GRADCUIT: CREDIT-ASSIGNED GRADIENT FLOW ENABLES ROBUST AND INTERPRETABLE TEST-TIME LATENT REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Optimization-based latent reasoning improves large language model outputs by optimizing instance-specific continuous states at test time while keeping model parameters frozen. Existing methods, however, typically connect these states to the reasoning trajectory through decoded tokens, making sequence-level credit assignment indirect and obscuring how latent updates shape subsequent reasoning. We introduce GRADCUIT (*gradient through circuit*), which inserts optimizable latent states at a selected Transformer layer between the hidden representations of the prompt and the generated continuation. Causal self-attention provides every continuation-token log-probability with a differentiable path to every preceding latent state through the remaining Transformer blocks, enabling reward-weighted gradients from the entire continuation to be assigned directly to the latents. Across five instruction-tuned backbones, three reasoning benchmarks, and two answer formats, GRADCUIT achieves an average accuracy of 64.5%, outperforming chain-of-thought prompting by 6.6 percentage points and the strongest competing method by 2.4 points. GRADCUIT also demonstrates greater **robustness**: across seven learning-rate settings, it consistently outperforms LatentSeek while reducing the standard deviation of accuracy from 1.53 to 0.82, and even its random-walk variant remains competitive with LatentSeek. For **interpretability**, token-level gradient attribution reveals that latent influence concentrates on reasoning-connector tokens, while layer analysis identifies early-to-middle Transformer layers as the most effective optimization space. By directly optimizing internal reasoning from outcome feedback, GRADCUIT opens a new axis of robust and interpretable test-time scaling, where LLMs adapt how they reason rather than merely regenerate, sample, or rerank outputs.

1 INTRODUCTION

A growing line of work explores augmenting large language models (LLMs) with *latent reasoning* (Hao et al., 2025; Li et al., 2025; Deng et al., 2024), where continuous latent variables serve as intermediate states beyond explicit chain-of-thought tokens (Wei et al., 2022). Recent reasoning-as-optimization methods, including LatentSeek (Li et al., 2025), LTPO (Ye et al., 2026), and MILR (Mi et al., 2026), optimize instance-specific latents at test time to improve generation quality without updating model parameters. However, since they rely on decoded tokens as the interface between latent variables and the reasoning path, optimization is restricted to policy gradients of token-level objectives, which can only reach the latents by backpropagating through the decoding process. This designation creates a fundamental **credit-assignment** challenge: it remains difficult to determine how latent variables contribute to subsequent reasoning tokens and the final answer.

To be specific, this challenge manifests in two aspects: (1) **Indirect optimization**: the decoding process introduces an information bottleneck between latent variables and their downstream effects, causing optimization signals to become indirect and entangled with intermediate token representations; (2) **Opaque latent dynamics**: the discrete generation pathway obscures how individual latent variables influence subsequent predictions, making it difficult to interpret a particular latent update.

We propose GRADCUIT, which takes a step toward addressing this credit-assignment challenge by leveraging the Transformer's self-attention mechanism as a *computational circuit* for latent reasoning. Our design choice is to introduce latents directly into the model's *intermediate hidden space*,

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.

allowing the self-attention circuit to serve as both the forward pathway for computation and the backward pathway for gradient propagation during optimization. This designation is related in spirit to the *Transformer Circuits* viewpoint (Elhage et al., 2021): the attention graph defines a compositional circuit of interactions, hence the name GRADCUIT (*gradient through circuit*). GRADCUIT enables **direct optimization** of the latents through self-attention, while simultaneously offering **a lens into the latent dynamics** [1] via the Jacobian of the subsequent tokens with respect to the latents.

Extensive experiments across five backbones and three reasoning benchmarks demonstrate the strong and consistent performance of GRADCUIT. Averaged across all backbones, benchmarks, and answer formats, GRADCUIT improves accuracy over the standard CoT baseline by 6.6 percentage points. It also outperforms all four enhanced reasoning baselines, achieving an overall average accuracy of 64.5%, compared with 62.1% for the strongest competing method. GRADCUIT further exhibits greater **robustness**: across seven learning-rate settings, it consistently outperforms LatentSeek while reducing the standard deviation in accuracy from 1.53 to 0.82. Even when its policy-gradient updates are replaced by a random walk, GRADCUIT remains competitive with LatentSeek (Li et al., 2025), demonstrating the robustness of its latent space to the choice of optimization direction. Finally, this circuit-like gradient pathway enables direct inspection of latent dynamics, revealing the important role of reasoning-connector tokens and thereby improving the **interpretability** of latent reasoning.

Our research contributions and findings are summarized as follows:

- **Circuit-like latent reasoning through self-attention.** We introduce GRADCUIT, which inserts a small set of learnable latent states at an intermediate Transformer layer, enabling self-attention to provide a direct computational pathway for both forward token–latent interaction and backward credit assignment.
- **Improved robustness through direct credit assignment.** By propagating reward-weighted gradients directly from generated tokens to the latent states, GRADCUIT achieves greater optimization robustness than previous latent-reasoning methods.
- **Effective and interpretable latent dynamics.** Across five backbones and three reasoning benchmarks, GRADCUIT improves accuracy over CoT by 6.6 percentage points and outperforms the strongest enhanced reasoning baseline by 2.4 points on average. Its circuit-like gradient pathway also makes latent dynamics more interpretable, revealing the important role of reasoning-connector tokens in latent reasoning.

2 CIRCUIT-LIKE GRADIENT FLOW FOR LATENT REASONING

In this section, we present the GRADCUIT algorithm.

2.1 PRELIMINARIES: OUTPUT-SIDE LATENT OPTIMIZATION

Let π be an autoregressive language model and R a scalar reward function. For an input problem $\mathbf{c}$, let $\mathbf{x} = (x_1, x_2, \ldots, x_T)$ denote a length-T token sequence. As a representative output-side formulation, LatentSeek (Li et al., 2025) introduces latent variables $\mathbf{z} = (z_1, z_2, \ldots, z_N)$, where each z_t lies in the latent space associated with x_t and $N \leq T$. Its ideal test-time objective is to maximize the expected reward:

$$\mathbf{z}^* = \arg\max_{\mathbf{z}} \; \mathbb{E}_{\mathbf{x}\sim\pi(\mathbf{x}|\mathbf{z},\mathbf{c})}\big[R(\mathbf{x},\mathbf{c})\big]. \tag{1}$$

The conditional generation distribution $\pi(\mathbf{x} \mid \mathbf{z}, \mathbf{c})$ is factorized as

$$\pi(\mathbf{x} \mid \mathbf{z}, \mathbf{c}) = \underbrace{\prod_{t=1}^{N} \pi_\theta(x_t \mid z_t)}_{\text{Latents} \to \text{tokens}} \prod_{t=N+1}^{T} \pi(x_t \mid \mathbf{x}_{<t}, \mathbf{c}), \tag{2}$$

[1] Concurrent work by Anthropic, J-lens (Gurnee et al., 2026), explores a related idea. However, whereas their focus is on explainability, our goal is to improve latent reasoning.

