COMPUTATION ALIGNMENT FOR CHAIN-OF-THOUGHT DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Chain-of-thought (CoT) distillation transfers reasoning capabilities from large teacher models to smaller student models by training on teacher-generated rationales. However, existing methods primarily align observable outputs or intermediate representations, leaving a fundamental question unaddressed: whether the student's computation actually follows the reasoning process prescribed by the teacher's CoT. We identify this overlooked problem as computation alignment and formalize it as the overlap between computational circuits underlying intermediate reasoning steps and final-answer generation, recovered via multi-step circuit tracing. Directly optimizing circuit overlap during distillation, however, is computationally prohibitive. To address this, we propose Computation-Aligned Distillation (*CAD*), which exploits the sensitivity discrepancy between reasoning and answer circuits to construct a lightweight proxy. Across diverse settings, *CAD* consistently improves distillation performance over corresponding baselines without it.

1 INTRODUCTION

Chain-of-thought (CoT) distillation uses teacher-generated reasoning chains (Wei et al., 2022; Kojima et al., 2022; Wang et al., 2022) as step-by-step supervision, training students to learn not only "what answer to produce" but "how to reach it" (Magister et al., 2023; Ho et al., 2023; Hsieh et al., 2023). Existing methods pursue this goal by aligning student and teacher reasoning chains at the token (Hsieh et al., 2023; Ho et al., 2023; Magister et al., 2023; Wang et al., 2022) or representation level (Zhuang et al., 2025; Hao et al., 2023), implicitly assuming that reproducing the content or structure of the teacher's reasoning steps is sufficient to transfer the underlying problem-solving strategy.

However, we argue that this assumption has a fundamental gap: existing methods constrain only what reasoning steps the student generates, never whether the student's answer generation actually follows those steps. As a result, a student can produce reasoning chains that closely match the teacher's while arriving at the answer through an entirely different internal process—reducing CoT to a stylistic output rather than a genuine computational pathway. To address this, we introduce **computation alignment: the computation process by which the student produces the final answer should be consistent with the problem-solving steps described in its CoT.** [1] This constraint complements existing teacher–student CoT alignment, enabling effective optimization of the computation underlying answer generation. This computation, rather than the generated CoT itself, constitutes the student's actual reasoning capability.

To this end, we apply circuit tracing to mechanistically analyze computation alignment—a standard approach for characterizing a model's computation process (Lindsey et al., 2025; Dai et al., 2025; Zhao et al., 2026). To overcome the single-token generation bottleneck of conventional tracing methods (Ameisen et al., 2025; Kamath et al., 2025; Palit et al., 2026), we propose multi-step circuit tracing for CoT reasoning. Specifically, we recursively compose local attribution graphs across reasoning steps: whenever tracing reaches an intermediate reasoning token, we treat it as a new

[1] Notably, this concern is orthogonal to CoT faithfulness: faithfulness studies the causal relation between the CoT and the answer, whereas computation alignment requires the computation underlying answer generation to match the problem-solving process specified by the CoT.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

target and continue tracing backward until reaching the original input query. Computation alignment is then mechanistically defined as the degree of overlap between the reasoning circuit (input → intermediate reasoning steps) and the answer circuit (input → final answer).

Directly optimizing circuit overlap, however, is computationally prohibitive: circuit tracing alone is expensive, and repeatedly reconstructing circuits throughout training is untenable at scale. To make computation alignment practically optimizable, we propose Computation-Aligned Distillation (*CAD*), a plug-in module that replaces explicit circuit comparison with a lightweight perturbation-based proxy. The key observation is that overlapping circuits produce consistent hidden-state shifts under the same input perturbation, whereas disjoint circuits produce divergent shifts. *CAD* exploits this property by constructing semantically equivalent perturbed inputs, measuring the hidden-state shifts at both reasoning-step and answer positions, and regularizing their discrepancy during distillation. As the shift patterns at the two positions converge, the underlying computational pathways increasingly overlap, steering the student toward reasoning–computation alignment. In summary, our contributions are as follows:

(1) We identify computation alignment, a previously unstudied property requiring that a student's forward-pass computation correspond to the reasoning procedure its chain of thought describes.

(2) We formalize computation alignment as circuit overlap via multi-step circuit tracing, and propose Computation-Aligned Distillation (*CAD*), a plug-in module that optimizes this objective through a differentiable perturbation-based proxy for circuit overlap during distillation.

(3) Extensive experiments across datasets, student models, teacher models, and base distillation methods show that *CAD* consistently improves reasoning performance, with *CAD*-augmented classic CoT distillation matching or surpassing state-of-the-art approaches.

2 Preliminaries

2.1 Single-Step Circuit Tracing

Given a target model $\mathcal{M}$ and a sequence under analysis $s = (x, o)$—x is an input query and $o = (o_1, \ldots, o_t)$ is the model-generated output, circuit tracing (Ameisen et al., 2025) attributes the prediction logit ℓ_t for output token o_t to an interpretable computational circuit within $\mathcal{M}$, by tracing backward from ℓ_t through intermediate representations to context tokens in $s_t = (x, o_{<t})$ (see right panel of Figure 1).

Since MLP activations entangle multiple concepts in dense vectors, circuit tracing replaces each MLP with a cross-layer transcoder (CLT; Dunefsky et al. (2024)) that decomposes layer-wise computation into sparse, interpretable features written additively to the residual stream, thereby making each feature's contribution individually attributable. On this basis, circuit tracing constructs an attribution graph $\mathcal{G}_t = (\mathcal{N}_t, \mathcal{E}_t, \mathcal{W}_t)$, with node set $\mathcal{N}_t$, directed edge set $\mathcal{E}_t$, and edge-weight function w_t. The graph is initialized as $\mathcal{N}_t = \{\ell_t\}$, $\mathcal{E}_t = \emptyset$, $\mathcal{W}_t = \emptyset$, with ℓ_t serving as the initial attribution node.

For a current attribution node v, the algorithm computes attribution weights with respect to every candidate source u at preceding layers—either a CLT feature or a context token in s_t:

$$w_{uv} = f_u \cdot e_v^\top d_u, \tag{1}$$

where $f_u d_u$ is the write vector of u to the residual stream and e_v is the read direction of v. The scalar weight w_{uv} thus quantifies how much of u's residual-stream contribution is absorbed by v.

The top-k sources ranked by $|w_{uv}|$ are selected to update $\mathcal{G}_t$: each selected u is added to $\mathcal{N}_t$, the directed edges (u, v) to $\mathcal{E}_t$, and the weight w_{uv} to $\mathcal{W}_t$. Every newly added CLT feature node in turn becomes an attribution node subject to the same procedure, while context tokens serve as leaf nodes admitting no further decomposition.

