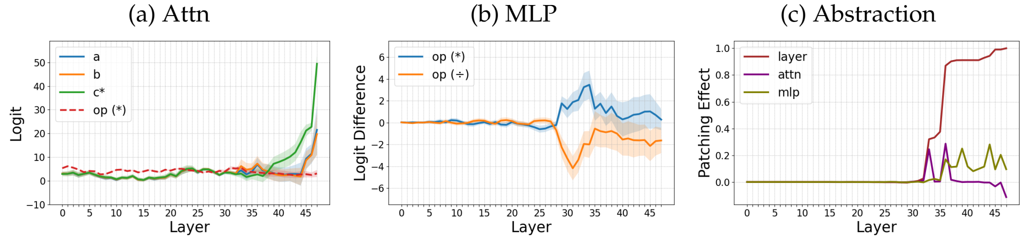


(d) Block (e) Resid Mid (f) Computation

Figure 20: Visualizations of internal computations at last token position in **Qwen 2.5 14B** for **subtraction** math word problems: (a, b, d, e) for logit attribution results, (c, d) activation patching for results.

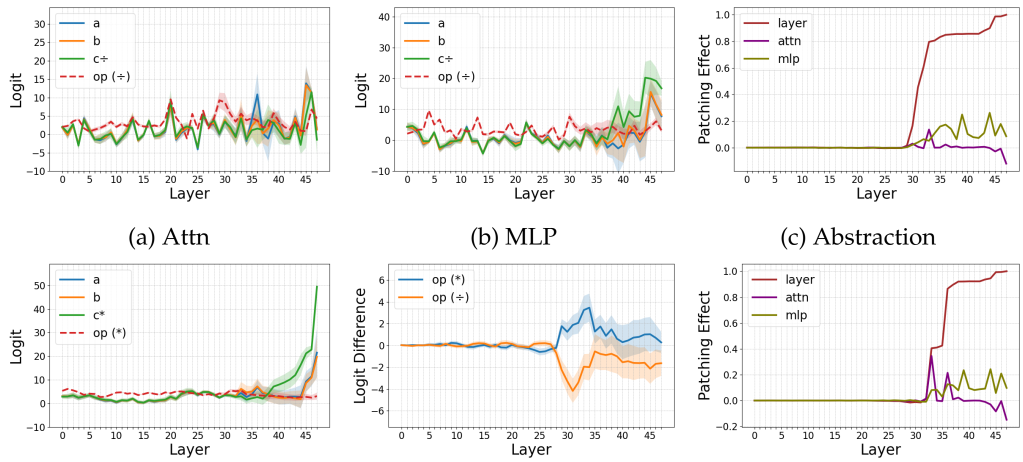


(a) Attn (b) MLP (c) Abstraction



(d) Block (e) Resid Mid (f) Computation

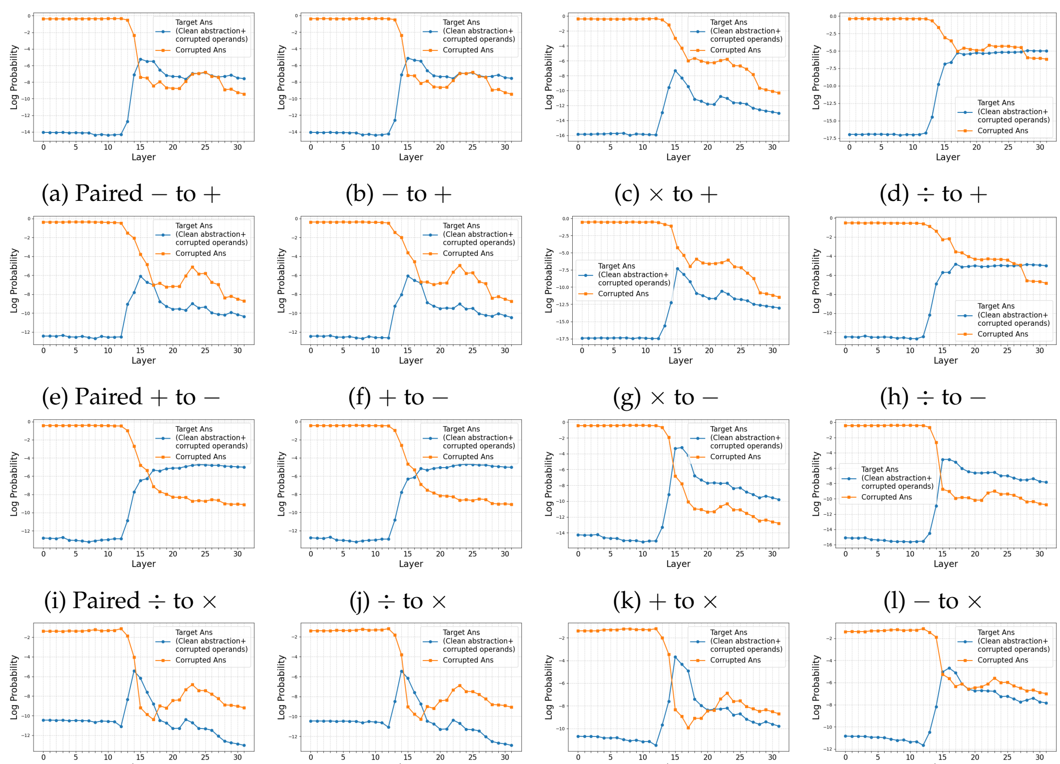
Figure 21: Visualizations of internal computations at last token position in **Qwen 2.5 14B** for **multiplication** math word problems: (a, b, d, e) for logit attribution results, (c, d) activation patching for results.



(a) Attn (b) MLP (c) Abstraction

(d) Block (e) Resid Mid (f) Computation