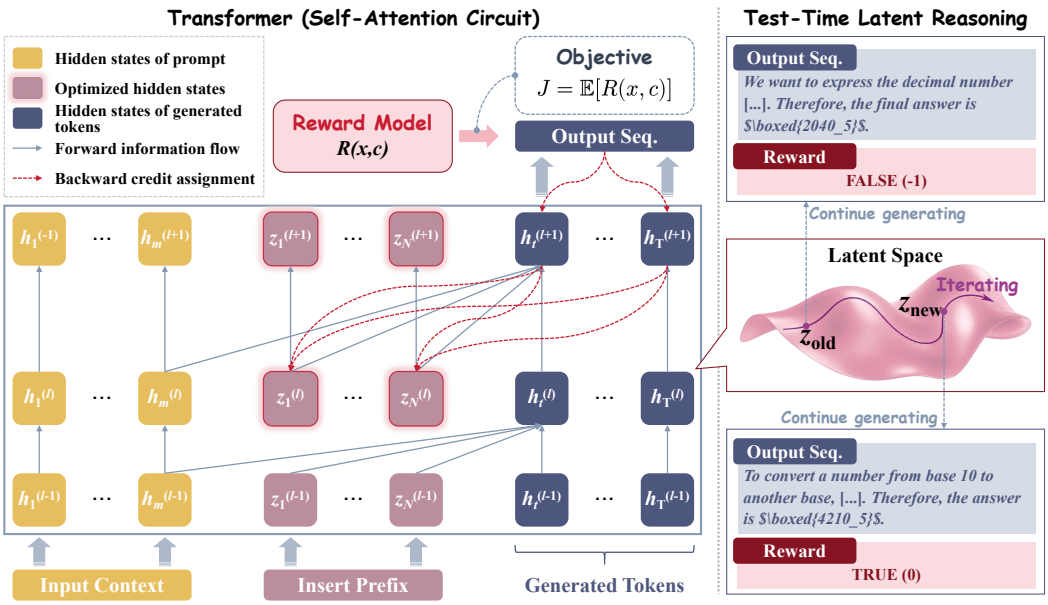


Figure 1: The GRADCUIT framework. The Transformer's self-attention mechanism functions as a gradient-routing circuit, enabling direct backward credit assignment from generated tokens to the latents along attention pathways. The red feedback loop visualizes the policy gradient update ($z \leftarrow z + \nabla_z \mathcal{J}$), used to iteratively refine the latents during inference without updating the base model parameters.

where θ parameterizes the language-model head (LM head).

Conceptually, LatentSeek first decodes the latent variables $\mathbf{z}$ into the first N tokens, and then performs standard autoregressive generation conditioned on this decoded prefix. After producing the full sequence $\mathbf{x}$ and evaluating its reward $R(\mathbf{x}, \mathbf{c})$, LatentSeek updates the latents by backpropagating gradients through the logits associated with these decoded tokens. Concretely, the policy-gradient term for the i-th latent takes the form

$$\mathbb{E}_{\mathbf{x}\sim\pi(\mathbf{x}|\mathbf{z},\mathbf{c})}[R(\mathbf{x},\mathbf{c})\nabla_{z_i} \log \pi(x_i \mid z_i)]. \tag{3}$$

2.2 GRADIENT FLOW THROUGH TRANSFORMER CIRCUITS

To precisely attribute the influence of individual latent variables and avoid the information loss introduced by decoding continuous latents into discrete tokens, we propose a new method, GRADCUIT, for solving Equation (1). Rather than defining the latent space at the transformer's *output*, as in Li et al. (2025), we define it within an *intermediate* hidden-state space. This design directly incorporates the pre-trained self-attention mechanism into the latent optimization process.

For an M-block decoder, we append the separately tokenized fixed prefix s ("Let's think about this problem and solve it step by step.") to the chat-formatted prompt and initialize its N latent states from the prefix representations after the first l blocks, where N is the prefix token count and $l = \lfloor M/2 \rfloor$ in our experiments. To predict continuation token x_t, we process $[\mathbf{c}, s, \mathbf{x}_{<t}]$ through these blocks and replace only the prefix span with $\mathbf{z}^{(l)}$, its initial states plus a zero-initialized trainable offset, keeping model parameters, sequence positions, and the causal mask unchanged.

The remaining Transformer blocks, final normalization, and language-model head produce the next-token distribution:

$$\begin{aligned} H^{(M)} &= \text{Transformer}^{l+1:M}\big([h_{\mathbf{c}}^{(l)}, \mathbf{z}^{(l)}, h_{\mathbf{x}_{<t}}^{(l)}]\big), \\ \pi(x_t \mid \mathbf{x}_{<t}, \mathbf{z}^{(l)}, \mathbf{c}) &= \text{softmax}\Big(\text{LM_Head}\Big(\text{Norm}(H_{-1}^{(M)})\Big)\Big)_{x_t}. \end{aligned} \tag{4}$$

Here, $H_{-1}^{(M)}$ denotes the final sequence position (the last prefix position for $t = 1$); the fixed prefix s is omitted from the conditioning notation.

This construction induces **a new autoregressive factorization** that replaces Equation (2):

$$\pi(\mathbf{x} \mid \mathbf{z}^{(l)}, \mathbf{c}) = \prod_{t=1}^{T} \pi(x_t \mid \mathbf{x}_{<t}, \mathbf{z}^{(l)}, \mathbf{c}). \tag{5}$$

Because the **latent variables participate in self-attention alongside the token representations**, each generated token can attend to all preceding latent positions. Consequently, gradient signals from every generated token can propagate directly to every latent variable through the remaining self-attention layers. This token-to-latent gradient flow is characterized as

$$\nabla_{z_i^{(l)}} \pi(x_t \mid \mathbf{x}_{<t}, \mathbf{z}^{(l)}, \mathbf{c}),$$

which measures the sensitivity of the probability of the t-th token to the i-th latent variable. We accordingly perform the following policy-gradient-style update, as in LatentSeek (Li et al., 2025):

$$\mathbf{z}^{(l)} \leftarrow \mathbf{z}^{(l)} + \eta \nabla_{\mathbf{z}^{(l)}} \mathcal{J}(\mathbf{z}^{(l)}), \tag{6}$$

where η denotes the latent optimization step size. For fair comparison with other methods, GRADCUIT uses *self-reward*: the frozen backbone judges only the original question and extracted final answer, without access to ground-truth answers or the generated reasoning. We use its verdict as a detached scalar reward of 0 for acceptance and -1 otherwise, including answer-extraction or verdict-parsing failures, and stop optimization upon acceptance. The gradient associated with each latent variable aggregates contributions from all generated token positions:

$$\nabla_{z_i^{(l)}} \mathcal{J} = \sum_{t=1}^{T} \mathbb{E}_{\mathbf{x} \sim \pi(\mathbf{x} \mid \mathbf{z}^{(l)}, \mathbf{c})} \Big[R(\mathbf{x}, \mathbf{c}) \times \nabla_{z_i^{(l)}} \log \pi(x_t \mid \mathbf{x}_{<t}, \mathbf{z}^{(l)}, \mathbf{c}) \Big]. \tag{7}$$

In summary, **gradient signals flow from the generated discrete tokens back to each latent variable through the transformer's self-attention connectivity**. Motivated by this circuit-like pathway for gradient propagation, we refer to the resulting latent reasoning method as GRADCUIT.

3 EXPERIMENTS

3.1 EXPERIMENTAL SETUP

We evaluate GRADCUIT against CoT, our one-pass Self-Reflection baseline, Self-Consistency, Self-Scored Best-of-N (BoN), and LATENTSEEK (Li et al., 2025) to assess its effectiveness relative to representative explicit, sampling-based, and latent reasoning methods. Complete baseline protocols and prompts are provided in the Appendix. Experiments are conducted using five instruction-tuned backbones: LLaMA-3.2-3B-Instruct,[2] LLaMA-3.1-8B-Instruct,[3] Qwen2.5-7B-Instruct,[4] Qwen2.5-14B-Instruct,[5] and Qwen3-4B-Instruct-2507.[6] We evaluate all methods on GPQA-Diamond (Rein et al., 2024), GSM8K (Cobbe et al., 2021), and MATH-500 (Lightman et al., 2024; Hendrycks et al., 2021), using both `\boxed` and JSON answer formats. Full details are provided in the Appendix.

