Reasoning Fails Where Step Flow Breaks

Xiaoyu Xu[1,2], Yulan Pan[2], Xiaosong Yuan[3‡], Zhihong Shen[2], Minghao Su[2], Yuanhao Su[2], Xiaofeng Zhang[1†]

[1]Shanghai Jiao Tong University, [2]Fuzhou University, [3]Jilin University

framebreak@sjtu.edu.cn

Abstract

Large reasoning models (LRMs) that generate long chains of thought now perform well on multi-step math, science, and coding tasks. However, their behavior is still unstable and hard to interpret, and existing analysis tools struggle with such long, structured reasoning traces. We introduce *Step-Saliency*, which pools attention–gradient scores into step-to-step maps along the question–thinking–summary trajectory. Across several models, Step-Saliency reveals two recurring information-flow failures: **Shallow Lock-in**, where shallow layers over-focus on the current step and barely use earlier context, and **Deep Decay**, where deep layers gradually lose saliency on the thinking segment and the summary increasingly attends to itself and the last few steps. Motivated by these patterns, we propose *StepFlow*, a saliency-inspired test-time intervention that adjusts shallow saliency patterns measured by Step-Saliency via Odds-Equal Bridge and adds a small step-level residual in deep layers via Step Momentum Injection. StepFlow improves accuracy on math, science, and coding tasks across multiple LRMs without retraining, indicating that repairing information flow can recover part of their missing reasoning performance. Code is available at https://github.com/XiaoyuXu-Vincent/step-saliency.

1 Introduction

Large reasoning models (LRMs) generate an internal thinking segment before producing a final summary, forming a question–thinking–summary trajectory. These models achieve strong performance on multi-step mathematics, code generation, and scientific question answering (Ahn et al., 2024; Plaat et al., 2025; Xu et al., 2025). Despite these gains, LRMs still exhibit persistent failure modes, including unfaithful chains of thought (Paul et al., 2024; Lightman et al., 2023; Barez et al., 2025), overconfident hallucinations (Fan et al., 2025; Cao et al., 2025), and brittleness on compositional tasks (Song et al., 2025; Li et al., 2025c; Boye and Moell, 2025; Simhi et al.; Chen et al., 2025b). When an LRM makes a mistake, we lack a clear way to attribute the final error to the model's internal reasoning trace. This motivates a diagnostic that tracks step-to-step influence across depth and links shifts in that influence to incorrect final answers (Luo and Specia, 2024; Cambria et al., 2024; Yang et al.).

Prior work offers several token-level diagnostics for inspecting LRMs during generation. Attention-based analyses visualize which tokens receive attention when producing later tokens (Vig and Belinkov, 2019; Yeh et al., 2023), but attention weights are not always faithful indicators of what actually drives the prediction (Jain and Wallace, 2019). Saliency-based methods instead assign gradient-weighted importance to earlier tokens (Ancona et al., 2017; Wu et al., 2023), yet along long reasoning traces the scores can be noisy and difficult to aggregate across positions. The main difficulty is not the lack of signals but the lack of a readable unit aligned with reasoning steps: token-level maps are dense, local, and do not naturally summarize step-to-step dependence. As Figure 1 shows, even when correct and error traces differ, the signal is dispersed across many tokens, making it difficult to summarize which steps remain influential later and how this changes with depth.

To address this limitation, we introduce **Step-Saliency**, a step-level diagnostic for long-form reasoning. We compute token saliency using attention–gradient influence and then pool the scores within each step. This yields a compact step→step map that is easy to compare across layers. Applying Step-Saliency to several LRMs, we observe two common patterns in incorrect outputs. In shallow layers, influence concentrates on the current think-

† Corresponding author,‡ Project leader

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 26337–26353
July 2-7, 2026

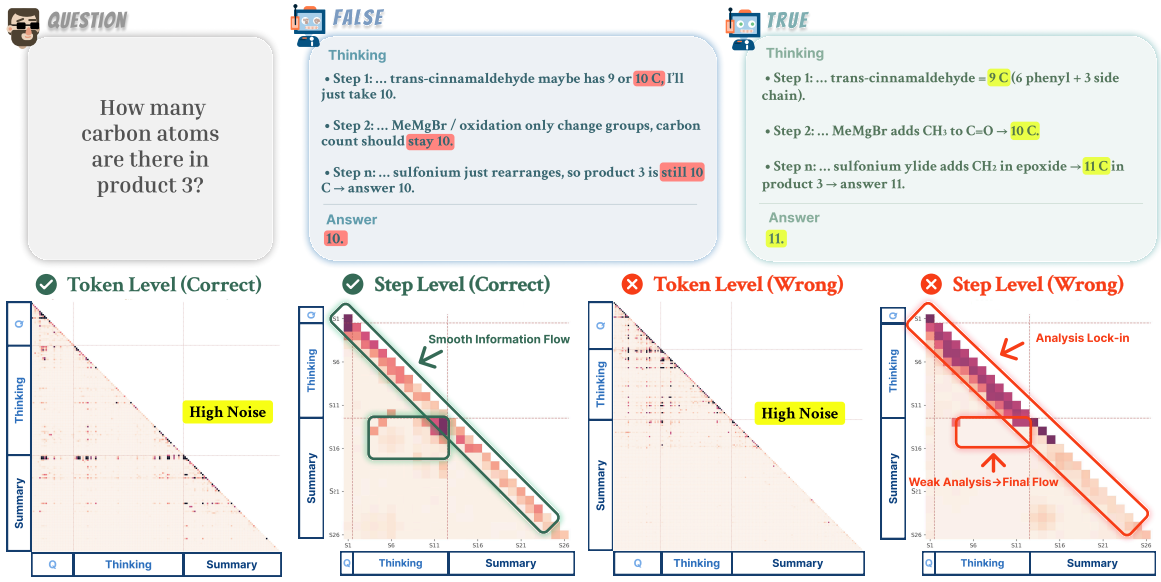

Figure 1: **From token- to step-level saliency.** Token-level saliency maps are dense and noisy; Step-Saliency pools them into question/thinking/summary blocks. Correct traces show smooth question→thinking→summary flow, while errors exhibit shallow lock-in and weak thinking→summary links.

ing step and its immediate neighbors, while earlier context is suppressed; we call this **Shallow Lock-in**. As depth increases, saliency on the thinking segment weakens for both correct and incorrect outputs, but it fades much faster on incorrect ones, and the summary becomes more self-focused; we call this **Deep Decay** (§3.2).

Building on these observations, we propose **StepFlow**, a lightweight test-time intervention that modifies a model's forward pass without retraining or backpropagation. StepFlow has two components that target the two failure patterns. **Odds-Equal Bridge (OEB)** is applied to the first few layers: when the model is generating a thinking step, it prevents step-level mass from collapsing onto the just-written tokens by keeping a non-trivial share on the question and earlier steps. **Step Momentum Injection (SMI)** is applied to the last few layers: at step boundaries, it carries a small residual summary of the previous step into the next step, so earlier reasoning remains available when the summary is produced. Together, OEB and SMI promote a steadier question–thinking–summary linkage during generation.

We evaluate StepFlow on DeepSeek-R1-Distill (7B/14B/32B), GPT-OSS-20B, and QwQ-32B-Preview across six benchmarks: AIME24, AIME25, AMC23, MATH-500, GPQA-Diamond, and LiveCodeBench. StepFlow consistently improves accuracy across all backbones, with the largest gains on competition-style problems that require long reasoning chains. Our main contributions are as follows:

- We introduce **Step-Saliency**, a diagnostic that aggregates token-level saliency into step→step maps, making long reasoning traces interpretable at the step level.
- We identify two depth-wise information-flow failure patterns in large reasoning models, which we term **Shallow Lock-in** and **Deep Decay**; they reliably separate incorrect from correct traces in our saliency analysis.
- We propose **StepFlow**, a test-time intervention that reshapes information flow via Odds-Equal Bridge and Step Momentum Injection, improving accuracy across multiple benchmarks without retraining.

2 Related Work

2.1 Large Reasoning Models

Chain-of-thought prompting (CoT) (Wei et al., 2022) substantially improved LLM reasoning and spurred *Large Reasoning Models* (LRMs) with strong multi-step reasoning and structured intermediate traces. Although LRMs demonstrate strong performance on multi-step reasoning benchmarks (Kojima et al., 2022; Chen et al., 2025a;

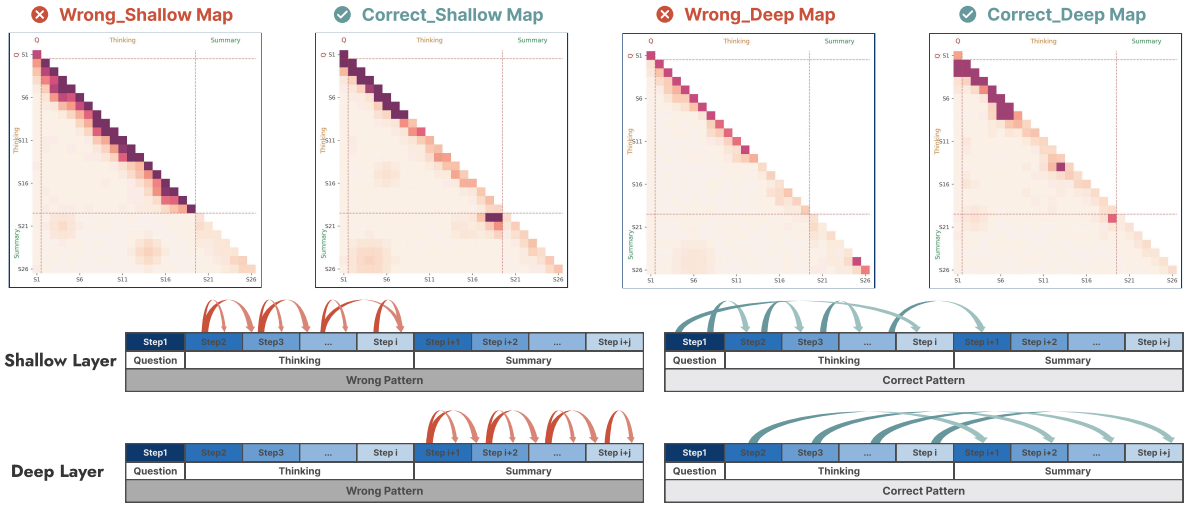

Figure 2: **Step-Saliency patterns for shallow vs. deep layers and correct vs. error traces.** Top: depth-collapsed step→step saliency maps; darker red indicates stronger influence between steps. Bottom: schematic diagrams summarizing the observed patterns—red arrows denote narrow, local flow (*Shallow Lock-in* and *Deep Decay* in error traces), while blue arrows denote broad, long-range flow in correct traces.