Figure 22: Visualizations of internal computations at last token position in **Qwen 2.5 14B** for **division** math word problems: (a, b, d, e) for logit attribution results, (c, d) activation patching for results. We label the starting layer of abstraction, operand moving and computation in pink, blue and green, respectively.



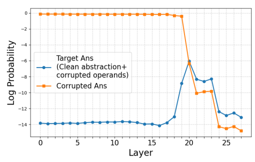
(a) Paired − to + (b) − to + (c) × to + (d) ÷ to +

(e) Paired + to − (f) + to − (g) × to − (h) ÷ to −

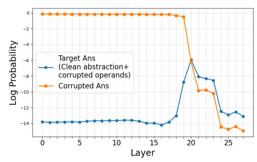
(i) Paired ÷ to × (j) ÷ to × (k) + to × (l) − to ×

(m) Paired × to ÷ (n) × to ÷ (o) + to ÷ (p) − to ÷

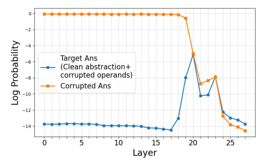
Figure 23: **Llama-3 8B** cross-prompt patching for **symbolic abstraction** results: **First row**: patching symbolic logic to concrete **addition**; **Second row**: patching symbolic logic to concrete **subtraction**; **Third row**: patching symbolic logic to concrete **multiplication**; **Fourth row**: patching symbolic logic to concrete **division**;