3.2 MAIN RESULTS

Overall Effectiveness. Table 1 summarizes the main results. Averaged over all 30 backbone–benchmark–format settings, GRADCUIT achieves **64.5%** accuracy, outperforming CoT by **6.6 points** and the strongest competing method by **2.4 points**. It attains the highest average accuracy under both answer formats on all three benchmarks and achieves the best individual result in **23 of**

[2] https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct
[3] https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct
[4] https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
[5] https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
[6] https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507

Backbone	LLaMA3.2-3B		LLaMA3.1-8B		Qwen2.5-7B		Qwen2.5-14B		Qwen3-4B		Avg.	
Prompt Type	Boxed	JSON	Boxed	JSON	Boxed	JSON	Boxed	JSON	Boxed	JSON	Boxed	JSON
GPQA-Diamond												
CoT	17.7	18.7	20.7	22.2	31.3	31.8	40.4	42.4	40.9	40.4	30.2	31.1
Self-Reflection	12.6	25.3	14.1	25.8	33.8	31.8	36.9	45.5	**58.6**	54.6	31.2	36.6
Self-Consistency	23.2	25.3	25.8	28.8	34.9	34.9	41.4	39.9	50.5	**55.1**	35.2	36.8
Self-Scored BoN	18.2	24.2	15.7	26.3	33.3	**37.9**	41.4	**46.5**	46.5	48.0	31.0	36.6
LATENTSEEK	25.3	20.7	27.8	29.3	33.3	32.3	**47.5**	42.9	50.0	51.0	36.8	35.2
GRADCUIT (Ours)	**30.3**	**27.8**	**31.3**	**30.3**	**38.4**	35.4	42.4	42.9	52.5	49.5	**39.0**	**37.2**
GSM8K												
CoT	75.4	68.2	81.7	76.6	89.2	79.9	92.4	89.6	88.1	87.3	85.4	80.3
Self-Reflection	67.9	58.5	70.9	67.3	89.2	79.4	93.2	89.1	**92.0**	88.6	82.6	76.6
Self-Consistency	79.5	69.7	84.0	84.0	88.2	81.4	92.2	90.6	89.2	88.7	86.6	82.9
Self-Scored BoN	77.9	64.6	83.4	75.7	89.2	76.3	92.6	89.5	89.9	88.2	86.6	78.9
LATENTSEEK	77.8	68.8	82.6	78.2	90.8	79.1	92.1	90.1	88.0	88.6	86.3	81.0
GRADCUIT (Ours)	**82.5**	**75.7**	**86.2**	**84.5**	**91.8**	**82.1**	**93.3**	**92.6**	90.0	**88.9**	**88.8**	**84.8**
MATH-500												
CoT	40.4	40.0	50.6	46.8	73.0	43.4	79.2	59.8	87.2	79.8	66.1	54.0
Self-Reflection	35.4	38.4	38.4	40.2	74.6	47.0	76.4	62.0	84.4	81.6	61.8	53.8
Self-Consistency	45.2	**48.0**	46.4	50.8	73.8	68.4	80.0	68.4	89.0	84.6	66.9	64.0
Self-Scored BoN	45.8	42.0	48.0	45.2	74.4	64.2	78.8	66.8	90.0	83.4	67.4	60.3
LATENTSEEK	47.6	40.2	56.8	51.0	76.4	45.2	78.2	62.6	87.0	81.4	69.2	56.1
GRADCUIT (Ours)	**53.6**	47.4	**57.4**	**53.6**	**77.2**	**68.6**	**80.2**	**68.6**	**91.8**	**86.8**	**72.0**	**65.0**

Table 1: Accuracy comparison across different reasoning settings. GRADCUIT denotes our method.

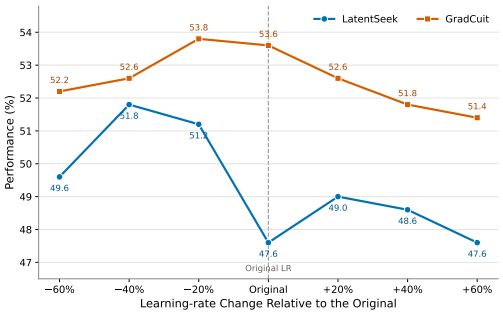


Figure 2: Learning-rate sensitivity of LATENTSEEK and GRADCUIT on MATH-500 using LLaMA-3.2-3B-Instruct.

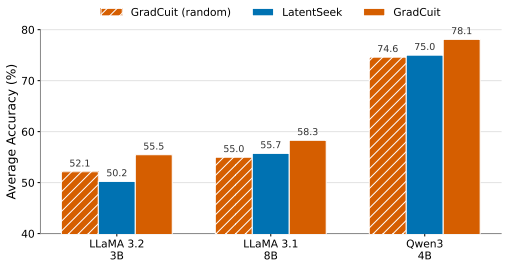


Figure 3: Average accuracy across three representative backbones. GRADCUIT (random) uses randomly sampled update directions.

30 settings. GRADCUIT also consistently outperforms Self-Consistency and Self-Scored BoN in all six benchmark–format aggregates. Moreover, its average number of optimization iterations is lower than the number of sampled responses used by these methods in every setting; additional statistics are reported in the Appendix.

Comparison with LatentSeek. GRADCUIT improves over LATENTSEEK in every benchmark–format aggregate. Under the Boxed/JSON formats, the respective gains are **2.2/2.0 points** on GPQA-Diamond, **2.5/3.8 points** on GSM8K, and **2.8/8.9 points** on MATH-500. These consistent gains support the benefit of aggregating continuation-token gradients through the remaining Transformer computation, rather than restricting each latent's update to the log-probability of its associated decoded prefix token.

Token category	Representative tokens
Formatting	`.`, `,`, `#`, `-`
Reasoning Connector	`because`, `therefore`, `then`, `however`
Content Explanation	`compute`, `equation`, `value`, `number`
Answer Marker	`\boxed`, `####`, `final`, `answer`
Answer Content	`42`, `C`, `3.14`, `7/8`

Table 2: Continuation-token categories used in the gradient attribution analysis.

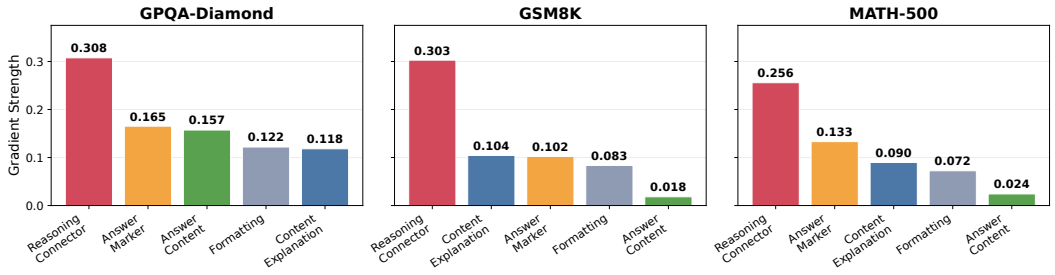


Figure 4: Gradient strength across continuation-token categories on three reasoning benchmarks.