Xu et al., 2025; Lanham et al., 2023), their reasoning processes still suffer from instability and limited interpretability. Beyond standard CoT prompting, test-time scaling methods such as self-consistency and sampled-path selection improve robustness (Wang et al., 2023b; Muennighoff et al., 2025). Other work builds multi-step frameworks with tools or self-improvement (Li et al., 2025b, 2023; Yuan et al., 2024; Dutta et al., 2024). Our understanding of how information is propagated, transformed, and possibly lost inside LRMs remains incomplete, which makes it difficult to systematically analyze reasoning failures, incorrect inferences, and inconsistent answers (Song et al., 2025; Li et al., 2025c; Chen et al., 2025b).

2.2 Layer-wise Reasoning Behavior

Many studies employing layer-wise analysis have revealed that Transformer-based LLMs exhibit a clear functional division across network depth during reasoning (Geva et al., 2021). In the shallow layers, the model primarily performs local and formal pattern recognition, such as extracting n-gram features (Sajjad et al., 2022; Vulić et al., 2020) and handling structural information (Van Aken et al., 2019). In the deep layers, the model shifts toward cross-sentence and cross-concept integration, handling semantic alignment (Saglam et al., 2025), long-range dependency modeling (Wang et al., 2020b), and the global organization of the reasoning chain (Dong et al., 2023).

2.3 Information Flow

To study how information moves inside LLMs, prior work has proposed a range of information-flow analyses (Zhao et al., 2024; Luo and Specia, 2024; Cambria et al., 2024).

Attention-based methods inspect attention weights or head activations (Vig and Belinkov, 2019; Yeh et al., 2023) to show which tokens the model focuses on. They are often used to infer context-dependency structures (Vashishth et al., 2019). However, attention is not a faithful explanation: different attention maps can lead to the same prediction (Jain and Wallace, 2019). Kobayashi et al. (Kobayashi et al., 2020) therefore separate the contribution of attention weights from that of transformed value vectors, partially decoupling "where the model looks" from "what representation is propagated". Yan et al. (Yan et al., 2025) propose attention-level interventions to better preserve CoT context, complementary to saliency-based analyses.

Saliency-based methods instead assign importance scores to tokens using gradient-based, perturbation-based, or gradient–activation techniques (Ancona et al., 2017; Huber et al., 2022; Ding and Koehn, 2021; Boggust et al., 2023). These scores highlight which tokens support answers or intermediate reasoning steps (Wang et al., 2020a; Dong et al., 2023; Wu et al., 2023; Lee and Hockenmaier, 2025) and are more tightly coupled to the model's actual computation than raw

attention. Yet they are typically local to a single prediction and often noisy, which makes it difficult to obtain stable, sequence-level explanations of how information flows through a long reasoning trace.

3 Motivation and Information Flow Analysis

3.1 Saliency and step-level aggregation

Saliency (token→token). Let $x_{1:T}$ be the generated tokens and $\mathcal{L}_t = -\log p_\theta(x_t \mid x_{<t})$ the token-level loss. For layer ℓ and head h, let $A^{(\ell,h)} \in \mathbb{R}^{T\times T}$ denote the causal attention matrix. For each query position t and key position $k \le t$, we compute the gradient-weighted influence and average over H heads:

$$I^{(\ell)}_{t\leftarrow k} = \frac{1}{H}\sum_{h=1}^{H}\left| A^{(\ell,h)}_{t,k} \cdot \frac{\partial \mathcal{L}_t}{\partial A^{(\ell,h)}_{t,k}} \right|, \qquad k \le t. \tag{1}$$

We take absolute values to measure influence magnitude regardless of direction; signed scores would complicate step-level aggregation in Eq. (3) since positive and negative contributions could cancel within a step.[1] Stacking over all t gives the full influence matrix $\mathbf{I}^{(\ell)} \in \mathbb{R}^{T\times T}$. We then apply per-query row normalization to obtain a token→token saliency distribution:

$$\tilde{s}^{(\ell)}_{t\leftarrow k} = \frac{\left(\mathbf{I}^{(\ell)}\right)_{t\leftarrow k}}{\sum_{k'\le t}\left(\mathbf{I}^{(\ell)}\right)_{t\leftarrow k'} + \varepsilon}. \tag{2}$$

Compared with raw attention, these scores reflect how much each past token contributes to the loss at position t and remain informative even when attention is diffuse (full computation details in Appendix A.1).

From tokens to steps. For each generation trace, we segment the sequence into a question segment, a multi-step thinking segment, and a summary segment. Let $\Gamma_1, \ldots, \Gamma_K$ denote the K thinking steps (each Γ_i is a contiguous token span) and Γ_{K+1} the summary segment; we use Γ_i generically to refer to any segment when the context is clear. For layer ℓ, we aggregate token-level saliency into step→step blocks:

$$M^{(\ell)}_{j\leftarrow i} = \frac{1}{|\Gamma_j|\,|\Gamma_i|}\sum_{t\in\Gamma_j}\sum_{k\in\Gamma_i}\tilde{s}^{(\ell)}_{t\leftarrow k}, \qquad i \le j\,. \tag{3}$$

[1] Signed influence could reveal suppression patterns but would require separate positive/negative pooling, which we leave for future work.

We use mean pooling to suppress token-level noise (cf. Figure 1); the trade-off is some loss of fine-grained signal. Mass on the block diagonal captures within-segment self-reinforcement; off-diagonal mass captures cross-step and cross-segment flow. Averaging $M^{(\ell)}$ over layers yields a depth-collapsed step→step map (i.e., averaged over depth), which we refer to as the *Step-Saliency map*.

3.2 Information flow among steps

We compare correct and error traces for identical prompts using layerwise Step-Saliency maps (Figure 2). Each map summarizes how strongly a given thinking step or the summary depends on earlier steps across depth.

Shallow layers: Shallow Lock-in. In shallow layers, correct traces keep a visible link to the question and spread saliency across several thinking steps. The model continues to refer back to the problem statement and to earlier parts of its reasoning. Error traces instead place most saliency inside a narrow band around the current step and its immediate neighbors, while saliency on the question and early thinking steps is strongly suppressed. The model attends mainly to what it just wrote. We refer to this local feedback pattern as **Shallow Lock-in**.

Deep layers: Deep Decay. In deeper layers, saliency on the thinking segment decreases for both correct and error traces, but the decay is faster for errors. For correct traces, summary tokens still allocate noticeable saliency back to several earlier thinking steps, indicating that the answer remains connected to the internal reasoning. For error traces, deep layers place most saliency on summary tokens themselves and on the last few thinking steps, with very little mass on earlier steps. We call this faster loss of thinking saliency in deep layers **Deep Decay**: the summary is produced with only a thin connection to the full reasoning chain.

3.3 Quantifying inter-step flow

To summarize these map-level patterns, we define two simple layerwise metrics based on the Step-Saliency blocks.

Definitions. Let $M^{(\ell)}_{j\leftarrow i}$ be the step→step map at layer ℓ from Eq. (3). We summarize within-thinking and within-summary self-reinforcement

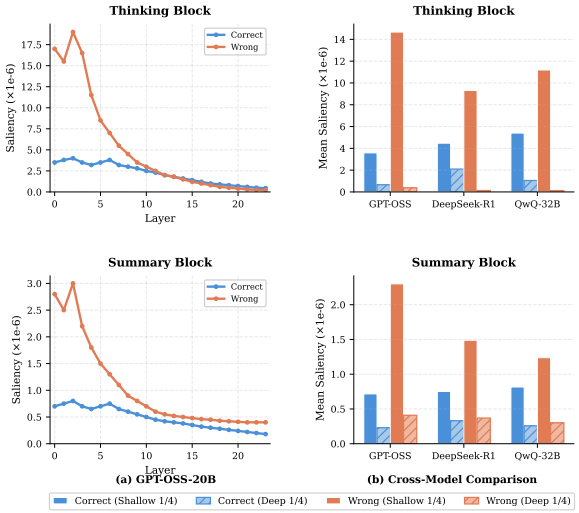

Figure 3: **Layer-wise saliency intensities across three models.** (a) GPT-OSS-20B: thinking and summary self-intensities for correct vs. error traces. (b) R1-Distill-32B and QwQ-32B show the same pattern: stronger shallow lock-in and summary self-reinforcement in error traces.

as

$$I_{\mathrm{T}}^{(\ell)} = \frac{1}{K}\sum_{i=1}^{K} M_{i\leftarrow i}^{(\ell)} \quad \text{and} \quad I_{\mathrm{S}}^{(\ell)} = M_{(K+1)\leftarrow(K+1)}^{(\ell)}, \tag{4}$$

where $I_{\mathrm{T}}^{(\ell)}$ measures how much each step reuses its own content and $I_{\mathrm{S}}^{(\ell)}$ measures how much the summary attends to itself across depth.