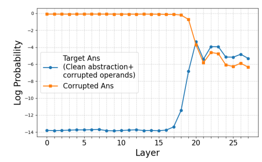
(a) Paired − to +



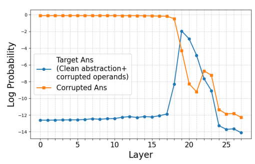
(b) − to +



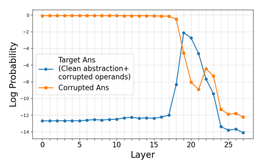
(c) × to +



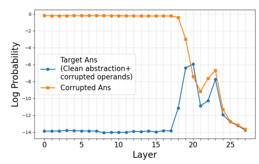
(d) ÷ to +



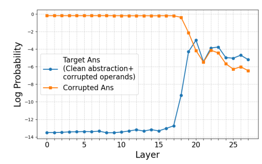
(e) Paired + to −



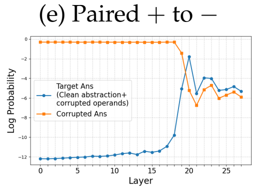
(f) + to −



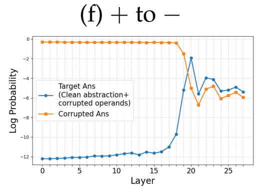
(g) × to −



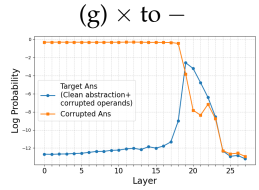
(h) ÷ to −



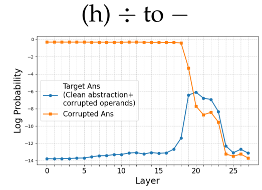
(i) Paired ÷ to ×



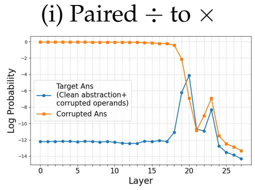
(j) ÷ to ×



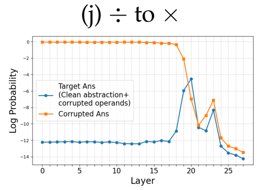
(k) + to ×



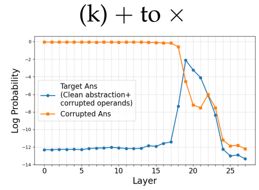
(l) − to ×



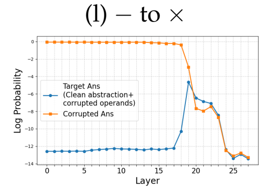
(m) Paired × to ÷



(n) × to ÷

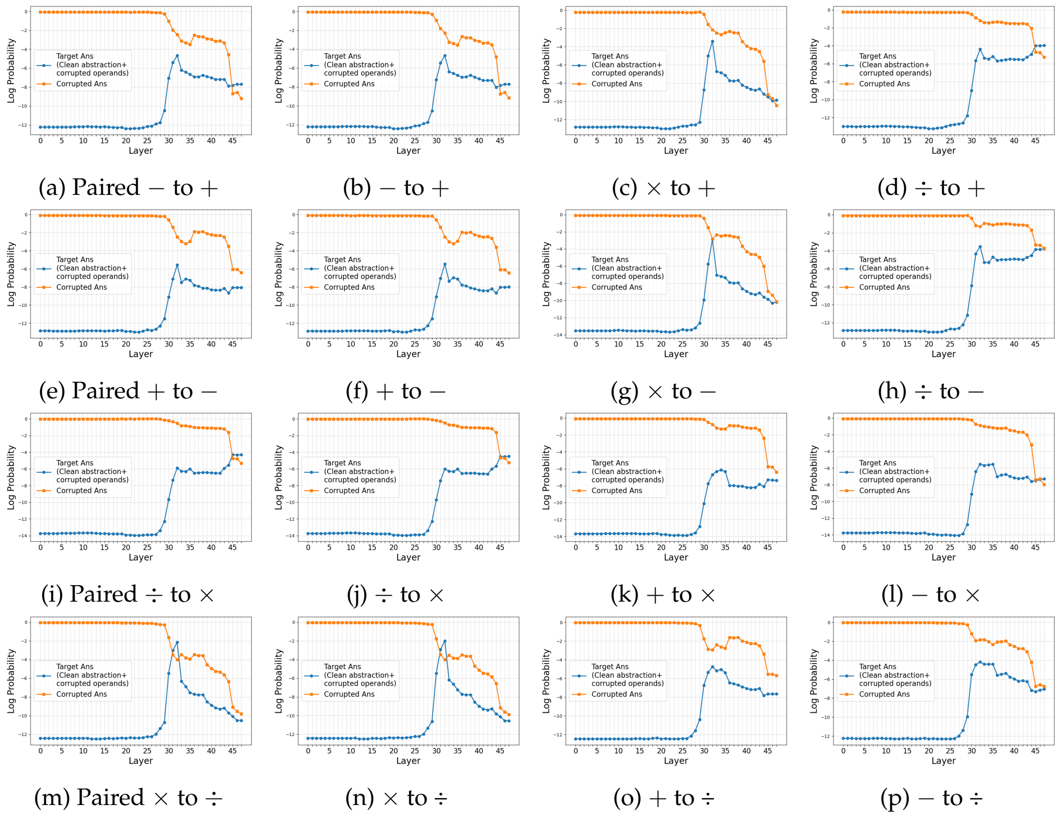


(o) + to ÷



(p) − to ÷