3.3 Robustness Gain

We examine the robustness of latent optimization from two complementary perspectives: sensitivity to the learning rate and dependence on reward guidance. For the learning-rate analysis, we evaluate GradCuit and LatentSeek using LLaMA-3.2-3B-Instruct on MATH-500 with the Boxed answer format. We scale their respective base learning rates, 0.001 and 0.03, by $\{0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6\}$ while holding all other settings fixed. For the gradient-direction analysis, we replace the reward-derived gradients in GradCuit with Gaussian random directions at each optimization step, resulting in a random-walk variant without reward guidance. We report Boxed accuracy averaged over GPQA-Diamond, GSM8K, and MATH-500 using LLaMA-3.2-3B-Instruct, LLaMA-3.1-8B-Instruct, and Qwen3-4B-Instruct.

Robustness to Learning Rates. As shown in Figure 2, GradCuit maintains strong performance across all seven learning-rate settings. Its accuracy varies only from **51.4% to 53.8%**, compared with 47.6% to 51.8% for LatentSeek. It also achieves both a higher average accuracy (**52.6%** versus 49.3%) and a substantially lower standard deviation (**0.82** versus 1.53). These results indicate that direct interaction with selected-layer latent states provides a more stable optimization interface that is less sensitive to step-size selection.

Robustness to Optimization Directions. As shown in Figure 3, even the random-walk variant of GradCuit achieves an average accuracy of **60.6%**, slightly exceeding the 60.3% of reward-guided LatentSeek. Thus, useful reasoning trajectories can be discovered through direct exploration of the selected-layer latent space even without an explicitly optimized direction. This comparison separates two sources of improvement: the direct latent interaction introduced by GradCuit makes the optimization space inherently more accessible and robust, while reward guidance determines how effectively that space is explored.

3.4 Latent Dynamics: Token-Level Gradient Attribution Analysis

We conduct the analysis using LLaMA-3.2-3B-Instruct on GPQA-Diamond, GSM8K, and MATH-500 with the Boxed answer format. Gradients are computed from the generated trajectory at the final latent-optimization step. For each continuation token, we define its gradient strength as the L_2 norm of its gradient with respect to all optimized latent states. We then average the gradient strength over tokens assigned to the same category and across evaluation examples. Tokens are automatically classified using the rule-based categories summarized in Table 2.

Method	LLaMA-3.2-3B			LLaMA-3.1-8B			Qwen2.5-7B			Qwen2.5-14B			Qwen3-4B		
	G	S	M	G	S	M	G	S	M	G	S	M	G	S	M
GRADCUIT	**30.3**	**82.5**	**53.6**	**31.3**	**86.2**	**57.4**	**38.4**	**91.8**	**77.2**	**42.4**	**93.3**	**80.2**	**52.5**	**90.0**	**91.8**
w/o Gradient	26.3 (-4.0)	80.7 (-1.8)	49.4 (-4.2)	25.3 (-6.0)	85.4 (-0.8)	54.2 (-3.2)	37.9 (-0.5)	89.5 (-2.3)	75.8 (-1.4)	**42.4** (-0.0)	93.0 (-0.3)	78.8 (-1.4)	48.0 (-4.5)	89.6 (-0.4)	86.2 (-5.6)
w/o Latent Update and the above	23.2 (-7.1)	78.5 (-4.0)	47.6 (-6.0)	20.2 (-11.1)	82.6 (-3.6)	49.0 (-8.4)	36.4 (-2.0)	89.2 (-2.6)	72.2 (-5.0)	41.4 (-1.0)	92.3 (-1.0)	78.4 (-1.8)	43.4 (-9.1)	89.3 (-0.7)	86.8 (-5.0)
w/o Inserted Prefix and the above	17.7 (-12.6)	75.4 (-7.1)	40.4 (-13.2)	20.7 (-10.6)	81.7 (-4.5)	50.6 (-6.8)	31.3 (-7.1)	89.2 (-2.6)	73.0 (-4.2)	40.4 (-2.0)	92.4 (-0.9)	79.2 (-1.0)	40.9 (-11.6)	88.1 (-1.9)	87.2 (-4.6)

Table 3: Ablation results across five instruction-tuned backbones. We report accuracy (%). Within each backbone, G, S, and M denote GPQA-Diamond, GSM8K, and MATH-500, respectively.

Reasoning connectors receive the strongest gradients. As shown in Figure 4, reasoning connectors consistently exhibit the highest gradient strength across all three benchmarks. Tokens such as *because*, *therefore*, and *then* mark transitions between successive reasoning steps. Their consistently high first-order sensitivity indicates that optimized latent states primarily influence how the model connects and advances its reasoning process, rather than affecting continuation tokens uniformly.

4 FURTHER ANALYSIS

In this section, we perform deeper analysis on GRADCUIT.

4.1 ABLATION STUDY

We conduct the ablation study across five backbones and three benchmarks using the Boxed answer format. We compare the full GRADCUIT with three variants: *w/o Gradient* replaces the reward-derived gradients with Gaussian random directions, *w/o Latent Update* retains the inserted prefix without updating it, and *w/o Inserted Prefix* reduces the method to standard CoT. The results are presented in Table 3.

The inserted prefix alone is insufficient. Averaged over all 15 backbone–benchmark settings, inserting a fixed prefix improves accuracy from **60.5%** to **62.0%**, but degrades performance in six settings and ties CoT in one. In contrast, the full GRADCUIT reaches **66.6%** and outperforms the fixed-prefix variant in all 15 settings by an average of **4.6 points**. The improvement therefore cannot be explained solely by the additional textual reasoning cue.

Reward guidance provides the decisive gain. Random Optimization outperforms the fixed-prefix variant in **14 of 15 settings**, indicating that modifying selected-layer latent states can expose useful alternative reasoning trajectories. Replacing random directions with reward-derived gradients provides a further average improvement of **2.4 points**: GRADCUIT outperforms random optimization in 14 settings, ties it in one, and is best or tied-best in every column. These results indicate that selected-layer optimization and reward-guided credit assignment provide complementary benefits.

4.2 ANALYSIS OF OPTIMIZED LAYERS

We investigate how the layer at which the latent prefix is optimized affects the performance of GRADCUIT. Using LLaMA-3.2-3B-Instruct as the backbone, we place the optimizable prefix at four normalized depths.

As shown in Figure 5, **optimizing the latent prefix at an intermediate layer generally outperforms optimization in the embedding space**. On GPQA-Diamond, the 25% and 50% placements both achieve the highest accuracy of 30.3%, compared with 29.8% at the embedding level. The benefit is most pronounced on MATH-500, where moving the latent toward the middle of the network yields the largest improvement.

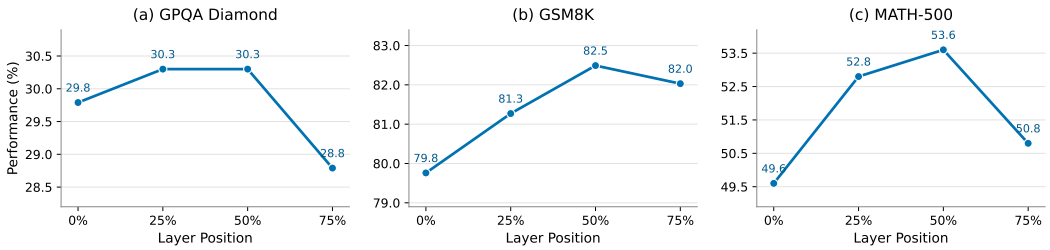


Figure 5: Sensitivity of GRADCUIT to the optimized layer on LLaMA-3.2-3B-Instruct. Performance is reported on (a) GPQA-Diamond, (b) GSM8K, and (c) MATH-500. The horizontal axis indicates the normalized depth at which the latent is optimized.