We treat single-step circuit tracing as a basic tracing operator:

$$\mathrm{SCT}(\mathrm{s_t}, \ell_t) \to \mathcal{G}_t. \tag{2}$$

Table 4 (Appendix A) summarizes the notation introduced here along with other key symbols used throughout this paper.

3 METHODOLOGY

3.1 MECHANISTIC CHARACTERIZATION OF COMPUTATION ALIGNMENT

In this section, we first characterize the model's underlying computation process at the mechanistic level (Section 3.1.1), and then formally define computation alignment on this basis (Section 3.1.2).

3.1.1 MULTI-STEP CIRCUIT TRACING

Our goal is to characterize how the model internally resolves the input query. This requires recovering the full computational pathway from the original input x to the target token o_t, rather than only the local attribution from the current context s_t to o_t within a single generation step (Section 2). We therefore introduce multi-step circuit tracing, which iteratively applies the single-step circuit tracing operator across generation steps and composes the resulting local attribution graphs into a global graph $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \mathcal{W})$.

Concretely, we maintain $\mathcal{G}$ alongside an expansion frontier $\mathcal{Q}$—a priority queue that collects generated tokens in $o_{\leq}t$ whose internal computation has not yet been traced back to x. We initialize $\mathcal{G} \leftarrow \emptyset$ and $\mathcal{Q} \leftarrow \{o_t\}$, and iteratively excute the following three operations:

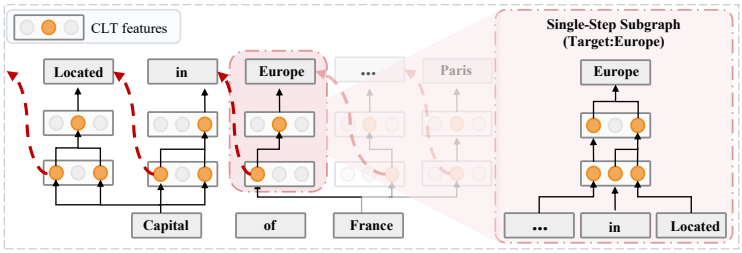


Figure 1: Multi-step circuit tracing from "Capital of France?" to "Paris", recursively extending single-step tracing (e.g., from "Capital of France? Located in" to "Europe".)

(i) Expand. Pop an attribution target o_i from the queue $\mathcal{Q}$ and invoke single-step circuit tracing (Equation 2) on its corresponding autoregressive context:

$$\mathcal{G}_i = (\mathcal{N}_i, \mathcal{E}_i, \mathcal{W}_i) \leftarrow \mathrm{SCT}(s_i, \ell_i), \tag{3}$$

where $s_i = (x, c_{<t})$ and ℓ_i denotes the logit corresponding to o_i. The resulting local graph $\mathcal{G}_i$ is rooted at ℓ_i, with context tokens in s_i as leaf nodes.

(ii) Merge. Integrate $\mathcal{G}_i$ into the global graph $\mathcal{G}$. For $i = t$ (the initial iteration), we directly set $\mathcal{G} \leftarrow \mathcal{G}_i$. For $i < t$, the token o_i appears as a leaf in $\mathcal{G}$. We replace this leaf with the subgraph $\mathcal{G}_i$: remove o_i from the global node set and insert all nodes of $\mathcal{G}_i$ except its root ℓ_i:

$$\mathcal{N} \leftarrow \big(\mathcal{N} \setminus \{o_i\}\big) \cup \big(\mathcal{N}_i \setminus \{\ell_i\}\big). \tag{4}$$

For every upstream edge $(u, \ell_i) \in \mathcal{E}_i$ and downstream edge $(o_i, v) \in \mathcal{E}$, remove edge (o_i, v) and create a composed edge (u, v), with weight updated via chain multiplication:

$$\mathcal{W}(u, v) \leftarrow \mathcal{W}_i(u, \ell_i)\, \mathcal{W}(o_i, v). \tag{5}$$

(iii) Prune. Evaluate each node's contribution to the final target ℓ_t via the global influence score:

$$\phi(u; \ell_t) = \big[(I - \widetilde{\mathcal{W}})^{-1} - I\big]_{u, \ell_t}, \tag{6}$$

where $\widetilde{\mathcal{W}}$ is the row-normalized absolute weight matrix and $(I - \widetilde{\mathcal{W}})^{-1} = \sum_{k=0}^{\infty} \widetilde{\mathcal{W}}^k$ aggregates contributions over all path lengths. To control the attribution cost, we remove every node with $\phi(u; \ell_t) < \tau$, together with its incident edges and weight. Among the surviving leaves, any generated tokens o_j $(j < i)$ not yet expanded is added to $\mathcal{Q}$, prioritized by their global influence $\phi(o_j; \ell_t)$.

The iteration continues from the highest-priority token in $\mathcal{Q}$ and terminates when $\mathcal{Q} = \emptyset$ or the cumulative influence exceeds a prescribed threshold. We denote the complete procedure as

$$\mathrm{MCT}(\mathrm{x}, \mathrm{o_t}) \rightarrow \mathcal{G}. \tag{7}$$

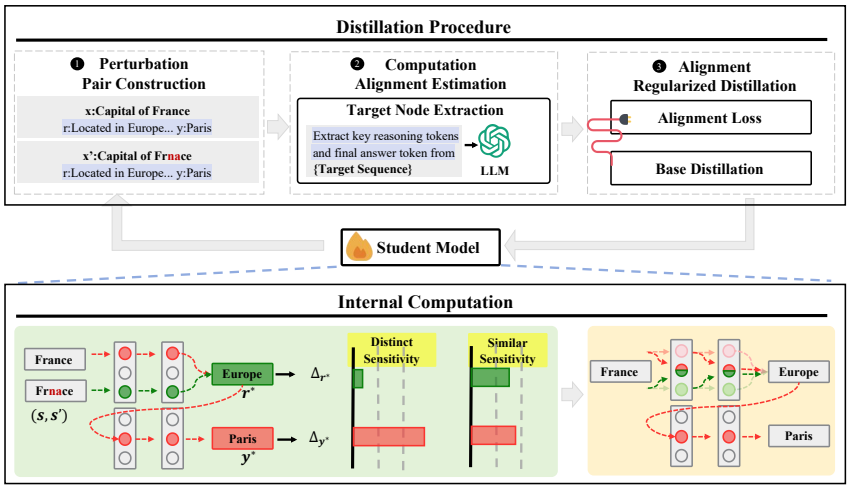


Figure 2: Top: *CAD* augments CoT distillation by constructing a perturbed pair (s, s') from the input query x, measuring the *sensitivity discrepancy* at the reasoning and answer circuit endpoints as a proxy for computation alignment, and using this proxy to regularize distillation. Bottom: the corresponding internal computation in the student model. Steps ❶ and ❷ perturb the circuit and probe its endpoint sensitivities, whereas Step ❸ pulls the answer circuit toward the reasoning circuit.