Figure 24: **Qwen-7b** cross-prompt patching for **symbolic abstraction** results: **First row**: patching symbolic logic to concrete **addition**; **Second row**: patching symbolic logic to concrete **subtraction**; **Third row**: patching symbolic logic to concrete **multiplication**; **Fourth row**: patching symbolic logic to concrete **division**;



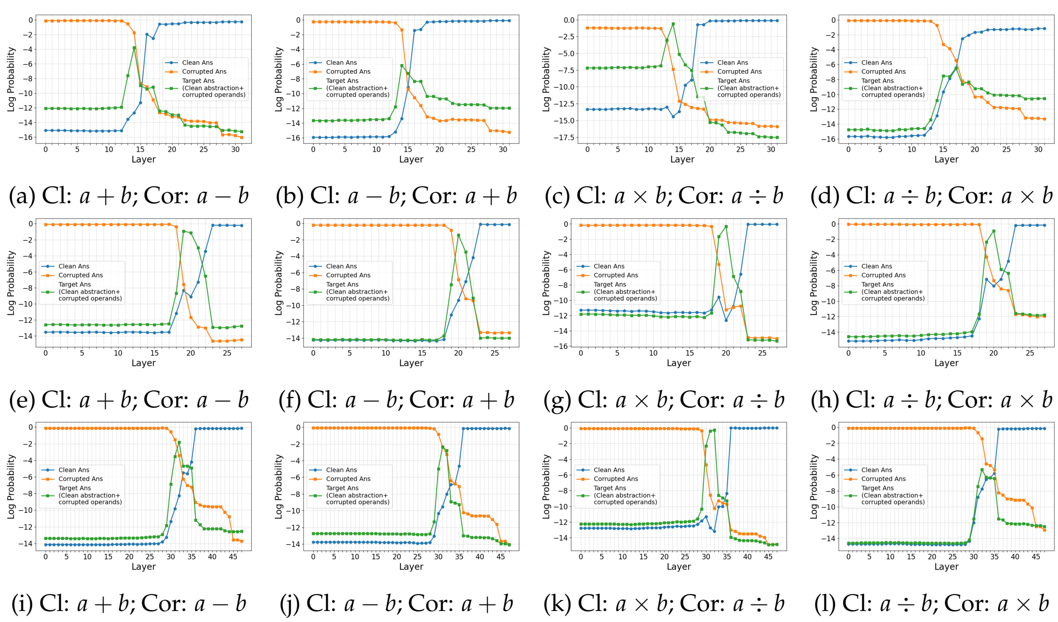
(a) Paired $-$ to $+$ (b) $-$ to $+$ (c) $\times$ to $+$ (d) $\div$ to $+$

(e) Paired $+$ to $-$ (f) $+$ to $-$ (g) $\times$ to $-$ (h) $\div$ to $-$

(i) Paired $\div$ to $\times$ (j) $\div$ to $\times$ (k) $+$ to $\times$ (l) $-$ to $\times$

(m) Paired $\times$ to $\div$ (n) $\times$ to $\div$ (o) $+$ to $\div$ (p) $-$ to $\div$

Figure 25: **Qwen-14b** cross-prompt patching for **symbolic abstraction** results: **First row**: patching symbolic logic to concrete **addition**; **Second row**: patching symbolic logic to concrete **subtraction**; **Third row**: patching symbolic logic to concrete **multiplication**; **Fourth row**: patching symbolic logic to concrete **division**;