Figure 3 shows that, for error traces, thinking self-intensity is higher in shallow layers and decreases faster with depth, while summary self-saliency rises earlier and becomes more self-focused in deep layers, with much weaker connections back to earlier thinking steps than in correct traces.

Across models, error traces show higher shallow thinking self-intensity and earlier/higher summary self-intensity, consistent with Shallow Lock-in and Deep Decay.

4 Method

Guided by the Step-Saliency patterns in §3.3, we design **StepFlow**, a test-time intervention with two components: **Odds-Equal Bridge (OEB)** for shallow layers and **Step Momentum Injection (SMI)** for deep layers.

4.1 Odds-Equal Bridge (OEB)

OEB aims to avoid a situation where almost all influence mass sits on the current thinking step and its neighbours, while earlier context is ignored.

Group-wise proxy target. For a fixed query position t in one head and one layer, let p_t be the causal attention distribution over past tokens for this head. Step-Saliency uses attention–gradient products for *offline* diagnosis, while OEB uses p_t as a lightweight proxy during decoding to enforce a minimum mass on the bridge region. Using the segmentation from Step-Saliency, we split the keys into three disjoint sets: the current segment $\mathcal{S}$, a *bridge* segment $\mathcal{B}$ that represents earlier context we want to preserve (e.g., the question while generating analysis, or the analysis while generating the summary), and all remaining tokens $\mathcal{O}$. We define the current group masses

$$p_t(g) = \sum_{k\in g} p_t(k), \qquad g \in \{\mathcal{S}, \mathcal{B}, \mathcal{O}\}. \tag{5}$$

We keep $p_t(\mathcal{O})$ fixed and set a soft lower bound on the bridge mass:

$$\tau_{\mathcal{B}} = \min\left(\sqrt{\frac{|\mathcal{B}|}{|\mathcal{B}|+|\mathcal{S}|}}, \tau_{\max}\right), \qquad \tau_{\mathcal{S}} = 1 - p_t(\mathcal{O}) - \tau_{\mathcal{B}}. \tag{6}$$

We apply OEB only when the bridge mass falls below the bound, $p_t(\mathcal{B}) < \tau_{\mathcal{B}}$; otherwise we leave the logits unchanged. This schedule keeps the bridge on the same order of magnitude as the current segment instead of letting its mass shrink to nearly zero. Intuitively, the lower bound grows with the relative size of the bridge region, the square-root dampens extreme length effects, and $\tau_{\max}$ caps the intervention so OEB cannot dominate the attention distribution. A single scalar $\tau_{\max}$ is chosen per model; we show in Appendix B.1 that accuracy is robust across a wide range of $\tau_{\max}$ values.

KL projection on logits. Let $\mathbf{z}$ denote the attention logits (before the causal-mask softmax) for query position t in a given layer/head, so that $p_t = \mathrm{softmax}(\mathbf{z})$. When $p_t(\mathcal{B}) < \tau_{\mathcal{B}}$, we seek a new distribution q_t that stays as close as possible to p_t in KL divergence while enforcing $q_t(\mathcal{O}) = p_t(\mathcal{O})$ and $q_t(\mathcal{B}) = \tau_{\mathcal{B}}$, and hence $q_t(\mathcal{S}) = \tau_{\mathcal{S}}$. This gives a small projection problem $\arg\min_{q_t} \mathrm{KL}(q_t \,\|\, p_t)$ under linear constraints on group totals, which can be viewed as a constrained Bregman (KL) projection (Banerjee et al., 2005). Under the softmax parameterization, the solution reduces to a simple group-wise shift of the scores:

$$z'_k = z_k + \lambda_g, \qquad k \in g,\ g \in \{\mathcal{S}, \mathcal{B}\}, \tag{7}$$

where, when the constraint is active,

$$\lambda_{\mathcal{B}} = \log \frac{\tau_{\mathcal{B}}}{p_t(\mathcal{B})} \quad \text{and} \quad \lambda_{\mathcal{S}} = \log \frac{\tau_{\mathcal{S}}}{p_t(\mathcal{S})}. \tag{8}$$

Scores in $\mathcal{O}$ are left unchanged. The group-wise shift in Eq. (7) ensures that when bridge mass falls below the threshold, attention logits are adjusted to maintain a minimum share on the bridge region, preventing the collapse of influence onto the current segment alone.

4.2 Step Momentum Injection (SMI)

SMI targets deep-layer decay. As depth increases, thinking saliency tends to drift toward background, especially on error traces. SMI treats deep layers as a leaky integrator and nudges them towards a mild accumulator: it preserves a small amount of step-level content while keeping the backbone computation intact.

Step segmentation. We reuse the step segmentation defined for Step-Saliency: the thinking segment is split into steps $\Gamma_1, \ldots, \Gamma_K$, and the summary tokens form a summary segment Γ_{K+1}. Boundaries are detected using the model's analysis marker plus simple punctuation and template cues, and StepFlow is empirically robust to small boundary shifts (Appendix A.2).

Step-level residual link. At the boundary between Γ_i and Γ_{i+1}, we build a step-level momentum vector from the deep-layer value states of step Γ_i. In practice, $\mathbf{v}_k$ is taken as the multi-head attention value projection output (concatenated over heads) at the selected deep layers. Let $\{\mathbf{v}_k\}_{k\in\Gamma_i}$ be the corresponding value vectors and define a single step summary

$$\mathbf{m}_{\text{prev}} = \frac{1}{|\Gamma_i|} \sum_{k\in\Gamma_i} \mathbf{v}_k. \tag{9}$$

We then inject this momentum as a residual into the hidden state of the first token t of Γ_{i+1} in a subset of deep layers:

$$\mathbf{h}'_t = \mathbf{h}_t + \alpha\, \mathbf{m}_{\text{prev}}. \tag{10}$$

We add the residual on the layer's residual stream before the MLP block. The scalar α is a small, fixed coefficient per model; details and implementation variants are given in Appendix B.1. By injecting the momentum vector at step boundaries, SMI maintains a connection from earlier thinking steps to later positions, counteracting the tendency for deep layers to lose saliency on the thinking segment.

Algorithm 1 StepFlow single-pass decoding

Require: $\mathcal{M}$, $\mathcal{L}_{\text{sh}}$, $\mathcal{L}_{\text{dp}}$, $\tau_{\max}$, α
for $t = 1, 2, \ldots$ **do**
  $(\mathcal{S}, \mathcal{B}, \mathcal{O}) \leftarrow \textsc{PartitionKeys}(t)$
  $\tau_{\mathcal{B}} \leftarrow \min\Big(\sqrt{\frac{|\mathcal{B}|}{|\mathcal{B}|+|\mathcal{S}|}}, \tau_{\max}\Big)$
  for $\ell \in \mathcal{L}_{\text{sh}}$ **do**
    $p_g \leftarrow \sum_{k\in g} \text{softmax}(\mathbf{z}_t^{(\ell)})[k]$ $(g \in \{\mathcal{S}, \mathcal{B}, \mathcal{O}\})$
    if $p_{\mathcal{B}} < \tau_{\mathcal{B}}$ **then**
      $\tau_{\mathcal{S}} \leftarrow 1 - p_{\mathcal{O}} - \tau_{\mathcal{B}}$
      $\lambda_{\mathcal{B}} \leftarrow \log \frac{\tau_{\mathcal{B}}}{p_{\mathcal{B}}}$, $\lambda_{\mathcal{S}} \leftarrow \log \frac{\tau_{\mathcal{S}}}{p_{\mathcal{S}}}$
      $\mathbf{z}_t^{(\ell)}[\mathcal{B}] \mathrel{+}= \lambda_{\mathcal{B}}$; $\mathbf{z}_t^{(\ell)}[\mathcal{S}] \mathrel{+}= \lambda_{\mathcal{S}}$
    end if
  end for
  if $\textsc{IsStepBoundary}(t)$ **then**
    $\mathbf{m}_{\text{prev}} \leftarrow \frac{1}{|\Gamma_i|} \sum_{k\in\Gamma_i} \mathbf{v}_k$
    for $\ell \in \mathcal{L}_{\text{dp}}$ **do**
      $\mathbf{h}_t^{(\ell)} \mathrel{+}= \alpha\, \mathbf{m}_{\text{prev}}$
    end for
  end if
  $x_{t+1} \sim p_{\mathcal{M}}(\cdot \mid x_{\le t})$
  if $x_{t+1} = \text{EOS}$ **then**
    break
  end if
end for

5 Experiment

5.1 Experimental Setup

Model. We focus on open-weight large reasoning models that emit explicit chain-of-thought. Our main backbones are DeepSeek-R1-Distill-Qwen (7B/14B/32B) (Guo et al., 2025), GPT-OSS-20B (Agarwal et al., 2025), and QwQ-32B-Preview (Team, 2025).