3.1.2 Computation Alignment

Building on multi-step circuit tracing (Section 3.1.1), which mechanistically characterizes the model's computation process, we now formally define computation alignment at the circuit level.

In the CoT setting, the model-generated output further consists of reasoning tokens r and final-answer tokens y, i.e., $o = (r, y)$. We use an LLM (Llama-3.3-70B in the paper) to identify a reasoning-critical token $r^* \in r$ and an answer-critical token $y^* \in y$ (prompt template in Appendix E.1).[2] We then trace the reasoning circuit $\mathcal{G}_{r^*}$ and the answer circuit $\mathcal{G}_{y^*}$ with MCT (Equation 7):

$$\mathcal{G}_c = (\mathcal{N}_c, \mathcal{E}_c, \mathcal{W}_c) = \mathrm{MCT}(x, c), \qquad c \in \{r^*, y^*\}. \tag{8}$$

Definition 1 (Computation Alignment)**.** *For a sample s, computation alignment is the overlap between its reasoning and answer circuits—the proportion of nodes shared by the two circuits:*

$$\mathcal{A}(s) = \frac{|\mathcal{N}_{r^*} \cap \mathcal{N}_{y^*}|}{|\mathcal{N}_{r^*} \cup \mathcal{N}_{y^*}|} \in [0, 1]. \tag{9}$$

A higher $\mathcal{A}(s)$ indicates that the circuit of the final answer and CoT steps are more overlap, computational process producing the final answer is more consistent with the CoT reasoning.

3.2 Distillation for Computation Alignment

Section 3.1 identifies computation alignment as the circuit overlap between CoT steps and answer generation. Directly optimizing this overlap during distillation is impractical because multi-step circuit tracing is prohibitively expensive inside the training loop. We therefore replace explicit circuit tracing with a lightweight proxy—the discrepancy between their sensitivities to the same controlled input perturbation—thereby avoiding circuit reconstruction during training.

The key intuition is that when CoT steps and answer generation rely on a highly overlapping computation circuit, the same input perturbation propagates through largely shared components and affects them similarly; conversely, when they rely on different computation circuits, their perturbation responses diverge accordingly.

[2]When multiple reasoning-critical tokens are identified, we compute the alignment for each token separately and report their average.

Building on this proxy, we propose *CAD* (**C**omputation-**A**ligned **D**istillation), a plug-in method that augments any existing CoT distillation objective with a single regularizer. As illustrated in Figure 2, *CAD* proceeds in three steps: (❶) constructing controlled input perturbations; (❷) measuring the student's perturbation sensitivity at the reasoning-token and answer-token positions separately and taking their discrepancy as the proxy; and (❸) minimizing this discrepancy jointly with the base distillation objective, so that the student's answer generation is computationally grounded in the steps described by the CoT. The detailed procedure is as follows.

3.2.1 Step ❶: Perturbation Pair Construction.

This step constructs the paired sequence required to estimate the sensitivity-discrepancy proxy in Step ❷. Given a teacher-generated training sequence $s = (x, r, y)$, we perturb the query x to obtain x' while keeping r and y fixed, yielding (s, s') where $s' = (x', r, y)$.

The perturbation should satisfy two competing constraints: (a) induce detectable hidden-state shifts at output-token positions, and (b) preserve query semantics such that the same reasoning path and answer remain valid, leaving the underlying computation circuits unchanged. We therefore adopt word-level typographical perturbation: for a small number of content words in x, exactly one character-level edit is applied per word—substitution, deletion, or transposition of adjacent characters (e.g., "France" → "Frnace").

3.2.2 Step ❷: Computation Alignment Estimation.

To approximate the overlap between the reasoning and answer circuits (Section 3.1.1) without explicit circuit tracing, we compare their responses to the same input perturbation as the proxy. Step ❶ applies a controlled perturbation to the input shared by both circuits. This step measures the resulting change in the last-layer hidden states of tokens r^* and y^*, the endpoints of the two circuits.

Let $h(c)$ and $h(c')$ denote the hidden states at position c in the original and perturbed sequences, respectively. We define the perturbation sensitivity at c as

$$\Delta_c(s) = \|h(c) - h(c')\|_2, \qquad c \in \{r^*, y^*\}. \tag{10}$$

When the two circuits overlap substantially, the perturbation propagates through more shared computation and thus induces similar responses at both endpoints. The lightweight proxy for the circuit-level computation alignment $\mathcal{A}(s)$ (Equation 9) can therefore be defined as the sensitivity-discrepancy:

$$\rho(s) = \frac{\Delta_{r^*}(s)}{\Delta_{y^*}(s)}. \tag{11}$$

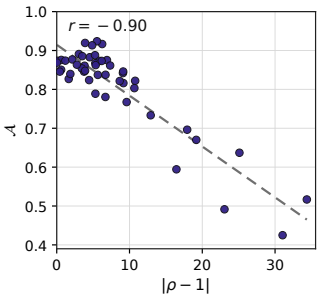


Figure 3: Relationship between sensitivity discrepancy and circuit-level computation alignment.

A value of $\rho(s)$ closer to 1 indicates more consistent responses between the reasoning and answer circuits, and therefore stronger computation alignment. We validate this proxy against circuit-level alignment $A(s)$ computed with pretrained Top-K transcoders for Llama-3.1-8B-Instruct (Zhao et al., 2026).[3] As shown in Figure 3, $|\rho(s) - 1|$ is strongly negatively correlated with $A(s)$ ($r = -0.90$), supporting it as a lightweight proxy for computation alignment.

3.2.3 Step ❸: Alignment-Regularized Distillation.

This step applies the computation alignment proxy $\rho(s)$ from Step ❷ to regularize the distillation objective. A value of $\rho(s)$ closer to 1 indicates stronger computation alignment, so we penalize the deviation of $\rho(s)$ from 1 with the computation alignment loss

$$\mathcal{L}_{\text{align}}(s) = \cosh\Big(\alpha\big[|\rho(s) - 1| - m\big]_+\Big) - 1, \tag{12}$$

where $[z]_+ = \max(z, 0)$, $m \geq 0$ is a tolerance threshold, and $\alpha > 0$ controls the growth rate of the penalty. The loss is zero when $|\rho(s) - 1| \leq m$ and otherwise increases monotonically as $\rho(s)$

[3] https://github.com/facebookresearch/CRV

deviates further from 1. Let $\mathcal{L}_{\mathrm{KD}}^{(\mathcal{D})}$ denote the training objective of an arbitrary base CoT distillation $\mathcal{D}$. *CAD* adds the alignment loss to this objective and minimizes

$$\mathcal{L}_{CAD} = \underbrace{\mathcal{L}_{\mathrm{KD}}^{(\mathcal{D})}}_{\text{base distillation}} + \underbrace{\lambda \mathcal{L}_{\mathrm{align}}}_{\text{computation alignment}} \tag{13}$$

where λ controls the strength of the regularization.