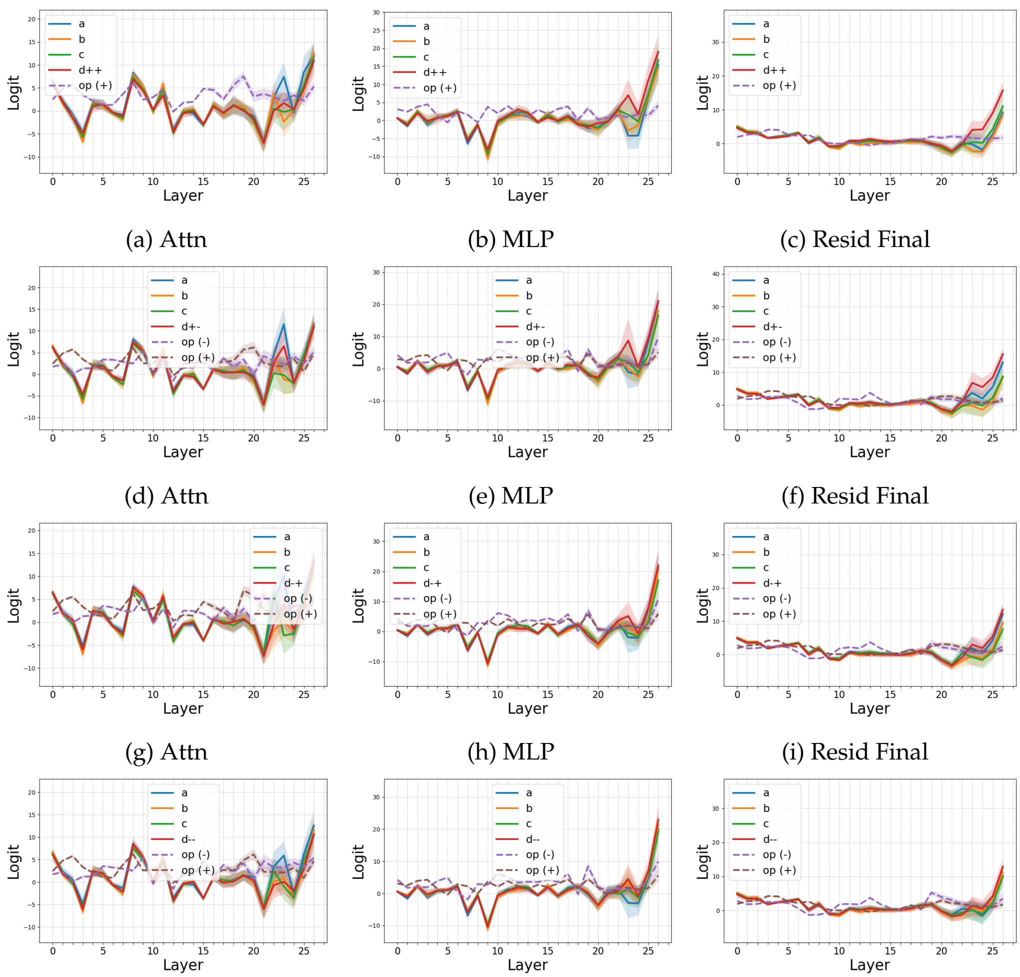


(a) Cl: $a+b$; Cor: $a-b$ (b) Cl: $a-b$; Cor: $a+b$ (c) Cl: $a\times b$; Cor: $a\div b$ (d) Cl: $a\div b$; Cor: $a\times b$

(e) Cl: $a+b$; Cor: $a-b$ (f) Cl: $a-b$; Cor: $a+b$ (g) Cl: $a\times b$; Cor: $a\div b$ (h) Cl: $a\div b$; Cor: $a\times b$

(i) Cl: $a+b$; Cor: $a-b$ (j) Cl: $a-b$; Cor: $a+b$ (k) Cl: $a\times b$; Cor: $a\div b$ (l) Cl: $a\div b$; Cor: $a\times b$

Figure 26: Cross-prompt patching results for **numerical abstraction**. **First row:** results for **Llama-3 8B** with corresponding clean and corrupted run. **Second row:** results for **Qwen2.5 7B** with corresponding clean and corrupted run. **Third row:** results for **Qwen2.5 14B** with corresponding clean and corrupted run.