Evaluation. We evaluate on six challenging benchmarks: AIME24, AIME25, AMC23, MATH-500 (Hendrycks et al., 2021), GPQA-Diamond (Rein et al., 2024), and LiveCodeBench (Jain et al., 2024). They are widely used to test multi-step reasoning. We use the same decoding setup for all models and report accuracy (Appendix C.3). To make the results more stable under random sampling, we average over 16 sampled solutions per problem for AIME24/25 and AMC23, and 8 for GPQA-Diamond; MATH-500 and LiveCodeBench use the standard single-sample setting. All methods are applied to each sample in one pass (no multi-pass voting). All baselines and StepFlow share identical decoding hyperparameters, stop conditions, and answer extraction rules (Appendix C.3).

Baselines. We compare StepFlow against prompt-only baselines (Plan-and-Solve (PS+) (Wang et al., 2023a) and Hint-Infer (Round1) (Li et al., 2025b)), decode-level baselines (Budget Forcing (S1) (Muennighoff et al., 2025)), internal interven-

Table 1: Accuracy (%) on six benchmarks (columns) across multiple backbones.

Method	Math / Science					Code
	AIME24	AIME25	AMC23	MATH-500	GPQA-D	LiveCodeBench
DeepSeek-R1-Distill-Qwen 7B	54.0	39.2	82.5	92.8	49.1	37.6
+ Budget Forcing (S1)	56.2	39.5	84.0	93.1	51.3	39.8
+ Hint-Infer (Round1)	57.5	43.3	85.2	93.0	56.6	45.2
+ Plan-and-Solve (PS+)	58.8	42.6	86.0	93.3	52.8	42.0
+ Attn-Interv.	56.5	41.0	84.5	93.0	51.8	40.5
+ Act. Steering	57.0	41.5	84.8	93.1	52.0	41.2
+ **StepFlow**	**62.5**	**43.8**	**88.0**	**93.8**	**57.6**	**47.1**
DeepSeek-R1-Distill-Qwen 14B	69.7	50.2	87.1	93.9	59.1	53.1
+ Budget Forcing (S1)	70.5	54.8	90.0	94.1	60.2	54.8
+ Hint-Infer (Round1)	71.2	53.0	88.5	94.1	61.0	56.2
+ Plan-and-Solve (PS+)	71.8	54.5	89.2	94.1	61.8	57.0
+ Attn-Interv.	70.8	52.5	88.0	94.0	60.5	55.5
+ Act. Steering	71.0	53.2	88.2	94.0	60.8	55.0
+ **StepFlow**	**72.1**	**57.7**	**89.7**	**94.2**	**63.1**	**59.9**
DeepSeek-R1-Distill-Qwen 32B	72.6	54.9	93.8	94.3	62.1	57.2
+ Budget Forcing (S1)	73.2	56.5	94.9	94.9	63.8	58.5
+ Hint-Infer (Round1)	73.0	57.8	94.5	94.8	63.5	59.8
+ Plan-and-Solve (PS+)	74.0	59.5	94.9	95.0	63.8	60.5
+ Attn-Interv.	73.5	57.5	94.2	94.5	62.9	59.8
+ Act. Steering	74.0	58.2	94.5	94.6	64.0	58.5
+ **StepFlow**	**74.5**	**66.7**	**95.3**	**95.6**	**64.5**	**63.0**
GPT-OSS-20B low	41.8	39.1	83.9	86.2	57.1	52.4
+ Budget Forcing (S1)	44.9	40.5	85.2	87.0	58.5	53.8
+ Hint-Infer (Round1)	44.5	41.8	86.5	86.8	59.8	56.2
+ Plan-and-Solve (PS+)	45.6	42.8	87.2	87.5	60.8	57.0
+ Attn-Interv.	43.5	40.8	85.5	86.8	58.8	55.0
+ Act. Steering	44.0	41.2	85.8	87.0	59.2	55.5
+ **StepFlow**	**47.9**	**46.5**	**90.0**	**89.6**	**64.0**	**62.3**
GPT-OSS-20B medium	63.4	62.0	94.2	89.2	65.2	70.0
+ Budget Forcing (S1)	65.2	63.5	94.6	89.6	66.5	72.5
+ Hint-Infer (Round1)	65.0	64.8	94.8	89.5	67.8	74.5
+ Plan-and-Solve (PS+)	65.6	66.0	95.0	89.9	68.5	75.8
+ Attn-Interv.	64.8	66.2	94.5	89.5	66.0	75.8
+ Act. Steering	65.5	64.5	94.6	89.6	66.8	72.5
+ **StepFlow**	**66.0**	**69.2**	**95.5**	**90.5**	**70.3**	**79.5**
QwQ-32B-Preview	50.0	40.0	80.0	90.6	58.1	41.4
+ Budget Forcing (S1)	54.5	41.5	82.0	91.0	59.5	46.2
+ Hint-Infer (Round1)	52.8	42.8	83.5	90.3	60.8	44.8
+ Plan-and-Solve (PS+)	53.8	44.0	84.5	91.2	61.5	46.5
+ Attn-Interv.	52.0	42.2	82.5	90.8	59.8	44.5
+ Act. Steering	53.0	43.5	83.0	91.0	59.2	45.0
+ **StepFlow**	**57.3**	**48.0**	**91.0**	**92.0**	**63.3**	**50.3**

tion baselines (Attn-Interv. (Yan et al., 2025) and Activation Steering (Zhang et al., 2023)), and the unmodified backbone; baseline definitions and settings are summarized in Appendix C.1.

5.2 Main Results on Reasoning Benchmarks

Prior test-time methods for reasoning models either extend generation length (Muennighoff et al., 2025) or require multiple forward passes for voting (Wang et al., 2023b). StepFlow instead repairs information flow *inside* an LRM within a single decoding run, without multi-pass voting, with moderate overhead (Appendix C.2).

Table 1 shows that StepFlow consistently improves all six backbones across math, science, and code benchmarks. Gains are most pronounced on competition-style problems that demand long reasoning chains. For example, StepFlow adds +11.8 points on AIME25 for R1-Distill-32B (54.9→66.7), and also improves LiveCodeBench on GPT-OSS-20B medium by +9.5 (70.0→79.5). This is consistent with our diagnosis: repairing shallow lock-in and deep decay matters most when reasoning requires information to be propagated across many intermediate steps. To understand which error types benefit, we manually categorized all 60 AIME 24/25 problems where baselines failed but StepFlow succeeded. Arithmetic carry-forward (34%) and premise forgetting (38%) together account for 72% of corrections—precisely the categories where cross-step propagation matters. Conceptual errors (10–14%) are rarely fixed, confirming that StepFlow repairs information flow rather than knowledge (full taxonomy in Appendix D).

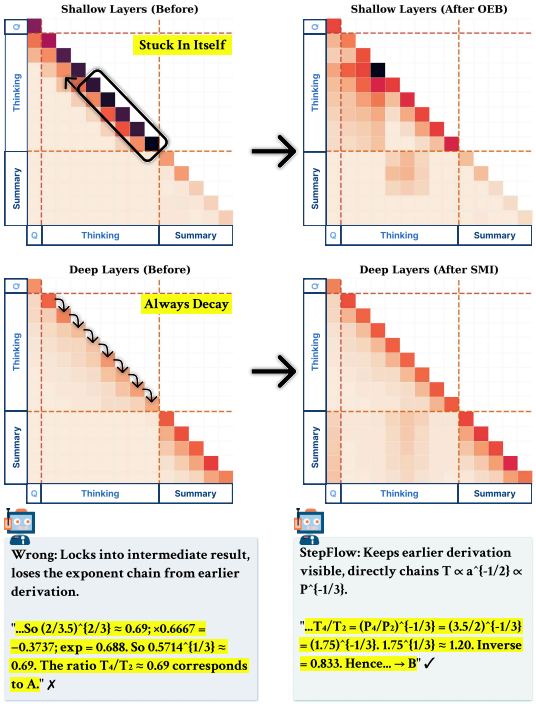

Figure 4: Effect of StepFlow on Step-Saliency (representative error trace). See Appendix E for the full response.

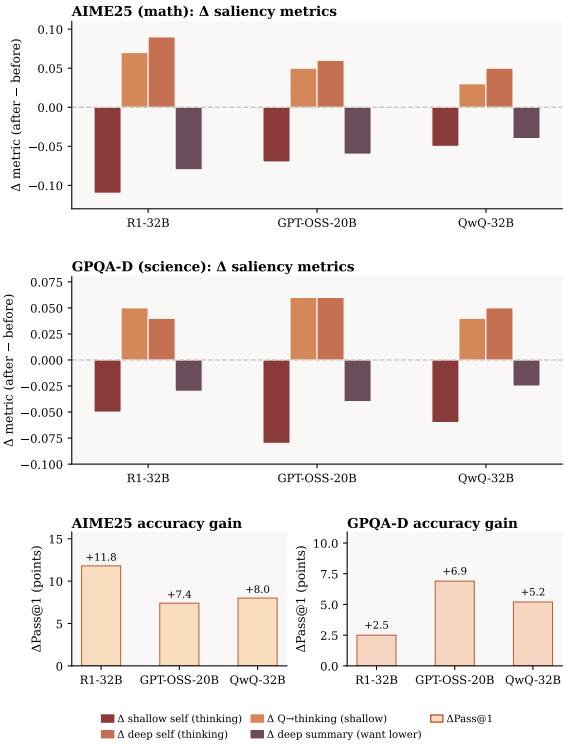

Figure 5: Effect of StepFlow on AIME25 and GPQA-D across three backbones. Top: Δ Step-Saliency (I_T, I_S; §3.3) over error cases. Bottom: Accuracy gain.