4 Experiments

4.1 Experimental Setup

Datasets. We use two widely adopted open-source datasets as the training data: GSM8K (mathematical reasoning) (Cobbe et al., 2021) and Boolean Expressions (relational reasoning) (Suzgun et al., 2023). For evaluation data, we select GSM8K, SVAMP (Patel et al., 2021), MultiArith (Roy & Roth, 2015), MAWPS (Koncel-Kedziorski et al., 2016), ASDiv (Miao et al., 2020), Boolformer (d'Ascoli et al., 2023), and DeepMind Formula Evaluation (Saxton et al., 2019), respectively. All evaluation sets are disjoint from the corresponding training data. Detailed descriptions of each dataset are provided in Appendix C.

Baselines. We instantiate the base CoT distillation method $\mathcal{D}$ with three representative methods: Classic CoT Distillation (Ho et al., 2023) and two state-of-the-art approaches, DFT (Wu et al., 2026) and GEM (Li et al., 2025). For each base method, we evaluate both its original objective and its *CAD*-augmented counterpart. Detailed descriptions of the baselines are provided in Appendix D.

Teacher and Student Models. We evaluate three teacher models: Llama-3.3-70B (Grattafiori et al., 2024), GPT-4.1-mini (OpenAI, 2025), and Grok-3-mini, and two student models: Qwen2.5-0.5B (Qwen et al., 2025) and Gemma-2-2B (Team et al., 2024).

Basic Distillation Setting. Unless specified otherwise, evaluations utilize this setting: Llama-3.3-70B as the teacher, Qwen2.5-0.5B as the student, Classic CoT Distillation as the base method, and GSM8K as the training and evaluation dataset.

Evaluation Metrics. We use accuracy, a widely adopted metric, to evaluate student model performance. It measures the proportion of predictions that correctly match the ground-truth answers.

Implementation Details. We use Llama-3.3-70B to extract reasoning-critical and answer-critical tokens from teacher-generated chains of thought. For each sample, we generate three independent perturbations and average the resulting sensitivity-based alignment estimates. Classic CoT Distillation, DFT, GEM, and their *CAD*-augmented variants use identical full-parameter fine-tuning settings. For each *CAD*-augmented variant, we perform a grid search over the tolerance threshold m, penalty strength α, and alignment loss weight λ. We select the hyperparameter configuration and checkpoint with the highest answer accuracy on the validation set. Baseline checkpoints are selected in the same way. We then evaluate all selected checkpoints on the evaluation set. We will publish the complete code after the paper is accepted. See Appendix E for complete training, search, and evaluation details.

4.2 Main Results

Computation Alignment and Reasoning Performance. To examine the relationship between computation alignment and reasoning performance, we analyze correct and incorrect predictions from the base model Llama-3.1-8B-Instruct. As shown in Figure 4 (a), correct predictions exhibit substantially smaller $|\rho - 1|$, indicating more consistent perturbation sensitivities between CoT reasoning and answer generation.

Figure 4 (b) further shows that correct samples concentrate near the diagonal $\Delta_{\mathrm{CoT}} = \Delta_{\mathrm{ans}}$, supporting ρ as a measure of their sensitivity discrepancy. Together, these results reveal a clear association between computation alignment and reasoning performance, motivating computation alignment as a meaningful optimization objective for CoT distillation.

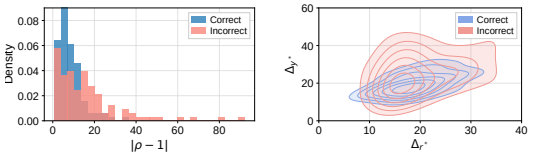
Table 1: Performance comparison across mathematical and relational reasoning benchmarks. Results are reported as accuracy (%). Blue cells denote results with our plug-in, while green cells report the corresponding absolute improvements (Δ). The best accuracy for each dataset is shown in **bold**.

Dataset	No Distill.	CoT Distillation			DFT			GEM		
		Base	+ Ours	Δ	Base	+ Ours	Δ	Base	+ Ours	Δ
GSM8K	24.73	18.94	38.50	+19.56	38.48	**42.47**	+3.63	19.55	38.33	+18.79
SVAMP	42.00	33.50	59.00	+25.50	56.00	**63.00**	+7.00	34.50	56.00	+21.50
MultiArith	38.33	52.78	**87.20**	+34.42	83.33	86.67	+3.33	44.44	85.00	+40.56
MAWPS	50.45	44.86	78.40	+33.54	75.96	**79.91**	+3.95	54.91	73.02	+18.12
ASDiv	49.27	39.06	64.00	+24.94	64.45	**65.39**	+0.94	47.53	64.23	+16.70
BoolExp	0.40	91.60	91.20	-0.40	60.40	91.60	+31.20	92.40	**94.00**	+1.60
Boolformer	0.75	84.40	85.30	+0.90	59.75	**87.35**	+27.60	84.10	86.80	+2.70
DeepMind	0.35	83.20	85.65	+2.45	58.40	**87.20**	+28.80	84.30	85.50	+1.20

Table 2: Performance comparison across different teacher models. Results are reported as accuracy (%). Blue cells denote results with our plug-in, while green cells report the corresponding absolute improvements (Δ). The best accuracy for each teacher is shown in **bold**.

Teacher	No Distill.	CoT Distillation			DFT			GEM		
		Base	+ Ours	Δ	Base	+ Ours	Δ	Base	+ Ours	Δ
Llama-3.3-70B	24.73	18.94	38.50	+19.56	38.48	**42.27**	+3.79	19.55	38.33	+18.79
GPT-4.1-mini	24.73	27.58	**42.73**	+15.15	28.79	40.90	+12.11	23.64	41.82	+18.18
Grok-3-mini	24.73	11.97	22.12	+10.15	27.73	**29.39**	+2.62	13.48	18.94	+5.46

Impact of Training Dataset. We evaluate *CAD* on GSM8K (mathematical reasoning) and BoolExp (relational reasoning) as training datasets, with all other configurations following the *Basic Distillation Setting* in Section 4.1.



(a) Alignment distribution. (b) Perturbation sensitivity.

Figure 4: Computation alignment and reasoning performance.

CAD yields consistent improvements across both datasets and all three distillation methods, with the most pronounced gain in DFT on BoolExp (+31.20%). Notably, Classic CoT distillation on GSM8K degrades below the No Distillation baseline, consistent with prior observations that long CoT supervision can impair small-model performance through error accumulation (Luo et al., 2025). *CAD* effectively reverses this degradation, boosting CoT accuracy well above the undistilled baseline.

Impact of Evaluation Dataset. To further assess within-domain transferability, we evaluate models trained on GSM8K against four held-out mathematical benchmarks (SVAMP, MultiArith, MAWPS, ASDiv) and models trained on BoolExp against two relational reasoning benchmarks (Boolformer, DM-FE), with all other configurations following the *Basic Distillation Setting*.