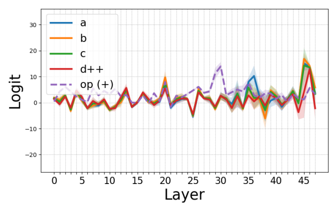
(a) Attn (b) MLP (c) Resid Final

(d) Attn (e) MLP (f) Resid Final

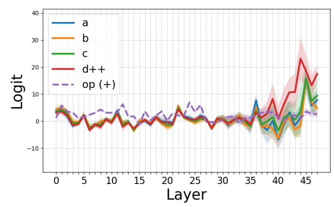
(g) Attn (h) MLP (i) Resid Final

(j) Attn (k) MLP (l) Resid Final

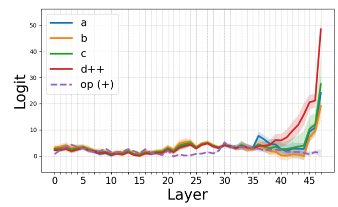
Figure 27: Visualizations of internal computations at last token position in **Qwen 2.5 7B** for **two-operation** math word problems. **First row:** for $a + b + c$. **Second row:** for $a + b - c$. **Third row:** for $a - b + c$. **Fourth row:** for $a - b - c$.



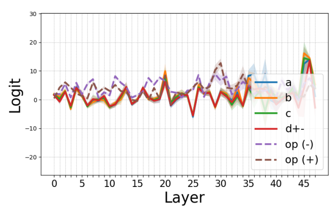
(a) Attn



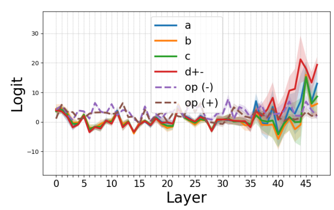
(b) MLP



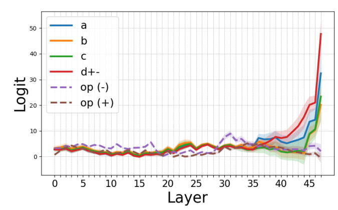
(c) Resid Final



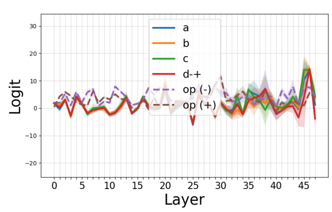
(d) Attn



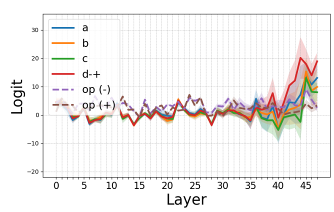
(e) MLP



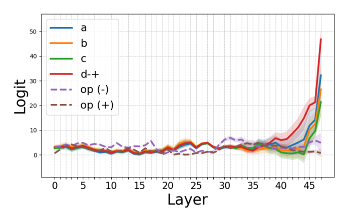
(f) Resid Final



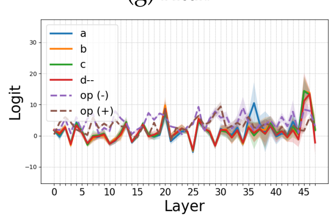
(g) Attn



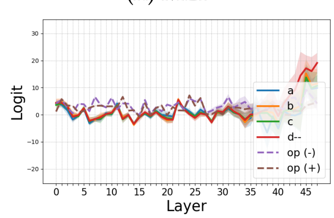
(h) MLP



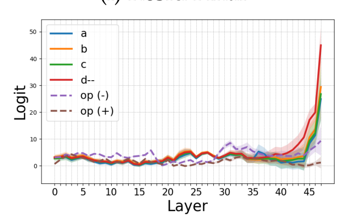
(i) Resid Final



(j) Attn

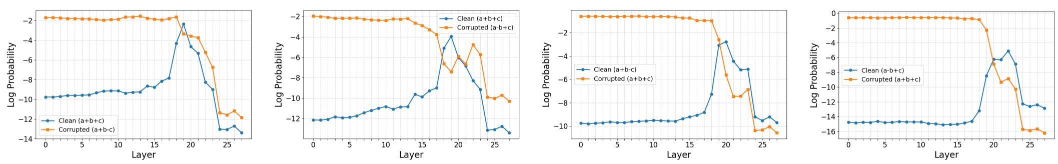


(k) MLP

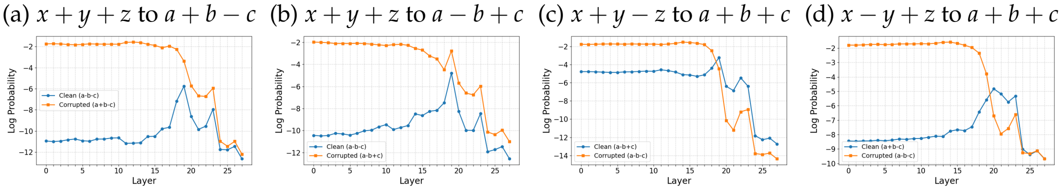