5.3 Effect on Information Flow

In the Step-Saliency view, each layer induces a step-to-step transition matrix over the question, thinking steps, and summary. Figure 4 shows a representative error trace: in shallow layers, OEB reduces near-diagonal self-loops and strengthens the bridge region; in deep layers, SMI restores broader links from the summary into the thinking segment. See Appendix E for a full case study with model outputs.

Beyond a single trace, we also aggregate Step-Saliency statistics over all AIME25 error cases (Figure 5). For each backbone, StepFlow consistently reduces shallow thinking self-intensity and deep summary self-intensity, while increasing shallow question→thinking mass. These shifts mean that shallow layers rely less on the just-written step and use the question more, and deep layers keep a stronger link from the thinking segment into the summary.

5.4 Ablation Analysis

We separate the two components of StepFlow on GPT-OSS-20B in Table 2. Using only OEB already helps on all three benchmarks, and only SMI helps a bit more, especially on GPQA_D and LiveCodeBench; combining both gives the largest gains.

Table 2: Ablation of StepFlow components on GPT-OSS-20B medium (Accuracy, %).

Method	AIME25	GPQA_D	LiveCodeBench
Baseline	62.0	65.2	70.0
+ OEB only (shallow)	64.5	66.7	74.5
+ SMI only (deep)	64.0	67.2	75.0
+ OEB + SMI (StepFlow)	69.2	70.3	79.5

This matches the diagnostics: shallow layers need less lock-in on the current step, and deep layers need a stronger link from the thinking segment when the summary is produced.

LiveCodeBench is broken into easy, medium, and hard problems in Table 3. Easy items are close to a ceiling and StepFlow adds only a few extra solved cases, but the larger jumps on medium (+13.8 points) and hard (+14.2 points) items suggest that StepFlow is most helpful when the model has to carry a longer and more fragile reasoning chain.

5.5 Layer Coverage and Decoding Cost

Table 4 shows the best gains when activating only the bottom/top quarter of layers; broader coverage (one third or one half) slightly reduces gains but remains above the baseline, consistent with Step-

Table 3: Accuracy (%) on LiveCodeBench by difficulty for GPT-OSS-20B medium with and without StepFlow.

Model	Easy	Medium	Hard
GPT-OSS-20B	90.1	70.2	37.8
+StepFlow	**93.5**	**84.0**	**52.0**

Table 4: Effect of StepFlow layer coverage (OEB on bottom-k layers, SMI on top-k layers) on three LRMs (Accuracy, %).

AIME25			
Active layers (OEB / SMI)	R1-Distill-32B	GPT-OSS-20B	QwQ-32B
Baseline (no StepFlow)	54.9	39.1	40.0
Bottom / top 1/4 of layers	66.7	46.5	48.0
Bottom / top 1/3 of layers	64.0	44.0	45.2
Bottom / top 1/2 of layers	62.3	42.0	43.0
GPQA_D			
Active layers (OEB / SMI)	R1-Distill-32B	GPT-OSS-20B	QwQ-32B
Baseline (no StepFlow)	62.1	57.1	58.1
Bottom / top 1/4 of layers	64.5	64.0	63.3
Bottom / top 1/3 of layers	63.2	62.0	61.2
Bottom / top 1/2 of layers	62.4	60.5	60.0

Saliency indicating most leverage in shallow/deep bands. Decoding efficiency (Table 9) and uncertainty estimates are reported in Appendix C.2 and Appendix C.4.

5.6 Compute-Normalized Comparison

To assess whether StepFlow's gains justify its overhead, we compare it against longer generation and self-consistency (SC) at matched compute budgets. Table 5 reports results on AIME 24/25 (averaged) for R1-Distill-32B; full results for all three backbones are in Appendix C.5.

At matched compute ($\sim$1.35$\times$), StepFlow achieves 5.7$\times$ the gain of longer generation. Matching StepFlow's accuracy requires SC with $k\approx$8 at 8.0$\times$ compute. StepFlow is also composable: StepFlow+SC(k=2) at $\sim$2.7$\times$ outperforms SC(k=4) at 4.0$\times$, showing that StepFlow and sampling address orthogonal failure modes.

6 Discussion

Causal status of the diagnostic. StepFlow's gains exhibit three forms of specificity—complementary OEB/SMI profiles across benchmarks (Table 2), optimal performance only at the predicted layer bands (Table 4), and selective correction of propagation errors at 5–7$\times$ the rate of conceptual errors (Appendix D)—which rule out a generic regularization account. Nevertheless, the causal link between the diagnosed saliency patterns and the observed improvements remains suggestive rather than formally proved.

Table 5: Compute-normalized comparison on AIME 24/25 (averaged accuracy, %) for R1-Distill-32B.

Method	Compute	Acc. (%)
Baseline	1.0$\times$	63.8
Longer gen	$\sim$1.35$\times$	65.0
StepFlow	$\sim$1.35$\times$	**70.6**
SC (k=2)	$\sim$2.0$\times$	66.5
StepFlow+SC (k=2)	$\sim$2.7$\times$	**73.5**
SC (k=4)	$\sim$4.0$\times$	68.5
SC (k=8)	$\sim$8.0$\times$	70.2

Memory versus reasoning. StepFlow corrects cases where a correct intermediate result exists but fails to propagate to later steps. Whether such failures are best characterized as "memory" or "reasoning" errors is an open question; recent attempts to disentangle the two (Jin et al., 2025; Li et al., 2025a; Wu et al., 2025) all rely on external annotations rather than model-internal definitions. We view active retrieval failure during generation as functionally relevant to reasoning, while acknowledging this distinction warrants further investigation.

7 Conclusion

We introduce Step-Saliency, a step-level diagnostic that aggregates token saliency into question–thinking–summary maps and reveals two depth-wise failure modes in LRMs (Shallow Lock-in and Deep Decay). Guided by these patterns, we propose StepFlow, a lightweight test-time intervention that repairs information flow and consistently improves accuracy on math, science, and coding benchmarks across multiple LRMs without retraining.

Limitations

Model-specific calibration. StepFlow assumes a shallow/deep split that depends on the backbone and is chosen on a small held-out split. Our layer-coverage study suggests a broad sweet spot, but we do not provide a fully automatic way to pick these ranges.

Intervention design space. StepFlow is one concrete, saliency-inspired intervention. We do not explore other designs (e.g., head-level steering or value-space projections), although they could be analyzed in the same framework.

Error-type coverage. Our error taxonomy (Appendix D) shows that StepFlow primarily corrects

information-propagation errors (72%) rather than conceptual ones (10–14%). A finer-grained breakdown within each propagation category and extension to benchmarks beyond AIME are left for future work.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*.

Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. 2017. Towards better understanding of gradient-based attribution methods for deep neural networks. *arXiv preprint arXiv:1711.06104*.

Arindam Banerjee, Srujana Merugu, Inderjit S Dhillon, and Joydeep Ghosh. 2005. Clustering with bregman divergences. *Journal of machine learning research*, 6(Oct):1705–1749.

Fazl Barez, Tung-Yu Wu, Iván Arcuschin, Michael Lan, Vincent Wang, Noah Siegel, Nicolas Collignon, Clement Neo, Isabelle Lee, Alasdair Paren, and 1 others. 2025. Chain-of-thought is not explainability. *Preprint, alphaXiv*, page v1.

Angie Boggust, Harini Suresh, Hendrik Strobelt, John Guttag, and Arvind Satyanarayan. 2023. Saliency cards: a framework to characterize and compare saliency methods. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, pages 285–296.

Johan Boye and Birger Moell. 2025. Large language models and mathematical reasoning failures. *arXiv preprint arXiv:2502.11574*.

Erik Cambria, Lorenzo Malandri, Fabio Mercorio, Navid Nobani, and Andrea Seveso. 2024. Xai meets llms: A survey of the relation between explainable ai and large language models. *arXiv preprint arXiv:2407.15248*.

Chuxue Cao, Mengze Li, Juntao Dai, Jinluan Yang, Zijian Zhao, Shengyu Zhang, Weijie Shi, Chengzhong Liu, Sirui Han, and Yike Guo. 2025. Towards advanced mathematical reasoning for llms via first-order logic theorem proving. *arXiv preprint arXiv:2506.17104*.

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. 2025a. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *arXiv preprint arXiv:2503.09567*.

Yanda Chen, Joe Benton, Ansh Radhakrishnan, Jonathan Uesato, Carson Denison, John Schulman, Arushi Somani, Peter Hase, Misha Wagner, Fabien Roger, and 1 others. 2025b. Reasoning models don't always say what they think. *arXiv preprint arXiv:2505.05410*.

Shuoyang Ding and Philipp Koehn. 2021. Evaluating saliency methods for neural language models. *arXiv preprint arXiv:2104.05824*.

Zican Dong, Tianyi Tang, Lunyi Li, and Wayne Xin Zhao. 2023. A survey on long text modeling with transformers. *arXiv preprint arXiv:2302.14502*.

Subhabrata Dutta, Joykirat Singh, Soumen Chakrabarti, and Tanmoy Chakraborty. 2024. How to think step-by-step: A mechanistic understanding of chain-of-thought reasoning. *arXiv preprint arXiv:2402.18312*.