CAD improves all base methods on every out-of-distribution benchmark, with gains up to +40.56% on MultiArith (Table 1). This indicates that the improvements from *CAD* are not overfitted to the training distribution but generalize to related tasks. Moreover, CoT+*CAD* on SVAMP (59.00%) surpasses both base DFT (56.00%) and base GEM (34.50%), showing that *CAD* can amplify even simple distillation pipelines beyond more sophisticated alternatives without overfitting to the training distribution.

Table 3: Performance comparison across different student models. Results are reported as accuracy (%). Blue cells denote results with our plug-in, while green cells report the corresponding absolute improvements (Δ). The best accuracy for each student is shown in **bold**.

Student	No Distill.	CoT Distillation			DFT			GEM		
		Base	+ Ours	Δ	Base	+ Ours	Δ	Base	+ Ours	Δ
Qwen2.5-0.5B	24.73	18.94	38.50	+19.56	38.48	**42.27**	+3.79	19.55	38.33	+18.79
Gemma-2-2B	0.91	36.06	**47.27**	+11.21	7.88	11.06	+3.18	14.70	39.39	+24.70

Impact of Teacher Models. We evaluate *CAD* across different teacher models with all other configurations following the *Basic Distillation Setting* in Section 4.1. As shown in Table 2, *CAD* yields consistent improvements across all three teachers. However, Grok-3-mini exhibits notably smaller gains (e.g., DFT: +2.62%, GEM: +5.46%) compared to GPT-4.1-mini and Llama-3.3-70B, indicating that a weaker teacher limits *CAD*'s performance ceiling.

This is expected: since *CAD* aligns the student's internal reasoning with teacher-generated CoT steps, the quality of those chains directly bounds the achievable corrective effect.

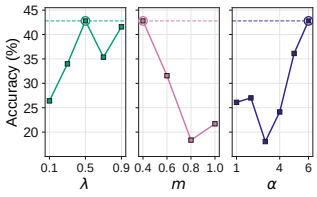


Figure 5: Sensitivity of *CAD* to λ, m, and α.

Impact of Student Models. We evaluate *CAD* across different student models with all other configurations following the *Basic Distillation Setting* in Section 4.1.

As shown in Table 3, *CAD* yields the largest gain across all experiments on Gemma-2-2B with GEM (+24.70%). Notably, despite Gemma-2-2B's near-zero No Distillation baseline (0.91%), CoT+*CAD* reaches 47.27%—the highest accuracy across all configurations.

4.3 Ablation Study and Analysis

All ablations follow the *Basic Distillation Setting* (Section 4.1) with Classic CoT Distillation as the base method.

Hyperparameter Sensitivity. The ablation result of the hyperparameter sensitivity is shown in Figure 5. A moderate λ performs best, as an overly strong alignment objective can interfere with the base distillation objective. Smaller m consistently performs better because it penalizes larger sensitivity discrepancies more strictly, while larger α strengthens the correction of clearly misaligned samples. Overall, *CAD* performs best with a moderate regularization weight, a tight tolerance, and a sufficiently strong penalty.

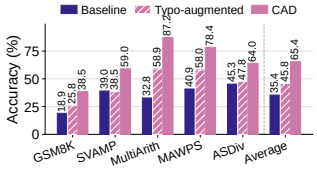


Figure 6: Typo augmentation ablation.

Role of Perturbation Robustness. To test whether *CAD* merely improves robustness to input perturbations, we compare it with typo augmentation without the alignment loss. As shown in Figure 6, typo augmentation improves average accuracy from 35.4% to 45.8%, whereas *CAD* reaches 65.4% and outperforms it on every benchmark. The difference is particularly clear on SVAMP: typo augmentation provides little improvement, while *CAD* improves 20%. This substantial gap confirms that the gains come more from computation alignment than from perturbation robustness.

5 Related Work

CoT Prompting for Improved Reasoning. Chain-of-Thought (COT) prompting was introduced by (Wei et al., 2022) to elicit more complex reasoning from LLMs through few-shot examples that demonstrate step-by-step problem solving. Subsequent work has expanded this idea in several directions. (Kojima et al., 2022) showed that even a simple zero-shot instruction such as "Let's think step

by step" can trigger coherent reasoning traces. While this reduces the need for handcrafted prompts, providing structured examples often remains beneficial. To scale this process, recent studies generate CoT exemplars synthetically ((Zhang et al., 2022; Shao et al., 2023; Chen et al., 2026)). Other work leverages test-time compute scaling to extend reasoning chains, enabling longer and more elaborate solutions (Snell et al., 2024). For comprehensive surveys of CoT techniques and their applications, see (Chu et al., 2024) and (Chen et al., 2026).

Circuit tracing for chain-of-thought. Lindsey et al. (2025) applied circuit tracing to chain-of-thought (CoT), using attribution graphs to uncover latent multi-hop computation and cases where verbalized reasoning diverges from the model's internal computation. CRV (Zhao et al., 2026) extended this direction by constructing attribution graphs for individual reasoning steps and learning structural signatures that distinguish correct from incorrect computation. CIE-Scorer (Shen et al., 2026) further traced compact sentence-level circuits and detected unfaithful CoTs by measuring discrepancies between internal computational graphs and external reasoning graphs. Despite this progress, these approaches inherit a key limitation of current circuit-tracing techniques: attribution is typically anchored to a selected output position or local reasoning span. Consequently, it remains difficult to trace computation across multiple successive reasoning steps and recover a unified computational pathway from the input query to the final answer.

CoT Distillation. Early CoT distillation methods use large teachers to generate reasoning traces and directly fine-tune smaller students on these sequences (Ho et al., 2023; Magister et al., 2023), or incorporate teacher-generated rationales as auxiliary supervision alongside the task objective (Hsieh et al., 2023). Subsequent work further refines the supervision signal, for example by disentangling symbolic structure from surface text (Li et al., 2023) and prioritizing critical tokens through progressive distillation (Feng et al., 2024). More recent studies seek to internalize explicit CoT into hidden or continuous representations. Deng et al. (2023) distill explicit reasoning steps into intermediate hidden states, followed by curriculum-based internalization that progressively removes explicit reasoning steps (Deng et al., 2024), while other work performs reasoning directly in continuous latent space (Hao et al., 2024). CODI further compresses natural-language CoT into continuous hidden representations through self-distillation (Shen et al., 2025). Despite these different formulations, they share a common objective: making the student's reasoning content, structure, or representation consistent with the teacher's CoT, which we refer to as teacher–student CoT alignment.

However, teacher–student CoT alignment does not ensure that the student's CoT reflects the computation that produces its answer. Prior work shows that generated reasoning may omit or misrepresent the factors that influence model predictions, and that modifying the CoT can leave the answer largely unchanged (Turpin et al., 2023; Lanham et al., 2023). An explicit CoT therefore cannot be equated with the internal computation behind the answer. Existing CoT distillation methods leave this gap unaddressed.