(l) Resid Final

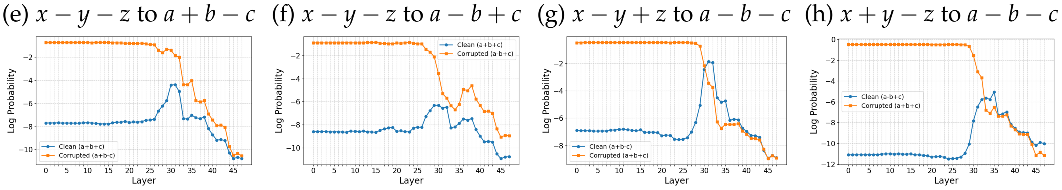
Figure 28: Visualizations of internal computations at last token position in **Qwen 2.5 14B** for **two-operation** math word problems. **First row:** for $a + b + c$. **Second row:** for $a + b - c$. **Third row:** for $a - b + c$. **Fourth row:** for $a - b - c$.



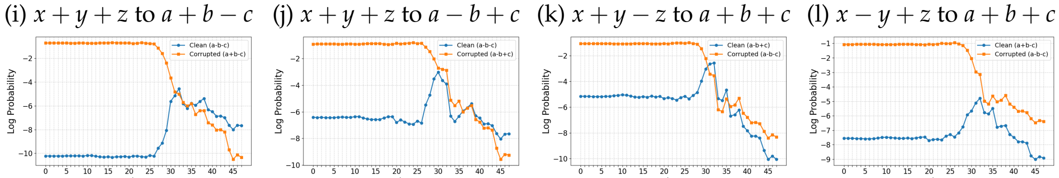
(a) $x + y + z$ to $a + b - c$ (b) $x + y + z$ to $a - b + c$ (c) $x + y - z$ to $a + b + c$ (d) $x - y + z$ to $a + b + c$



(e) $x - y - z$ to $a + b - c$ (f) $x - y - z$ to $a - b + c$ (g) $x - y + z$ to $a - b - c$ (h) $x + y - z$ to $a - b - c$



(i) $x + y + z$ to $a + b - c$ (j) $x + y + z$ to $a - b + c$ (k) $x + y - z$ to $a + b + c$ (l) $x - y + z$ to $a + b + c$



(m) $x - y - z$ to $a + b - c$ (n) $x - y - z$ to $a - b + c$ (o) $x - y + z$ to $a - b - c$ (p) $x + y - z$ to $a - b - c$

Figure 29: **Two-operation** cross-prompt patching for **symbolic abstraction** results: **First row & Second row**: patching symbolic logic to concrete problems for Qwen 2.5 7B. **Third row & Fourth row**: patching symbolic logic to concrete problems for Qwen 2.5 14B.