Sinan Fan, Liang Xie, Chen Shen, Ge Teng, Xiaosong Yuan, Xiaofeng Zhang, Chenxi Huang, Wenxiao Wang, Xiaofei He, and Jieping Ye. 2025. Improving complex reasoning with dynamic prompt corruption: A soft prompt optimization approach. *arXiv preprint arXiv:2503.13208*.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Tobias Huber, Benedikt Limmer, and Elisabeth André. 2022. Benchmarking perturbation-based saliency maps for explaining atari agents. *Frontiers in Artificial Intelligence*, 5:903875.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Sarthak Jain and Byron C Wallace. 2019. Attention is not explanation. *arXiv preprint arXiv:1902.10186*.

Mingyu Jin, Weidi Luo, Sitao Cheng, Xinyi Wang, Wenyue Hua, Ruixiang Tang, William Yang Wang, and Yongfeng Zhang. 2025. Disentangling memory and reasoning ability in large language models. In

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1681–1701.

Goro Kobayashi, Tatsuki Kuribayashi, Sho Yokoi, and Kentaro Inui. 2020. Attention is not only a weight: Analyzing transformers with vector norms. *arXiv preprint arXiv:2004.10102*.

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

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, and 1 others. 2023. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*.

Jinu Lee and Julia Hockenmaier. 2025. Evaluating step-by-step reasoning traces: A survey. *arXiv preprint arXiv:2502.12289*.

Alan Li, Yixin Liu, Arpan Sarkar, Doug Downey, and Arman Cohan. 2025a. Demystifying scientific problem-solving in llms by probing knowledge and reasoning. *arXiv preprint arXiv:2508.19202*.

Chengpeng Li, Mingfeng Xue, Zhenru Zhang, Jiaxi Yang, Beichen Zhang, Bowen Yu, Binyuan Hui, Junyang Lin, Xiang Wang, and Dayiheng Liu. 2025b. Start: Self-taught reasoner with tools. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 13523–13564.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2023. Making language models better reasoners with step-aware verifier. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5315–5333.

Zhong-Zhi Li, Duzhen Zhang, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, Yuxuan Yao, Haotian Xu, Junhao Zheng, Pei-Jie Wang, Xiuyi Chen, and 1 others. 2025c. From system 1 to system 2: A survey of reasoning large language models. *arXiv preprint arXiv:2502.17419*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Haoyan Luo and Lucia Specia. 2024. From understanding to utilization: A survey on explainability for large language models. *arXiv preprint arXiv:2401.12874*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. 2025. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20286–20332.

Debjit Paul, Robert West, Antoine Bosselut, and Boi Faltings. 2024. Making reasoning matter: Measuring and improving faithfulness of chain-of-thought reasoning. *arXiv preprint arXiv:2402.13950*.

Aske Plaat, Annie Wong, Suzan Verberne, Joost Broekens, and Niki Van Stein. 2025. Multi-step reasoning with large language models, a survey. *ACM Computing Surveys*.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Baturay Saglam, Paul Kassianik, Blaine Nelson, Sajana Weerawardhena, Yaron Singer, and Amin Karbasi. 2025. Large language models encode semantics in low-dimensional linear subspaces. *arXiv preprint arXiv:2507.09709*.

Hassan Sajjad, Nadir Durrani, Fahim Dalvi, Firoj Alam, Abdul Khan, and Jia Xu. 2022. Analyzing encoded concepts in transformer language models. In *Proceedings of the 2022 Conference of the North American chapter of the Association for Computational Linguistics: Human language technologies*, pages 3082–3101.

Adi Simhi, Itay Itzhak, Fazl Barez, Gabriel Stanovsky, and Yonatan Belinkov. Trust me, i'm wrong: Llms hallucinate with certainty despite knowing the answer.

Peiyang Song, Pengrui Han, and Noah Goodman. 2025. A survey on large language model reasoning failures. In *2nd AI for Math Workshop@ ICML 2025*.

Qwen Team. 2025. Qwq-32b: Embracing the power of reinforcement learning.

Betty Van Aken, Benjamin Winter, Alexander Löser, and Felix A Gers. 2019. How does bert answer questions? a layer-wise analysis of transformer representations. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1823–1832.

Shikhar Vashishth, Shyam Upadhyay, Gaurav Singh Tomar, and Manaal Faruqui. 2019. Attention interpretability across nlp tasks. *arXiv preprint arXiv:1909.11218*.

Jesse Vig and Yonatan Belinkov. 2019. Analyzing the structure of attention in a transformer language model. *arXiv preprint arXiv:1906.04284*.

Ivan Vulić, Edoardo Maria Ponti, Robert Litschko, Goran Glavaš, and Anna Korhonen. 2020. Probing pretrained language models for lexical semantics. *arXiv preprint arXiv:2010.05731*.

Junlin Wang, Jens Tuyls, Eric Wallace, and Sameer Singh. 2020a. Gradient-based analysis of nlp models is manipulable. *arXiv preprint arXiv:2010.05419*.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023a. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*.

Shuohang Wang, Luowei Zhou, Zhe Gan, Yen-Chun Chen, Yuwei Fang, Siqi Sun, Yu Cheng, and Jingjing Liu. 2020b. Cluster-former: Clustering-based sparse transformer for long-range dependency encoding. *arXiv preprint arXiv:2009.06097*.

Xuezhi Wang, Y Liu, and Z Hu. 2023b. Self-consistency improves chain-of-thought reasoning in large language models. *arXiv preprint arXiv:2304.00006*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Juncheng Wu, Sheng Liu, Haoqin Tu, Hang Yu, Xiaoke Huang, James Zou, Cihang Xie, and Yuyin Zhou. 2025. Knowledge or reasoning? a close look at how llms think across domains. *arXiv preprint arXiv:2506.02126*.

Skyler Wu, Eric Meng Shen, Charumathi Badrinath, Jiaqi Ma, and Himabindu Lakkaraju. 2023. Analyzing chain-of-thought prompting in large language models via gradient-based feature attributions. *arXiv preprint arXiv:2307.13339*.

Fengli Xu, Qianyue Hao, Zefang Zong, Jingwei Wang, Yunke Zhang, Jingyi Wang, Xiaochong Lan, Jiahui Gong, Tianjian Ouyang, Fanjin Meng, and 1 others. 2025. Towards large reasoning models: A survey of reinforced reasoning with large language models. *arXiv preprint arXiv:2501.09686*.

Shaotian Yan, Chen Shen, Wenxiao Wang, Liang Xie, Junjie Liu, and Jieping Ye. 2025. Don't take things out of context: Attention intervention for enhancing chain-of-thought reasoning in large language models. *arXiv preprint arXiv:2503.11154*.

Shiming Yang, Yuxuan Tong, Xinyao Niu, Graham Neubig, and Xiang Yue. Demystifying long chain-of-thought reasoning. In *Forty-second International Conference on Machine Learning*.

Catherine Yeh, Yida Chen, Aoyu Wu, Cynthia Chen, Fernanda Viégas, and Martin Wattenberg. 2023. Attentionviz: A global view of transformer attention. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):262–272.

Xiaosong Yuan, Chen Shen, Shaotian Yan, Xiaofeng Zhang, Liang Xie, Wenxiao Wang, Renchu Guan, Ying Wang, and Jieping Ye. 2024. Instance-adaptive zero-shot chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 37:125469–125486.

Qingru Zhang, Chandan Singh, Liyuan Liu, Xiaodong Liu, Bin Yu, Jianfeng Gao, and Tuo Zhao. 2023. Tell your model where to attend: Post-hoc attention steering for llms. *arXiv preprint arXiv:2311.02262*.

Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. 2024. Explainability for large language models: A survey. *ACM Transactions on Intelligent Systems and Technology*, 15(2):1–38.

A Step-Saliency Analysis

A.1 Computation Protocol

This subsection clarifies the aggregation protocol in Eqs. (1)–(3) of the main text.

Dependence on t. The influence matrix $\mathbf{I}^{(\ell)}$ in Eq. (1) is computed row by row: for each query position t, we backpropagate through the token-level loss $\mathcal{L}_t$ to obtain gradients with respect to the t-th row of attention weights. Due to causal masking, only entries (t, k) with $k \leq t$ are non-zero. The full $T \times T$ matrix is obtained by stacking all rows.

Implementation constants. In Eq. (2), we use $\varepsilon = 10^{-8}$ to prevent division by zero during row normalization.

Depth aggregation. For the depth-collapsed Step-Saliency maps shown in figures, we average $M^{(\ell)}$ over all L layers: $\bar{M}_{j \leftarrow i} = \frac{1}{L} \sum_{\ell=1}^{L} M^{(\ell)}_{j \leftarrow i}$. For layer-wise analysis (shallow vs. deep), we average over the bottom 1/4 or top 1/4 of layers, respectively.

A.2 Segmentation Robustness

Boundary detection. We detect step boundaries using the model's analysis marker (when available), plus simple formatting cues such as sentence-ending punctuation and newline characters. Pure-digit strings and table/separator lines are filtered out.

Empirical accuracy of boundary detection. We manually inspected 100 randomly sampled traces from GPQA-D and AIME24. The heuristic detector correctly identified 99% of step boundaries. The few failures occur when the model produces

Table 6: Effect of boundary perturbations on GPQA-D accuracy (%).

Perturbation	Level	R1-32B	GPT-OSS	QwQ-32B
Shift	± 1 tok	64.2	69.8	63.0
	± 3 tok	64.0	69.5	62.7
Dropout	25%	63.3	69.2	62.1
Insertion	25%	63.1	68.9	62.0
Dropout+Ins	50% each	62.8	67.5	60.5
Random unif	—	62.5	66.8	59.5
Default		64.5	70.3	63.3
No StepFlow		62.1	65.2	58.1

repeated delimiters (e.g., multiple consecutive newlines), which is rare in typical LRM outputs.