We address this gap with computation alignment, which requires the student's CoT to be consistent with the internal computation underlying its answer. Instead of comparing teacher and student CoTs, we compare the circuits underlying CoT reasoning and answer generation within the student model. We then quantify their consistency and convert it into the trainable *CAD* objective.

6 Conclusion

In this work, we identify computation alignment as a missing constraint in existing CoT distillation, requiring the student's answer generation to be computationally grounded in the problem-solving steps described by its CoT. To characterize this property, we extend conventional single-step circuit tracing to multi-step circuit tracing, and further develop a lightweight perturbation-based proxy that makes computation alignment practical despite the high cost of circuit reconstruction. Beyond distillation, we believe computation alignment offers a more general principle for improving the monitorability of CoT by encouraging its verbalized reasoning to reflect the computation actually used to produce the answer. This opens opportunities to incorporate computation-aware constraints into broader training paradigms, including distillation and fine-tuning, and we hope our work provides a useful foundation for researchers and practitioners pursuing more reliable and interpretable reasoning models.

AI USE STATEMENT

We used generative AI tools to assist with language editing and translation, manuscript drafting and refinement, and the presentation and interpretation of experimental results. All AI-assisted content and analyses were carefully reviewed and verified by the authors. The authors take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We provide the details necessary to reproduce our results in the appendix and supplementary materials, including model and dataset configurations, training hyperparameters, implementation details of multi-step circuit tracing and computation-aligned distillation, prompt templates, and evaluation protocols. We will publish the complete code after the paper is accepted

REFERENCES

Emmanuel Ameisen, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L. Turner, Brian Chen, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. Circuit tracing: Revealing computational graphs in language models. *Transformer Circuits Thread*, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/methods.html.

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *Science China Information Sciences*, 69(6):161101, 2026.

Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. Navigate through enigmatic labyrinth a survey of chain of thought reasoning: Advances, frontiers and future. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1173–1203, 2024.

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.

Xinnan Dai, Chung-Hsiang Lo, Kai Guo, Shenglai Zeng, Dongsheng Luo, and Jiliang Tang. Uncovering graph reasoning in decoder-only transformers with circuit tracing, 2025. URL https://arxiv.org/abs/2509.20336.

Stéphane d'Ascoli, Arthur Renard, Vassilis Papadopoulos, Samy Bengio, Josh Susskind, and Emmanuel Abbé. Boolformer: Symbolic regression of logic functions with transformers. *arXiv preprint arXiv:2309.12207*, 2023.

Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*, 2023.

Yuntian Deng, Yejin Choi, and Stuart Shieber. From explicit cot to implicit cot: Learning to internalize cot step by step. *arXiv preprint arXiv:2405.14838*, 2024.

Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. Transcoders find interpretable llm feature circuits. *Advances in Neural Information Processing Systems*, 37:24375–24410, 2024.

Kaituo Feng, Changsheng Li, Xiaolu Zhang, Jun Zhou, Ye Yuan, and Guoren Wang. Keypoint-based progressive chain-of-thought distillation for llms. *arXiv preprint arXiv:2405.16064*, 2024.

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.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 8154–8173, 2023.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

Namgyu Ho, Laura Schmid, and Se-Young Yun. Large language models are reasoning teachers. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 14852–14882, 2023.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the association for computational linguistics: ACL 2023*, pp. 8003–8017, 2023.

Harish Kamath, Emmanuel Ameisen, Isaac Kauvar, Rodrigo Luger, Wes Gurnee, Adam Pearce, Sam Zimmerman, Joshua Batson, Thomas Conerly, Chris Olah, and Jack Lindsey. Tracing attention computation through feature interactions. *Transformer Circuits Thread*, 2025. URL https://transformer-circuits.pub/2025/attention-qk/index.html.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. Mawps: A math word problem repository. In *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: human language technologies*, pp. 1152–1157, 2016.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, et al. Measuring faithfulness in chain-of-thought reasoning, july 2023. *URL http://arxiv. org/abs/2307.13702*, 2023.

Liunian Harold Li, Jack Hessel, Youngjae Yu, Xiang Ren, Kai-Wei Chang, and Yejin Choi. Symbolic chain-of-thought distillation: Small models can also "think" step-by-step. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2665–2679, 2023.

Ziniu Li, Congliang Chen, Tian Xu, Zeyu Qin, Jiancong Xiao, Zhi-Quan Luo, and Ruoyu Sun. Preserving diversity in supervised fine-tuning of large language models. In *International Conference on Learning Representations*, volume 2025, pp. 66127–66154, 2025.

Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. On the biology of a large language model. *Transformer Circuits Thread*, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/biology.html.

Renjie Luo, Jiaxi Li, Chen Huang, and Wei Lu. Through the valley: Path to effective long cot training for small language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 4972–4992, 2025.

Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. Teaching small language models to reason. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1773–1781, 2023.

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.

OpenAI. Introducing GPT-4.1 in the API. https://openai.com/index/gpt-4-1/, April 2025.

Vedant Palit, Florent Draye, Terry Jingchen Zhang, Bernhard Schölkopf, and Zhijing Jin. How do linear probes emerge? a circuit-tracing framework with concept-targeted attribution. *arXiv preprint arXiv:2608.27510*, 2026.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? In *Proceedings of the 2021 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 2080–2094, 2021.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi 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,

Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pp. 1743–1752, 2015.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*, 2019.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Synthetic prompting: Generating chain-of-thought demonstrations for large language models. In *International conference on machine learning*, pp. 30706–30775. PMLR, 2023.

Xu Shen, Zhen Tan, Song Wang, Pingjun Hong, Rui Miao, Xin Wang, and Tianlong Chen. Detecting unfaithful chain-of-thought via circuit-guided internal-external discrepancy. *arXiv preprint arXiv:2605.25603*, 2026.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. Codi: Compressing chain-of-thought into continuous space via self-distillation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 677–693, 2025.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965, 2023.

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.

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.

Yongliang Wu, Yizhou Zhou, Ziheng Zhou, Yingzhe Peng, Xinyu Ye, Xinting Hu, Wenbo Zhu, Lu Qi, Ming-Hsuan Yang, et al. On the generalization of sft: A reinforcement learning perspective with reward rectification. In *International Conference on Learning Representations*, volume 2026, pp. 27550–27571, 2026.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*, 2022.

Zheng Zhao, Yeskendir Koishekenov, Xianjun Yang, Naila Murray, and Nicola Cancedda. Verifying chain-of-thought reasoning via its computational graph. In *International Conference on Learning Representations*, volume 2026, pp. 122812–122839, 2026.