Question. Find the unique **odd** integer t such that $0 < t < 23$ and $t + 2$ is the inverse of t modulo 23.	
LATENTSEEK	
Original	*We seek the unique integer t such that $0 < t < 23$ and $t + 2$ is the multiplicative inverse of [...] The final answer is $\boxed{19}$.*
Optimized	*We solve unique integer t, that that $0 < t < 23$ and $t+2$ is the inverse of [...] modulo 23 is $\boxed{20}$.*
GRADCUIT (Ours)	
Original	*We are given that $t+2$ is the inverse of t modulo 23. [...] t modulo 23 is $\boxed{20}$.*
Optimized	*We are given that $t + 2$ is the inverse of t modulo 23. [...] The answer is $t = nboxed\{17\}$.*

Table 4: A representative case comparing the outputs before and after latent optimization. All model outputs are displayed verbatim, and the ground-truth answer is 17.

The optimal layer is therefore task-dependent, but the strongest results consistently occur between 25% and 50% of the network depth. Moving the optimization point further to 75% does not provide an additional gain and can be detrimental, most notably on GPQA-Diamond. These results are consistent with a favorable trade-off at early-to-middle depths: **the hidden states are contextualized enough to provide informative optimization signals while retaining sufficient downstream Transformer computation to propagate and refine the effect of the optimized latent**.

4.3 CASE STUDY

To qualitatively investigate how test-time latent optimization affects model reasoning, we present a representative example in Table 4. We compare the outputs before and after optimization by LatentSeek and GRADCUIT.

As shown in Table 4, LatentSeek changes the decoded token sequence during optimization but still produces an incorrect answer and introduces malformed text such as "that that." In contrast, GradCuit directly updates the hidden states without explicitly modifying the token sequence, thereby avoiding such textual corruption while correcting the answer from 20 to 17. This case further illustrates the greater robustness of GradCuit.

5 RELATED WORK

Test-Time Optimization. A broad set of methods improve language-model outputs *at inference time* by allocating additional computation. Discrete approaches include repeated sampling and aggregation, such as self-consistency (Wang et al., 2023), as well as recitation-augmented generation (Sun et al., 2023) and explicit trajectory search (Hao et al., 2023). Continuous inference-time steering includes PPLM, which updates internal activations using gradients from a differentiable attribute model while keeping the base language model frozen (Dathathri et al., 2020). Amulet instead formu-

lates each token-decoding step as an online optimization problem for test-time preference adaptation (Zhang et al., 2025). CTRL is related to controllable generation, but its control-code conditioning is learned during pre-training rather than optimized at test time (Keskar et al., 2019). Test-Time Training (TTT) updates model parameters or fast model state online using self-supervised objectives (Sun et al., 2020; 2025; Hardt & Sun, 2024). In contrast, GRADCUIT keeps all model parameters fixed and optimizes only a small set of instance-specific states inserted at a selected Transformer layer.

Reinforcement Learning for Language Models. Reinforcement learning from human feedback is exemplified by InstructGPT (Ouyang et al., 2022), whereas Constitutional AI uses reinforcement learning from AI feedback (Bai et al., 2022). Policy-gradient methods such as PPO are widely used in this setting (Schulman et al., 2017). More recent alignment objectives and algorithms include Direct Preference Optimization (Rafailov et al., 2023) and Statistical Rejection Sampling (Liu et al., 2024a). Group Relative Policy Optimization was introduced in DeepSeekMath (Shao et al., 2024) and subsequently used at scale in DeepSeek-R1 (Guo et al., 2025). In parallel, reward modeling has evolved from human preference models to automated and language-model-driven reward design (Kwon et al., 2023), as well as multi-agent verification frameworks (Lifshitz et al., 2025). A related variational perspective is explored by Chen et al. (2024), which improves latent trajectory fitting by updating model parameters. Compared with these training-time approaches, our method operates entirely at test time: it does not update the policy model, but instead uses gradients of token-level objectives to refine instance-specific latent states.

Latent Reasoning. Prompting-based reasoning methods such as chain-of-thought prompting (Wei et al., 2022; Kojima et al., 2022; Zhou et al., 2023) elicit intermediate reasoning steps in text. Compute-adaptive strategies further adjust inference effort based on task complexity (Snell et al., 2025; Inoue et al., 2025). Motivated by the limitations of explicit textual traces, latent chain-of-thought methods replace or compress reasoning traces into continuous representations (Hao et al., 2025; Shen et al., 2025; Cheng & Van Durme, 2024; Deng et al., 2024). Prompt tuning and soft/prefix prompting (Lester et al., 2021; Liu et al., 2024b; Li & Liang, 2021; Liu et al., 2022) also inject trainable vectors, but they typically require supervised data and training-time backpropagation through the model. Our work is closest to test-time latent-optimization methods. LatentSeek optimizes output-side latent representations that are decoded into token sequences for reward evaluation (Li et al., 2025). LTPO optimizes input-level latent thought vectors using a confidence reward and perturbation-based policy gradients (Ye et al., 2026). MILR searches intermediate text–image output representations for multimodal generation (Mi et al., 2026), while DMLR refines latent think tokens using confidence-guided policy gradients and dynamic visual injection (Liu et al., 2026). GradCuit instead inserts optimizable states at a selected Transformer layer and directly differentiates continuation-token log-probabilities with respect to these states through the remaining Transformer computation.

6 CONCLUSION

We study credit assignment in test-time latent reasoning: propagating sequence-level rewards to instance-specific latent states. GRADCUIT realizes *Credit-Assigned Gradient Flow* through optimizable latent states at a selected Transformer layer. This makes every continuation-token log-probability differentiable with respect to every latent state, enabling direct reward-weighted optimization with the model frozen. Across five backbones and three reasoning benchmarks, GRADCUIT achieves the highest average accuracy among all evaluated methods while exhibiting substantially lower sensitivity to learning-rate selection than LatentSeek. Even when reward-derived gradients are replaced with random directions, GRADCUIT remains competitive with LatentSeek, supporting the robustness of this latent space. Token-level gradient attribution shows that latent influence concentrates on reasoning-connector tokens, while ablation studies establish the complementary roles of the inserted prefix, latent-state optimization, and reward guidance. Additional analyses identify early-to-middle Transformer layers as the most effective optimization space and illustrate, through a representative case, that direct hidden-state updates can correct an erroneous answer without introducing malformed text. Together, these findings demonstrate that GRADCUIT enables **more robust and interpretable** test-time latent reasoning.

AI USE STATEMENT

Our use of LLM-based research assistance was limited to polishing the language of the manuscript and implementing auxiliary utilities, such as plotting and visualization scripts. The research conception, methodological design, mathematical derivations, and implementation of the core algorithms and experimental logic were completed entirely by the human authors, without generative AI assistance. All scientific decisions and judgments remained with the authors, who take full responsibility for the paper and its accompanying code, including all AI-assisted material.

ETHICS STATEMENT

This work investigates test-time reasoning using existing instruction-tuned language models and established mathematical and scientific reasoning benchmarks. The reported study is confined to computational experiments on existing benchmark data and does not involve human-subject studies or the collection of personal data. We do not anticipate significant ethical concerns specific to this research setting. To support responsible interpretation of the findings and reduce the risk of overgeneralizing performance gains, we scope our empirical claims to the evaluated reasoning tasks and answer formats, report comparisons across multiple backbones and benchmarks, and describe the experimental settings. We also provide the experimental code in the supplementary materials to enable independent scrutiny of the reported results.