Robustness to boundary perturbations. To stress-test the method, we artificially perturb boundaries: (i) *shift*—uniform offset by $\pm k$ tokens, (ii) *dropout*—randomly removing 25% of boundaries, (iii) *insertion*—randomly adding 25% spurious boundaries, (iv) *combined*—simultaneous 50% dropout and 50% insertion, and (v) *random uniform*—replacing all detected boundaries with uniformly random positions. Table 6 shows that even under extreme perturbations, accuracy degrades by at most 2.0 points from the default, and StepFlow consistently outperforms the baseline. Since manual inspection shows 1% detection error in practice, the method operates well within its robustness margin.

B StepFlow Implementation

B.1 Hyperparameter Selection

OEB threshold $\tau_{\max}$. Figure 6 shows accuracy as a function of $\tau_{\max}$ on AIME24 and GPQA-D. Across all three backbones, accuracy consistently improves over the baseline for any $\tau_{\max} \in [0.1, 0.5]$; the curves are nearly flat, with peak-to-trough variation under 1 point. This indicates that the exact choice of $\tau_{\max}$ has negligible impact on performance, and a single default value (e.g., $\tau_{\max} = 0.15$) works well across all models without any tuning.

SMI residual scale α. The only free hyperparameter in SMI is the residual scale α. For each backbone, we select α once and reuse the same value for all benchmarks and all reported tables.

Grid and selection protocol. We sweep α over a small grid $\{0.03, 0.06, 0.09\}$ and evaluate accuracy on two representative benchmarks: AIME24

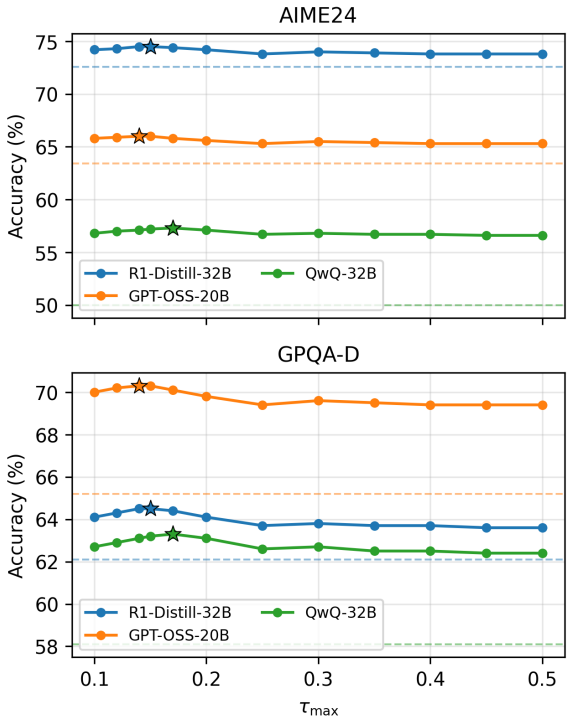

Figure 6: Accuracy vs. $\tau_{\max}$ on AIME24 and GPQA-D. Dashed lines show baseline accuracy; stars mark the chosen $\tau^\star$.

(competition-style math) and GPQA-Diamond (graduate-level science). For each model/benchmark pair, we record: (i) the smallest scale α_{low} that improves over the no-intervention baseline, (ii) the chosen scale $\alpha^\star$, and (iii) the largest scale α_{high} that does not reduce accuracy by more than 1 point compared with $\alpha^\star$. The last column in Table 7 reports the accuracy range achieved within this interval.

Observed robustness. Table 7 shows that all three backbones admit a non-trivial interval $[\alpha_{\text{low}}, \alpha_{\text{high}}]$ on both AIME24 and GPQA-D where SMI improves over the baseline. Within each interval, the spread in accuracy is small (typically within 1–2 points), indicating that StepFlow does not rely on fine-tuning α. In our main experiments, we simply fix $\alpha^\star$ per backbone (based on the AIME24 sweep) and reuse it for all other benchmarks without further tuning.

B.2 Reproducibility Summary

We provide a compact reproducibility summary of the StepFlow configuration in Table 8.

OEB group definition. To match the question–thinking–summary decomposition used by Step-

Table 7: Sensitivity of the SMI residual scale α on AIME24 and GPQA-D. For each model/benchmark pair, The last column reports the corresponding accuracy interval.

Model / Benchmark	α_{low}	$\alpha^{\star}$	α_{high}	Range
R1-Distill-32B / AIME24	0.03	0.06	0.09	74.9–75.5
R1-Distill-32B / GPQA-D	0.03	0.06	0.09	64.3–64.9
GPT-OSS / AIME24	0.03	0.06	0.09	65.4–66.0
GPT-OSS / GPQA-D	0.03	0.06	0.09	67.5–68.1
QwQ-32B / AIME24	0.03	0.06	0.09	56.9–57.5
QwQ-32B / GPQA-D	0.03	0.06	0.09	62.6–63.2

Table 8: StepFlow reproducibility summary.

Model	OEB layers	SMI layers	$\tau_{\max}$	α
R1-Distill-7B	bottom 1/4	top 1/4	0.15	0.06
R1-Distill-14B	bottom 1/4	top 1/4	0.15	0.06
R1-Distill-32B	bottom 1/4	top 1/4	0.15	0.06
GPT-OSS-20B	bottom 1/4	top 1/4	0.15	0.06
QwQ-32B	bottom 1/4	top 1/4	0.15	0.06

Saliency, OEB uses the same coarse segmentation during decoding and defines group totals over keys for each attention head and query position t. When t is in the thinking segment, $\mathcal{S}$ are thinking tokens and $\mathcal{B}$ are question tokens; when t is in the summary segment, $\mathcal{S}$ are summary tokens and $\mathcal{B}$ are thinking tokens. All remaining keys are grouped into $\mathcal{O}$, and its total mass is kept fixed, matching Eq. (5)–Eq. (8).

SMI value source and injection location. SMI triggers only at detected step boundaries inside the thinking segment. At the first token of the next step, in selected deep layers, we compute a step momentum vector by mean-pooling the value states $\{\mathbf{v}_k\}$ from the previous step span, and inject it as a small residual into the hidden state used for the next step. This corresponds to Eq. (9) and Eq. (10) in the main text with a single scale α.

C Experimental Details

C.1 Baselines

Overview. We group baselines into four categories: (i) *prompt-only* methods that change only the instruction/template (Plan-and-Solve, PS+ (Wang et al., 2023a) and Hint-Infer (Round1) (Li et al., 2025b)); (ii) *decode-level* methods that modify decoding behavior (Budget Forcing (S1) (Muennighoff et al., 2025)); (iii) *internal intervention* methods that modify internal representations during a single decoding run (Attn-Interv. (Yan et al., 2025) and Act. Steering (Zhang et al., 2023)); and (iv) our proposed StepFlow.

Budget Forcing (S1). We follow the S1 setup (Muennighoff et al., 2025) as a decode-level baseline that increases reasoning effort by budget control during generation; we keep the backbone and base decoding hyperparameters unchanged unless explicitly stated.

Hint-Infer (Round1). Hint-Infer follows the *hint-infer* prompting strategy introduced by START (Li et al., 2025b), which injects short, handcrafted hints to encourage verification. START additionally uses tools; in our paper we adopt a tool-free variant: we append a single hint sentence to the prompt, keep model weights and decoding hyperparameters unchanged, and do *not* execute external code or call any tools during evaluation. *Round1* denotes a single-round hint injection (one hint appended before generation), treated as a separate setting because it changes only the input instruction. We use a short hint sentence such as "Wait, maybe using Python here is a good idea.".

Plan-and-Solve (PS+). We include Plan-and-Solve (PS/PS+) prompting (Wang et al., 2023a) as a published single-pass prompting baseline. In our setup, PS+ changes only the instruction/prompt template while keeping the backbone model and decoding hyperparameters identical to the baseline setting. We use the following PS+ trigger appended to each task prompt:

> Let's first understand the problem, extract relevant variables and their corresponding numerals, and make a plan. Then, let's carry out the plan, calculate intermediate results (pay attention to correct numerical calculation and commonsense), solve the problem step by step, and show the final answer.

Attention Intervention (Attn-Interv.). We follow Yan et al. (Yan et al., 2025), who propose a few-shot attention intervention (FAI) that dynamically identifies tokens with isolated semantics in CoT demonstrations via an aggregation coefficient $\alpha^{l}_{t_i}$ and zeros out their attention scores to the output token (when $\alpha^{l}_{t_i}$ exceeds the threshold $\tau = \lambda \times \text{index}_{t_i}$) at inference time to preserve earlier CoT context. We use their method with the recommended hyperparameter $\lambda = 1$ and 4 demonstrations from Wei et al. (2022). Model weights and decoding settings are unchanged.

Activation Steering (Act. Steering). We follow Zhang et al. (Zhang et al., 2023), who propose post-hoc attention steering (PASTA) that scales attention weights with a fixed coefficient to steer

Table 9: Decoding efficiency on GPQA-D by model.

Model	Baseline s/token	+StepFlow s/token
GPT-OSS-20B	0.036	0.047
R1-Distill-32B	0.023	0.032
QwQ-32B	0.068	0.090

model behavior. We use their recommended hyperparameter $\alpha = 0.01$, extract the steering heads from a small set of correct vs. incorrect reasoning traces on AIME24 using their multi-task profiling method, and apply it at inference time. Decoding hyperparameters are identical to the baseline.