Xianwei Zhuang, Zhihong Zhu, Zhichang Wang, Xuxin Cheng, and Yuexian Zou. Unicott: A unified framework for structural chain-of-thought distillation. In *International Conference on Learning Representations*, volume 2025, pp. 81314–81338, 2025.

A Notation of Symbols

Table 4: Notation of symbols used in the paper.

Notation	Description
x	The input query
o	The model-generated output sequence
$s = (x, o)$	The sequence under analysis
r	The reasoning tokens in a CoT output
y	The final-answer tokens in a CoT output
r^*	A reasoning-critical token selected from r
y^*	An answer-critical token selected from y
s'	The perturbed counterpart of s
$\mathrm{SCT}(\cdot)$	The single-step circuit tracing operator
$\mathrm{MCT}(\cdot)$	The multi-step circuit tracing operator
$\mathcal{A}(x, o)$	The computation alignment between reasoning and answer circuits

B Circuit Tracing

Algorithm 1 Single-Step Circuit Tracing $\mathrm{SCT}_{\mathcal{M}}(s_t,\, \ell_t)$

Require: Target model $\mathcal{M}$; autoregressive context $s_t = (x,\, o_{<t})$; prediction logit ℓ_t for output token o_t; top-k parameter k
Ensure: Local attribution graph $G_t = (\mathcal{N}_t,\, \mathcal{E}_t,\, W_t)$

$\mathcal{N}_t \leftarrow \{\ell_t\},\ \mathcal{E}_t \leftarrow \varnothing,\ W_t \leftarrow \varnothing$ ▷ Initialise graph
$\mathcal{F} \leftarrow \{\ell_t\}$ ▷ Attribution frontier
while $\mathcal{F} \neq \varnothing$ **do**
  Pop an attribution node v from $\mathcal{F}$
  for each candidate source u at preceding layers **do** ▷ CLT features **or** context tokens in s_t
    Compute attribution weight: $w_{uv} \leftarrow f_u \cdot e_v^{\top}\, d_u$ (Eq. 1)
  end for
  $\mathcal{S} \leftarrow \text{Top-k}\big(\{u : |w_{uv}|\},\ k\big)$
  for each $u \in \mathcal{S}$ **do**
    $\mathcal{N}_t \leftarrow \mathcal{N}_t \cup \{u\}$
    $\mathcal{E}_t \leftarrow \mathcal{E}_t \cup \{(u, v)\}$
    $W_t(u, v) \leftarrow w_{uv}$
    if u is a CLT feature node **then**
      $\mathcal{F} \leftarrow \mathcal{F} \cup \{u\}$ ▷ Continue tracing
    end if ▷ Context tokens are leaves; do not expand
  end for
end while
return $G_t = (\mathcal{N}_t,\, \mathcal{E}_t,\, W_t)$

C Datasets

Here we provide detailed descriptions of the datasets used in this work.

GSM8K (Cobbe et al., 2021). GSM8K is a dataset of 8.5K linguistically diverse grade-school mathematics word problems that require multi-step arithmetic reasoning. We use its official training split as arithmetic training data and its non-overlapping test split for evaluation.

SVAMP (Patel et al., 2021). SVAMP (Simple Variations on Arithmetic Math Word Problems) is a challenge set of elementary arithmetic word problems constructed by applying controlled variations

Algorithm 2 Multi-Step Circuit Tracing $\mathrm{MCT}_{\mathcal{M}}(x,\, o_t)$

Require: Target model $\mathcal{M}$; input query x; generated output $o = (o_1, \ldots, o_t)$; pruning threshold τ; cumulative influence budget $\Phi_{\max}$
Ensure: Global attribution graph $G = (\mathcal{N},\, \mathcal{E},\, W)$

$\mathcal{N} \leftarrow \varnothing,\ \mathcal{E} \leftarrow \varnothing,\ W \leftarrow \varnothing$ ▷ Initialise global graph
$\mathcal{Q} \leftarrow \{o_t\}$ ▷ Priority queue (frontier)
while $\mathcal{Q} \neq \varnothing$ **and** cumulative influence $< \Phi_{\max}$ **do**
  % ——— *(i) Expand* ———
  Pop the highest-priority token o_i from $\mathcal{Q}$
  $G_i = (\mathcal{N}_i,\, \mathcal{E}_i,\, W_i) \leftarrow \mathrm{SCT}_{\mathcal{M}}(s_i,\, \ell_i)$, where $s_i = (x,\, o_{<i})$ (Eq. 3)
  % ——— *(ii) Merge* ———
  if $i = t$ **then** ▷ Initial iteration
    $G \leftarrow G_t$
  else
    $\mathcal{N} \leftarrow \big(\mathcal{N} \setminus \{o_i\}\big) \cup \big(\mathcal{N}_i \setminus \{\ell_i\}\big)$ ▷ Replace leaf o_i (Eq. 4)
    for each $(u, \ell_i) \in \mathcal{E}_i$ **and** $(o_i, v) \in \mathcal{E}$ **do**
      Remove (o_i, v) from $\mathcal{E}$; insert (u, v) into $\mathcal{E}$
      $W(u, v) \leftarrow W_i(u, \ell_i) \cdot W(o_i, v)$ ▷ Chain rule (Eq. 5)
    end for
  end if
  % ——— *(iii) Prune* ———
  Compute row-normalised absolute weight matrix $\widetilde{W}$
  for each node $u \in \mathcal{N}$ **do**
    $\phi(u; \ell_t) \leftarrow \big[(I - \widetilde{W})^{-1} - I\big]_{u,\, \ell_t}$ (Eq. 6)
    if $\phi(u; \ell_t) < \tau$ **then**
      Remove u and its incident edges from G
    end if
  end for
  for each surviving leaf token o_j $(j < i)$ not yet expanded **do**
    Insert o_j into $\mathcal{Q}$ with priority $\phi(o_j; \ell_t)$
  end for
end while
return $G = (\mathcal{N},\, \mathcal{E},\, W)$

to existing problems. The variations alter quantities, question structure, or required operations while preserving the elementary mathematical setting. We use SVAMP only as a test set.

MultiArith (Roy & Roth, 2015). MultiArith is a collection of arithmetic word problems that require composing multiple arithmetic operations to obtain the final answer. We use MultiArith only as a test set.

MAWPS (Koncel-Kedziorski et al., 2016). MAWPS (Math Word Problem Repository) is a repository that consolidates arithmetic word problems from multiple existing collections into a standardized resource. It contains diverse natural-language problem formulations and arithmetic expressions. We use MAWPS only as a test set.

ASDiv (Miao et al., 2020). ASDiv (Academia Sinica Diverse Math Word Problem Dataset) is an English mathematics word-problem corpus designed to increase diversity in both linguistic patterns and problem types. It contains 2,305 elementary-school mathematics problems annotated with problem types and grade levels. We use ASDiv only as a test set.