REPRODUCIBILITY STATEMENT

The experimental code used in this study is included in the supplementary materials submitted with this paper. The main paper describes the proposed method and specifies the evaluated backbones, benchmarks, and comparison settings. We will also make the code publicly available on GitHub to facilitate independent reproduction of the reported results and further research.

REFERENCES

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *CoRR*, 2022.

Haolin Chen, Yihao Feng, Zuxin Liu, Weiran Yao, Akshara Prabhakar, Shelby Heinecke, Ricky Ho, Phil Mui, Silvio Savarese, Caiming Xiong, and Huan Wang. Language models are hidden reasoners: Unlocking latent reasoning capabilities via self-rewarding, 2024. URL https://arxiv.org/abs/2411.04282.

Jeffrey Cheng and Benjamin Van Durme. Compressed chain of thought: Efficient reasoning through dense representations. *CoRR*, 2024.

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

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. In *The Eighth International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=U3uGMztyH5S2.

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

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z. URL https://www.nature.com/articles/s41586-025-09422-z.

Wes Gurnee, Nicholas Sofroniew, Adam Pearce, Mateusz Piotrowski, Isaac Kauvar, Runjin Chen, Anna Soligo, Paul Bogdan, Euan Ong, Rowan Wang, Ben Thompson, David Abrahams, Subhash Kantamneni, Emmanuel Ameisen, Joshua Batson, and Jack Lindsey. Verbalizable representations form a global workspace in language models. *Transformer Circuits Thread*, 2026. URL https://transformer-circuits.pub/2026/workspace/index.html.

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 *Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8154–8173, 2023.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason E. Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Itxz7S4Ip3.

Moritz Hardt and Yu Sun. Test-time training on nearest neighbors for large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

Yuichi Inoue, Kou Misaki, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. Wider or deeper? scaling LLM inference-time compute with adaptive branching tree search. In *Advances in Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=jAsr5GHt3P.

Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *CoRR*, 2019.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

Hengli Li, Chenxi Li, Tong Wu, Xuekai Zhu, Yuxuan Wang, Zhaoxin Yu, Eric Hanchen Jiang, Song-Chun Zhu, Zixia Jia, Ying Nian Wu, and Zilong Zheng. Seek in the dark: Reasoning via test-time instance-level policy gradient in latent space, 2025. URL https://arxiv.org/abs/2505.13308.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.

Shalev Lifshitz, Sheila A McIlraith, and Yilun Du. Multi-agent verification: Scaling test-time compute with multiple verifiers. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=LriQ3NY9uL.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024.

Chengzhi Liu, Yuzhe Yang, Yue Fan, Qingyue Wei, Sheng Liu, and Xin Eric Wang. Reasoning within the mind: Dynamic multimodal interleaving in latent space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Findings*, pp. 9225–9236, 2026. URL https://openaccess.thecvf.com/content/CVPR2026F/html/Liu_Reasoning_Within_the_Mind_Dynamic_Multimodal_Interleaving_in_Latent_Space_CVPRF_2026_paper.html.

Tianqi Liu, Yao Zhao, Rishabh Joshi, Misha Khalman, Mohammad Saleh, Peter J. Liu, and Jialu Liu. Statistical rejection sampling improves preference optimization. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2024a. URL https://openreview.net/forum?id=xbjSwwrQOe.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 61–68. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.acl-short.8. URL https://aclanthology.org/2022.acl-short.8/.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. Gpt understands, too. *AI Open*, 5:208–215, 2024b.

Yapeng Mi, Yanpeng Zhao, Hengli Li, Chenxi Li, Huimin Wu, Xiaojian Ma, Song-Chun Zhu, Ying Nian Wu, and Qing Li. MILR: Improving multimodal image generation via test-time latent reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=dw0pORtnKI.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:27730–27744, 2022.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2023.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, 2024. URL https://arxiv.org/abs/2402.03300.

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. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.36. URL https://aclanthology.org/2025.emnlp-main.36/.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations (ICLR)*, 2025.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International Conference on Machine Learning (ICML)*, 2020.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *International Conference on Machine Learning (ICML)*, 2025.

Zhiqing Sun, Xuezhi Wang, Yi Tay, Yiming Yang, and Denny Zhou. Recitation-augmented language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:24824–24837, 2022.

Wengao Ye, Yan Liang, and Lianlei Shan. Thinking on the fly: Test-time reasoning enhancement via latent thought policy optimization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=r1WEQzkCQv.

Zhaowei Zhang, Fengshuo Bai, Qizhi Chen, Chengdong Ma, Mingzhi Wang, Haoran Sun, Zilong Zheng, and Yaodong Yang. Amulet: Realignment during test time for personalized preference adaptation of LLMs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=f9w89OY2cp.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WZH7099tgfM.

A Appendix

A Implementation Details

A.1 Details of GradCuit

Latent Construction and Deployment. For each problem, we first render the system and user messages using the native Hugging Face chat template of the corresponding backbone. We then append the fixed textual prefix, "Let's think about this problem and solve it step by step.", after the rendered prompt and before the generated continuation. The number of inserted latent states, denoted by N, is therefore determined by the number of tokens into which each backbone tokenizer decomposes this prefix. The optimized layer l is a method hyperparameter that determines the decoder-block input at which the prefix representations are treated as optimizable latent states. For a model with M decoder blocks, we set $l = \lfloor M/2 \rfloor$ in the main experiments to obtain a consistent representative configuration across backbones. We additionally vary l in the layer-position analysis to examine how the location of the latent optimization space affects optimization performance. We initially pass the prompt and textual prefix through the first l Transformer blocks and extract the hidden representations corresponding to the prefix span, denoted by $\boldsymbol{z}_0^{(l)} \in \mathbb{R}^{N \times d}$. Rather than repeatedly encoding the textual prefix, we introduce a trainable offset $\Delta \boldsymbol{z}^{(l)}$, initialized to zero, and use

$$\widetilde{\boldsymbol{z}}^{(l)} = \boldsymbol{z}_0^{(l)} + \Delta \boldsymbol{z}^{(l)}$$

as the prefix representation throughout optimization. At the input to the selected decoder block, a forward pre-hook replaces the original prefix span with $\widetilde{\boldsymbol{z}}^{(l)}$. All prompt and continuation representations retain their original positions, and the standard causal attention mask is used without modification. Consequently, continuation tokens can attend to all inserted latent states while the model architecture and parameters remain unchanged. All model parameters are frozen, and gradients are computed only with respect to $\Delta \boldsymbol{z}^{(l)}$.

Test-Time Optimization. Before optimization, we greedily generate an initial continuation using $\boldsymbol{z}_0^{(l)}$ and evaluate its final answer with the self-reward verifier described below. At each optimization step, the currently generated token sequence $\boldsymbol{x} = (x_1, \ldots, x_T)$ is treated as fixed and passed through the model using teacher forcing. We disable the key–value cache during this gradient computation and collect the log-probability assigned to every continuation token. A single trajectory is used at each step to estimate the sequence-level objective, and the implementation minimizes

$$\mathcal{L}_{\text{opt}} = -R(\boldsymbol{x}, \boldsymbol{c}) \sum_{t=1}^{T} \log \pi \left(x_t \mid \boldsymbol{x}_{<t}, \widetilde{\boldsymbol{z}}^{(l)}, \boldsymbol{c} \right). \tag{8}$$

The token log-probabilities are summed without length normalization. The generated token IDs and their input embeddings are detached from the computational graph, as is the scalar reward, so the gradient propagates only from the continuation-token log-probabilities through the remaining Transformer blocks to the latent offset. We update $\Delta \boldsymbol{z}^{(l)}$ using Adam with a learning rate of 10^{-3}, the default coefficients $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. No weight decay, gradient clipping, latent-norm constraint, or additional regularization is applied. After every update, we greedily regenerate the complete continuation using the updated latent states and obtain a new reward. Optimization terminates immediately once the verifier accepts the final answer or after at most ten latent updates. If no answer is accepted within this budget, the continuation produced after the final update is returned.