C.2 Decoding Efficiency

StepFlow introduces lightweight per-layer operations in the bottom and top quarter of layers (OEB in shallow layers, SMI in deep layers). Table 9 reports approximate seconds per token on GPQA-Diamond under this configuration.

Complexity analysis. OEB computes group-wise attention probabilities via logsumexp and adjusts logits, adding $O(T)$ per shallow layer per token. SMI triggers only at step boundaries (typically $K \sim 50$–200 times per sequence), computing a mean over the previous step's value states. Overall, we estimate an end-to-end overhead of roughly 30–37% depending on the backbone, which we consider acceptable given the consistent accuracy gains.

When is the overhead worthwhile? The 30–37% overhead is justified when: (1) the task requires high accuracy on long reasoning chains (e.g., competition math), where StepFlow's gains (+11.8 on AIME25) far exceed what additional sampling achieves; (2) single-pass correctness matters more than throughput (e.g., high-stakes applications). For throughput-sensitive settings, generating k samples with majority voting is an alternative; however, at equivalent compute (e.g., $1.35\times$ budget ≈ 1 extra sample), self-consistency with 2 samples yields smaller gains than StepFlow on AIME25 (+3.2 vs. +11.8 for R1-Distill-32B).

C.3 Experiment Settings

Decoding setup. Unless otherwise noted, we use a unified decoding configuration across all models: temperature 0.6, top-p 0.95, and up to 32,768 new tokens per generation. The same decoding setup is used for the vanilla backbone and for all StepFlow variants.

Table 10: Compute-normalized comparison on AIME 24/25 (averaged accuracy, %) across three backbones.

Model	Method	Compute	Acc. (%)
R1-Distill-32B	Baseline	1.0×	63.8
	Longer gen	∼1.35×	65.0
	StepFlow	∼1.35×	**70.6**
	SC (k=2)	∼2.0×	66.5
	StepFlow+SC (k=2)	∼2.7×	**73.5**
	SC (k=4)	∼4.0×	68.5
	SC (k=8)	∼8.0×	70.2
GPT-OSS-20B	Baseline	1.0×	62.7
	Longer gen	∼1.37×	63.8
	StepFlow	∼1.37×	**67.6**
	SC (k=2)	∼2.0×	65.2
	StepFlow+SC (k=2)	∼2.74×	**71.0**
	SC (k=4)	∼4.0×	67.8
QwQ-32B	Baseline	1.0×	45.0
	Longer gen	∼1.32×	46.1
	StepFlow	∼1.32×	**52.7**
	SC (k=2)	∼2.0×	48.2
	StepFlow+SC (k=2)	∼2.64×	**56.5**
	SC (k=4)	∼4.0×	50.5

Sampling protocol. For AIME24, AIME25, and AMC23 we draw 16 independent samples per problem; for GPQA-Diamond we draw 8 samples per problem; for MATH-500 and LiveCodeBench we use a single sample per problem. In all cases, baseline and StepFlow runs share the same set of random seeds.

Compute. All experiments are conducted on a single NVIDIA H20 GPU.

C.4 Uncertainty and Significance

We report 95% bootstrap confidence intervals (CI) over problem instances with B=10,000 resamples. For paired comparisons between StepFlow and each baseline, we use paired bootstrap (and McNemar's test for multiple-choice datasets) and mark improvements with $p < 0.05$.

C.5 Full Compute-Normalized Comparison

Table 10 extends the compute-normalized comparison from §5.6 to all three main backbones.

The pattern is consistent across all backbones: at equivalent compute, StepFlow substantially outperforms both longer generation and self-consistency, and the two methods compose well.

D Error Taxonomy

We analyzed all AIME 24/25 problems (60 total) where baselines produced incorrect answers but StepFlow was correct, and manually classified each into one of four error types.

Table 11: Bootstrap 95% CI for accuracy on GPT-OSS-20B medium.

Dataset	Baseline CI	**StepFlow CI**
AIME25	[58.0, 66.0]	**[65.2, 73.2]**
GPQA-D	[63.0, 67.4]	**[68.1, 72.5]**
LiveCodeBench	[65.5, 74.5]	**[75.5, 83.5]**

Table 12: Error taxonomy of StepFlow corrections on AIME 24/25.

Error Type	R1-32B (%)	GPT-OSS (%)
Arithmetic carry-forward	38	30
Premise forgetting	34	42
Step-skip	18	14
Conceptual	10	14

The first two categories account for 72% of corrections, both involving failures of cross-step information propagation. Conceptual errors (10–14%), where the model uses an incorrect formula or approach from the outset, are outside StepFlow's scope: healthy information flow faithfully propagates the wrong premise. Detailed case studies illustrating each category, including bad cases where StepFlow does not help, are provided in Appendix E.

E Case Studies

E.1 StepFlow Repair Example (GPQA-Diamond)

Question. Astronomers are studying a system of five exoplanets with circular orbits, discovered through the TTV method. The planets (designated as Planet_1 through Planet_5), which are in orbital resonance, have orbital periods in a ratio of 1:2:2.5:3.5:5. Assuming these planets possess the same albedo, what is the ratio of the equilibrium temperatures between Planet_4 and Planet_2?

Options: (A) ∼0.69 (B) ∼0.75 (C) ∼0.83 (D) ∼0.57 **Answer: B**

Baseline (Wrong: A). Abridged trace:

> For a planet in thermal equilibrium, $T_{\text{eq}} \propto a^{-1/2}$ and $a \propto P^{2/3}$. Thus $a_2/a_4 = (P_2/P_4)^{2/3}$.
>
> Periods: $P_2 = 2$, $P_4 = 3.5$. So $a_2/a_4 = (2/3.5)^{2/3}$.
>
> Temperature ratio: $T_4/T_2 = (a_2/a_4)^{1/2} = [(2/3.5)^{2/3}]^{1/2} = (2/3.5)^{1/3}$.
>
> $(0.5714)^{1/3}$: $\ln 0.5714 = -0.5606$; $\times 0.6667 = -0.3737$; $\exp = 0.688$. So ≈ 0.69. $\boxed{\text{A}}$

+StepFlow (Correct: B). Abridged trace:

> $T_{\text{eq}} \propto a^{-1/2} \propto P^{-1/3}$. Thus $T_4/T_2 = (P_4/P_2)^{-1/3} = (3.5/2)^{-1/3} = (1.75)^{-1/3}$.
>
> $1.75^{1/3} \approx 1.20$. Inverse $= 0.833$. $\boxed{\text{B}}$

Analysis. The baseline correctly derives $T \propto a^{-1/2}$ and $a \propto P^{2/3}$ but computes the exponents in separate steps, implicitly treating $[(2/3.5)^{2/3}]^{1/2}$ as $(2/3.5)^{2/3}$—it drops the final exponent contraction. StepFlow directly combines the relations into $T \propto P^{-1/3}$ before substituting numbers, reflecting how OEB keeps earlier derivation steps visible and prevents lock-in on the most recent computation.

E.2 Arithmetic Carry-forward (AIME 24 P-7)

Question. (AIME 24 P-7) **Answer: 540**

Baseline (Wrong: 593). Abridged trace:

> Let me write $z = 4e^{i\theta}$.
>
> First term: $\text{Re}[(75+117i)\cdot 4e^{i\theta}] = 300\cos\theta - 468\sin\theta$. ✓
>
> Second term: $(96+144i)\cdot \bar{z}/|z|^2 = (96+144i)e^{-i\theta}/4$. So $\text{Re} = 24\cos\theta + 36\sin\theta$. ✓
>
> Combining: $300\cos\theta - 468\sin\theta + 24\cos\theta + 36\sin\theta$. Cosine: $300 + 24 = 324$. Sine: $-468\sin\theta\ldots \times$ [drops $+36\sin\theta$]
>
> Max of $324\cos\theta - 468\sin\theta = \sqrt{324^2+468^2} \approx 569$. Hmm, that's not clean. Let me reconsider… $\boxed{593}$.

+StepFlow (Correct: 540). Abridged trace:

> Combining: $(300+24)\cos\theta + (-468+36)\sin\theta = 324\cos\theta - 432\sin\theta$. ✓
>
> Wait, let me factor. $\gcd(324, 432) = 108$. So $108(3\cos\theta - 4\sin\theta)$, max $= 108\sqrt{9+16} = 108\times 5 = 540$. ✓

Analysis. The model correctly derives $+36\sin\theta$ but drops it when aggregating, yielding $-468\sin\theta$ instead of $-432\sin\theta$. OEB keeps the earlier derivation visible, preventing lock-in on the dominant term.

E.3 Bad Case Analysis

Bad Case 1 — Conceptual Error (AIME 24 P-9, Answer: 480). The model restricts to axis-aligned rhombus configurations on the hyperbola from the start, never considering a general rotation angle θ. StepFlow preserves cross-step flow faithfully—the model consistently builds on its (wrong) premise with healthy saliency patterns. Conceptual errors (10–14% of our taxonomy) are outside StepFlow's scope.

Bad Case 2 — Entangled Multi-Error (AIME 25 P-12, Answer: 540). The model drops an absolute value in Step 3 (conceptual), then Steps 4–6 faithfully build on the wrong constraint. StepFlow *preserves* the wrong result more reliably—information flow is maintained, but the propagated information is incorrect. This illustrates the fundamental limitation: StepFlow repairs *how* information flows, not *what* the model generates.