Boolean Expressions (Suzgun et al., 2023). Boolean Expressions is a logical-reasoning task included in BIG-Bench Hard in which Boolean expressions composed of logical operators must be evaluated to determine their truth values. Following its expression grammar, we construct a Boolean-expression corpus used as the Boolean reasoning training set.

Boolformer (d'Ascoli et al., 2023). Boolformer studies symbolic representations of Boolean functions and provides systematically generated Boolean expressions with their corresponding evalua-

tions. We adapt its Boolean-expression generation setting to construct a formula-evaluation test set that is disjoint from our Boolean training corpus.

DeepMind Formula Evaluation (Saxton et al., 2019). The DeepMind Mathematics Dataset is a procedurally generated collection of mathematical question–answer pairs covering arithmetic, algebra, polynomials, calculus, probability, and related reasoning tasks. We use its formula-evaluation problems, which require evaluating symbolic mathematical expressions or functions under given variable assignments, as a test set.

D Baseline Details

Classic CoT Distillation. (Ho et al., 2023) This method is the standard CoT distillation paradigm of Fine-tune-CoT, where teacher-generated reasoning steps and final answers are used as supervision to fine-tune the student model with the standard cross-entropy objective.

DFT(Wu et al., 2026). Dynamic Fine-Tuning (DFT) reinterprets SFT from a reinforcement-learning perspective and dynamically rescales each target token according to its current model probability. This reduces excessive updates on low-probability target tokens and improves generalization.

GEM(Li et al., 2025). This method introduces a game-theoretic SFT objective connected to reverse-KL minimization with entropy regularization. It reduces over-memorization of training examples while preserving diversity in the model's output distribution.

E Implementation Details

Table 5 summarizes the settings shared by all methods. Each training run uses one NVIDIA RTX PRO 6000 Blackwell GPU with 96 GB of memory.

Table 5: Training configurations used in the experiments.

Hyperparameter	**Value**
Number of epochs	3
Micro-batch size	16
Gradient accumulation steps	4
Effective batch size	64
Optimizer	AdamW
Learning rate	2×10^{-5}
Adam β_1	0.9
Adam β_2	0.999
Adam ϵ	10^{-8}
Weight decay	0.01
Learning-rate scheduler	Cosine
Learning-rate warmup ratio	0.05
Maximum sequence length	2048
Maximum gradient norm	1.0
Random seed	42
Gradient checkpointing	Enabled
Training dtype	bfloat16

Table 6: *CAD* hyperparameter search spacess.

Hyperparameter	Classic	DFT	GEM
λ	$\{0.1, 0.5, 0.9\}$	$\{0.001, 0.003, 0.01, 0.03, 0.1, 0.3\}$	$\{0.02, 0.05, 0.1, 0.2\}$
m	$\{0.4, 0.6, 0.8, 1.0\}$	$\{0.1, 0.2, 0.4\}$	$\{0.1, 0.2, 0.4\}$
α	$\{2, 4, 6\}$	$\{0.25, 0.5, 2.0, 4.0\}$	$\{0.5, 1.0, 2.0, 4.0, 6.0\}$

E.1 Prompt Templates

CoT generation. We use the following prompts for GSM8K and BoolExp, respectively.

GSM8K: CoT generation

```
System message:
Solve the following math problem step by step.

User message:
[QUESTION]
```

BoolExp: CoT generation

```
System message:
You are a precise formal-logic solver. Evaluate the Boolean expression
    using the precedence order parentheses, not, and, or. Show a short,
     deterministic bottom-up reduction trace. Before every newly
    derived Boolean result, briefly state which subexpression is being
    evaluated. Then put that result alone on its own line in exactly
    the form 'Node k: True' or 'Node k: False', numbering nodes
    consecutively. Do not use a Node line for a True/False constant
    merely copied from the question. End with exactly one line 'Final
    answer: True' or 'Final answer: False'.

User message:
[QUESTION]
```

Critical Token extraction. Each extracted token contains a `value` and a `context`, both copied verbatim from the supplied CoT. The context supports locating the intended occurrence of a repeated value. The field name `key_tokens` refers to extracted text spans, which need not correspond to single tokenizer tokens. The final answer is returned separately using the same format.

GSM8K

```
You are a precise math reasoning analyst.
Given a math problem and its step-by-step solution, extract the
    important
intermediate mathematical results and the final answer.

RULES:
1. key_tokens are important derived RESULTS (numeric or symbolic), not
    values
   or expressions merely copied from the problem.
2. For each key token, value must be the exact substring appearing
   in the solution.
3. context must be an EXACT verbatim substring from the solution
    containing
   that value. Do not paraphrase it.
4. final_answer must directly answer the problem.
```

```
5. final_answer.value must appear inside final_answer.context.
6. Order key_tokens by their appearance in the solution.
7. Do not mechanically select the last number in the solution.
8. If a line begins with "Final answer:", its value is the
    authoritative answer.
9. Return JSON only.

Problem:
"""
[QUESTION]
"""

Solution:
"""
[COT]
"""

Return exactly:
{
  "key_tokens": [
    {"value": "...", "context": "..."}
  ],
  "final_answer": {"value": "...", "context": "..."}
}
```

BoolExp

```
You are a precise Boolean reasoning-node extractor. Extract the atomic
    Boolean
results from the supplied formal reasoning trace and its final answer.

FEW-SHOT EXAMPLE
Problem:
"""
Evaluate this Boolean expression: not ( False or True ) and True
"""
Solution:
"""
Evaluate the parenthesized expression `False or True`.
Node 1: True
Apply `not` to that result.
Node 2: False
Evaluate `False and True`.
Node 3: False
Final answer: False
"""
Correct extraction:
{
  "key_tokens": [
    {"value": "True", "context": "Node 1: True"},
    {"value": "False", "context": "Node 2: False"},
    {"value": "False", "context": "Node 3: False"}
  ],
  "final_answer": {
    "value": "False",
    "context": "Final answer: False"
  }
}

RULES FOR THE TARGET:
1. Each key_tokens.value must be exactly the single token "True" or "
    False".
```

```
2. Extract only newly derived results on explicit "Node k:" lines.
    Never
   extract True/False constants copied from the problem or explanatory
    lines.
3. context must be the complete exact "Node k: True|False" line from
    the
   solution and must contain value verbatim.
4. Preserve all Node lines in their original order, including repeated
    values.
5. final_answer.value must be the exact True/False token on the
    authoritative
   "Final answer:" line, and context must be that exact complete line.
6. Return at least one key token and JSON only. Do not include
    commentary.

TARGET PROBLEM:
"""
[QUESTION]
"""

TARGET SOLUTION:
"""
[COT]
"""

Return exactly:
{
  "key_tokens": [
    {"value": "True", "context": "Node 1: True"}
  ],
  "final_answer": {
    "value": "False",
    "context": "Final answer: False"
  }
}
```