[t] GradCuit inference for one problem instance. [1] Problem context $\boldsymbol{c}$ and frozen model π Prefix s, optimized layer l, learning rate η, and update budget K Render the model-specific chat prompt and append s Extract the prefix states $\boldsymbol{z}_0^{(l)}$ at the input to decoder block l Initialize $\Delta \boldsymbol{z}^{(l)} \leftarrow \mathbf{0}$ $\widetilde{\boldsymbol{z}}^{(l)} \leftarrow \boldsymbol{z}_0^{(l)} + \Delta \boldsymbol{z}^{(l)}$ Generate $\boldsymbol{x}$ greedily using $\widetilde{\boldsymbol{z}}^{(l)}$ Obtain $R(\boldsymbol{x}, \boldsymbol{c})$ from the verifier $k = 1, \ldots, K$ $R(\boldsymbol{x}, \boldsymbol{c}) = 0$ $\boldsymbol{x}$ Recompute continuation-token log-probabilities with teacher forcing Compute $\mathcal{L}_{\text{opt}}$ using Eq. equation 8 Update $\Delta \boldsymbol{z}^{(l)}$ by one Adam step with learning rate η $\widetilde{\boldsymbol{z}}^{(l)} \leftarrow \boldsymbol{z}_0^{(l)} + \Delta \boldsymbol{z}^{(l)}$ Regenerate $\boldsymbol{x}$ greedily using $\widetilde{\boldsymbol{z}}^{(l)}$ Re-evaluate $R(\boldsymbol{x}, \boldsymbol{c})$ $\boldsymbol{x}$

Self-Reward Verifier. Across all experiments that require a reward or verification signal, **the evaluated LLM itself** serves as the self-verifier and supplies the self-reward; no separate reward model is introduced. For each backbone, GradCuit, Self-Reflection, Self-Scored BoN, and LatentSeek all invoke that same backbone using an identical verifier prompt, answer-extraction procedure, and decision rule. The verifier receives only the original question and the extracted final answer. It evaluates final-answer correctness without scoring the completeness or quality of the generated reasoning. Its binary verdict is mapped to the discrete reward

$$R(\boldsymbol{x}, \boldsymbol{c}) = \begin{cases} 0, & \text{correct,} \\ -1, & \text{otherwise.} \end{cases}$$

An output from which no valid final answer can be extracted is assigned a reward of -1. This unified self-reward protocol ensures that performance differences among GradCuit, Self-Reflection, Self-Scored BoN, and LatentSeek do not arise from different reward mechanisms. The complete verifier prompt is:

```
You are a critical verifier for mathematical questions.
You will be given the original question and one final
answer.
Decide whether that answer is correct for the question.

QUESTION:
{question}

FINAL ANSWER:
{extracted_answer}

INSTRUCTIONS:
1. Verify only the final answer. Do not evaluate any
missing reasoning steps.
```

Table 5: Backbone-specific GradCuit configurations used in the main experiments. Layer indices identify the decoder-block input at which the inserted prefix states are optimized.

Backbone	Decoder Blocks	Optimized Layer	Maximum New Tokens
LLaMA-3.2-3B-Instruct	28	14	2,048
LLaMA-3.1-8B-Instruct	32	16	2,048
Qwen2.5-7B-Instruct	28	14	2,048
Qwen2.5-14B-Instruct	48	24	2,048
Qwen3-4B-Instruct-2507	36	18	4,096

```
2. Do not solve the problem independently from scratch
unless a tiny auxiliary calculation is strictly necessary
for verification.
3. Prefer reverse verification methods such as
substitution, plugging the answer back into the original
conditions, checking algebraic consistency, checking
boundary cases, or other direct validation targeted at the
proposed answer.
4. Accept mathematically equivalent forms when they
represent the same final answer.
5. If the final answer is correct, the verdict is True.
6. If the final answer is incorrect, the verdict is False.
```

Generation Prompts and Decoding. For the Boxed format, the system and user messages are:

```
{"role": "system", "content": "Please reason step by
step, and put your final answer within \boxed{}."}
{"role": "user", "content": q}
```

For the JSON format, the system and user messages are:

```
{"role": "system", "content": "Please reason step by
step, and put your final answer in a json dict with exactly
one key "answer", for example {"answer": "1.234"}."}
{"role": "user", "content": q}
```

GradCuit uses greedy decoding with a batch size of one. We set the maximum continuation length to 4,096 tokens for Qwen3-4B-Instruct-2507 because it typically produces longer reasoning trajectories, and to 2,048 tokens for all other backbones. To ensure a fair comparison, these backbone-specific maximum continuation lengths are kept identical across GradCuit and all baselines, while each method retains its required decoding strategy.

Random-Direction Variant. For the random-direction analysis, the reward-derived update is replaced at every step by an independently sampled Gaussian direction. Specifically, with $\boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, we apply

$$\Delta \boldsymbol{z}_{k+1}^{(l)} = \Delta \boldsymbol{z}_{k}^{(l)} + \eta \frac{\boldsymbol{\epsilon}_k}{\|\boldsymbol{\epsilon}_k\|_2}.$$

The sampled direction is globally L_2-normalized and then scaled by the learning rate. This variant does not use Adam or match the random-vector norm to the reward-derived gradient norm. The verifier is retained only for the common stopping rule. All remaining settings, including the inserted prefix, optimized layer, learning rate, generation budget, and decoding strategy, are identical to those of GradCuit.

Numerical and Randomness Settings. Backbone inference and latent states use bfloat16 precision. The main experiments use a fixed random seed of 42 and report one run for each backbone–benchmark–format configuration. For the layer-position analysis with LLaMA-3.2-3B-Instruct, the

0%, 25%, 50%, and 75% positions correspond to the word-embedding space and decoder-block input indices 7, 14, and 21, respectively.

A.2 Details of Other Baselines

Shared Protocol. All baselines use the native Hugging Face chat template of the corresponding backbone and the same Boxed and JSON system prompts used by GradCuit. Except for interactions required by each method, no additional instructions are added. The backbone-specific maximum generation lengths are also kept identical across methods: 4,096 tokens for Qwen3-4B-Instruct-2507 and 2,048 tokens for all other backbones. For methods requiring answer verification, the currently evaluated backbone itself serves as the self-verifier. The verifier uses the same prompt and binary reward rule described above, with greedy decoding and a maximum generation length of 8,192 tokens. All experiments use a fixed random seed of 42.

Chain-of-Thought. The Chain-of-Thought (CoT) baseline directly generates one complete response from the corresponding Boxed or JSON prompt using greedy decoding. It does not include the fixed prefix introduced by GradCuit and performs no subsequent verification, reflection, sampling, or test-time optimization. The single generated response is directly used as the final output.

One-Pass Self-Reflection. The initial response of Self-Reflection is generated using exactly the same procedure as CoT. The self-verifier then evaluates the extracted final answer. If the answer is accepted, the initial response is returned without further generation. Otherwise, we continue the existing multi-turn conversation by retaining the initial response as an assistant message and appending one user turn that asks the backbone to reconsider and revise its answer. The verifier supplies no textual critique or auxiliary reasoning to the backbone. The backbone generates one revised response using greedy decoding, which is returned as the final output. No additional reflection round is performed.

Self-Consistency. Self-Consistency generates five candidate responses for each problem using stochastic decoding with $\texttt{temperature} = 1.0$. We explicitly enable sampling and leave all other Hugging Face generation parameters at their default values. A final answer is extracted from each candidate according to the corresponding Boxed or JSON format. Candidates from which no valid answer can be extracted are discarded. The remaining extracted answers are compared using strict string matching, without invoking the verifier or performing additional mathematical-equivalence checking. The answer occurring most frequently is returned as the final prediction. If multiple answers receive the same highest number of votes, the one appearing first in generation order is selected.

Self-Scored Best-of-N. Self-Scored Best-of-N (BoN) generates 5 candidate responses using the same stochastic decoding configuration as Self-Consistency. All 5 candidates are generated and subsequently evaluated by the same self-verifier used for GradCuit. The verifier verdict for each candidate is mapped to the shared discrete reward, with accepted and rejected candidates receiving rewards of 0 and -1, respectively. The candidate with the highest reward is selected, and its complete generated response is returned as the final output. When multiple candidates receive the same highest reward, the final candidate in generation order is selected.

LatentSeek. We use the official implementation of LatentSeek (Li et al., 2025) and retain its recommended method-specific configuration. LatentSeek uses the same backbone, Boxed or JSON input prompt, answer parser, self-verifier, and backbone-specific maximum generation length as GradCuit. Its latent variables are optimized using Adam with a learning rate of 0.03 for at most ten optimization steps. All remaining method-specific implementation details follow the official LatentSeek implementation.

A.3 Hardware and Software.

All experiments were conducted on a server equipped with eight NVIDIA L40 GPUs. The software environment used Python 3.10, PyTorch 2.1.0, and CUDA 13.2.

Method	LLaMA3.2-3B		LLaMA3.1-8B		Qwen2.5-7B		Qwen2.5-14B		Qwen3-4B		Avg.	
Prompt Type	Boxed	JSON	Boxed	JSON	Boxed	JSON	Boxed	JSON	Boxed	JSON	Boxed	JSON
GPQA-Diamond												
GRADCUIT (Ours)	2.64	2.68	3.63	3.25	1.79	1.85	2.49	1.98	4.98	4.52	3.11	2.86
Self-Consistency	5	5	5	5	5	5	5	5	5	5	5	5
Self-Scored BoN	5	5	5	5	5	5	5	5	5	5	5	5
GSM8K												
GRADCUIT (Ours)	2.34	2.02	1.94	1.83	1.73	1.73	1.77	1.87	1.36	1.32	1.83	1.75
Self-Consistency	5	5	5	5	5	5	5	5	5	5	5	5
Self-Scored BoN	5	5	5	5	5	5	5	5	5	5	5	5
MATH-500												
GRADCUIT (Ours)	2.75	3.59	3.46	3.35	2.18	2.37	1.99	1.39	1.62	2.05	2.40	2.55
Self-Consistency	5	5	5	5	5	5	5	5	5	5	5	5
Self-Scored BoN	5	5	5	5	5	5	5	5	5	5	5	5

Table 6: Average optimization iterations and repeated-sampling budgets in the main experiments. For GRADCUIT, the values include the original unoptimized generation as the first round, followed by any subsequent optimized generations before termination. For Self-Consistency and Self-Scored BoN, the values report the fixed number of sampled responses.

B COMPUTE BUDGET AND EFFICIENCY

Self-Consistency and Self-Scored BoN use a fixed sampling budget of five responses and generate all five candidates for every problem. In contrast, GRADCUIT first generates an unoptimized response and then iteratively updates the latent states until the self-verifier accepts the answer.

Table 6 counts the initial response as the first round and includes all subsequent optimized generations. Across the 30 backbone–benchmark–format settings, GRADCUIT averages 2.42 rounds, ranging from 1.32 to 4.98 and remaining below five in every setting. These results show that its improvements do not rely on more reasoning rounds.

C ADDITIONAL CASES

The main text presents a representative case from MATH-500; here, we provide additional cases from the other two benchmarks, GPQA-Diamond and GSM8K. In the GPQA-Diamond case shown in Table 7, the original output selects option D despite identifying an inconsistency in its reaction sequence, whereas the optimized output correctly reasons about the order in which the substituents should be introduced and selects option C. In the GSM8K case shown in Table 8, the original output incorrectly compares the final values of the two investments and selects the electronic gadgets, while the optimized output instead compares their respective profits of $\$125$ and $\$96$, yielding the correct maximum profit of $\$125$. These cases further illustrate that hidden-state optimization of GRADCUIT can reorganize the overall reasoning structure and strategy, whereas LatentSeek modifies the early decoded token sequence and may consequently introduce malformed or duplicated text.

Question. Which sequence of reactions would lead to the high-yield synthesis of 1-(3-bromo-5-nitrophenyl)ethan-1-one from benzene? **Choices:** (A) i) HNO_3/H_2SO_4; ii) Fe/HCl; iii) $NaNO_2$/HCl; iv) H_3PO_2; v) $Br_2/FeBr_3$; vi) $CH_3COCl/AlCl_3$; vii) HNO_3/H_2SO_4 (B) i) $CH_3COCl/AlCl_3$; ii) $Br_2/FeBr_3$; iii) HNO_3/H_2SO_4; iv) Fe/HCl; v) HNO_3/H_2SO_4; vi) $NaNO_2$/HCl; vii) H_3PO_2 (C) i) HNO_3/H_2SO_4; ii) Fe/HCl; iii) $CH_3COCl/AlCl_3$; iv) $Br_2/FeBr_3$; v) HNO_3/H_2SO_4; vi) $NaNO_2$/HCl; vii) H_3PO_2 (D) i) $Br_2/FeBr_3$; ii) HNO_3/H_2SO_4; iii) $CH_3COCl/AlCl_3$; iv) HNO_3/H_2SO_4; v) Fe/HCl; vi) $NaNO_2$/HCl; vii) H_3PO_2	
Original	*1. The target compound is 1-(3-bromo-5-nitrophenyl)ethan-1-one, which is an aromatic ketone with a bromo and a nitro group on the benzene ring. [...] The closest sequence that matches the above steps is option (D), but it includes an extra step (v) HNO3/H2SO4, which is not necessary. Therefore, the correct answer is $\boxed{D}$.*
Optimized	*The target compound is 1-(3-bromo-5-nitrophenyl)ethan-1-one. To synthesize this compound, we need to introduce a nitro group (-NO2) [...] A, B and D options do not include the correct sequence of reactions. Therefore, the correct answer is: $\boxed{C}$.*

Table 7: A case from GPQA-Diamond. The backbone is Llama3.1-8B-Instruct and the ground-truth answer is C.

Question. A merchant must choose between purchasing jewelry worth $\$5{,}000$ and electronic gadgets worth $\$8{,}000$. The jewelry market is expected to rise by 2.5%, while the electronics market is expected to rise by 1.2% within the month. What is the maximum profit the merchant can obtain?	
Original	*Step 1: Calculate the future value of the jewelry after a 2.5% increase. [...] Profit = $8,096 - $8,000 Profit = $96 Therefore, the merchant will make a profit of $96 by choosing the electronic gadgets. $\boxed{\$96}$*
Optimized	*1. The merchant has two options: - Option 1: Purchase jewelry worth $5,000. [...] 9. Since $125 is greater than $96, the merchant should choose Option 1 (jewelry) to maximize profit. 10. Therefore, the maximum profit the merchant can make is $125. $\boxed{\$125}$*

Table 8: A case from GSM8K. The backbone is Llama3.1-8B-Instruct and the ground-truth answer is $\$125$.