

TRLM: Shared-Weight Recursive Reasoning for Non-Autoregressive Language Generation

Doğa Oytaç

İzmir Katip Çelebi University

dogaoytac@gmail.com

Abstract

Large language models achieve strong reasoning but require substantial parameters and memory, limiting deployment in resource-constrained settings. We introduce the Tiny Recursive Language Model (TRLM), a small masked non-autoregressive model that increases effective computation depth through shared-weight recursion. TRLM represents a masked sequence with input, answer, and latent reasoning states. A shared Transformer repeatedly refines the latent reasoning state, conditioning each update on trajectory time and step size via adaptive layer normalization. Training combines long- and short-trajectory answer-prediction losses with representation-level consistency. At inference, TRLM recursively refines latent states and fills all masked answer tokens in a single parallel decoding pass. We pretrain TRLM on approximately one billion OpenWebText and FineWeb-Edu tokens, then fine-tune and evaluate it on GSM8K and SVAMP against scratch-pretrained GPT-2-style, Gemma-4-Text-style, Diffusion-Gemma-style, and Mamba-2-style baselines. TRLM matches the best observed GSM8K result with 2.88% exact-match accuracy and achieves the highest SVAMP point estimate in our controlled comparison at 9.00%. Performance is non-monotonic with recursion depth: the highest observed GSM8K accuracy occurs at eight steps, whereas SVAMP peaks at four, suggesting that the preferred recursion depth may depend on the task in this setting.

1 Introduction

Scaling dominates the current paradigm in Natural Language Processing. State-of-the-art Large Language Models (LLMs), such as DeepSeek R1 (Guo et al., 2025) and Gemini 3 (Team et al., 2025), achieve their capabilities primarily through massive parameter counts and computationally expensive inference strategies like Test-Time Compute

(TTC) (Snell et al., 2024). To solve complex reasoning tasks, these models rely heavily on Chain-of-Thought (CoT) prompting (Wei et al., 2023), attempting to mimic human deduction by generating intermediate steps. However, this approach faces a fundamental bottleneck. Auto-regressive generation is inherently brittle; a single incorrect token early in the sequence can cascade, rendering the entire reasoning chain and the final answer invalid (You et al., 2025). Furthermore, the efficacy of CoT is contingent upon high-quality reasoning data, which is often scarce or unavailable, making the process both data-hungry and computationally expensive (Shum et al., 2024).

These limitations are particularly relevant in small-scale language modeling, where increasing model size is not always a desirable or practical route to improved reasoning. This motivates architectures that can make more effective use of a limited parameter budget. In this work, we study a complementary form of efficiency: rather than increasing the number of learned parameters, we ask whether a compact model can increase its effective reasoning depth by reusing the same computation module recursively. This makes it possible to trade additional inference-time computation for reasoning depth while keeping the parameter count fixed.

A promising alternative to sheer scale is *recursive reasoning*, where a smaller model iteratively refines its internal state to solve a problem. This direction was recently pioneered by the Hierarchical Reasoning Model (HRM) (Wang et al., 2025), demonstrating that a relatively small, 27-million-parameter model could outperform substantially larger language models on several puzzle-based reasoning benchmarks. HRM consists of two coupled recurrent modules operating at different update rates: a *low-level* module (f_L), which updates frequently to perform rapid, detailed computations, and a *high-level* module (f_H), which updates less frequently to support slower, more abstract reason-

ing. Its training method is theoretically motivated by the Implicit Function Theorem (IFT) (Krantz and Parks, 2002) and deep equilibrium models (Bai et al., 2019). HRM uses a one-step gradient approximation that backpropagates only through the most recent update of each module while treating earlier recurrent states as constants.

The Tiny Recursive Model (TRM) (Jolicoeur-Martineau, 2025) is a streamlined architecture built on the philosophy that “Less is More.” Unlike its predecessor, TRM uses a single ultra-compact network consisting of only two layers. Instead of complex hierarchies, it employs a straightforward recursive process in which the model repeatedly updates a latent reasoning trace (z) and a predicted answer (y). TRM uses Deep Supervision, in which prediction losses are applied repeatedly across recursive updates rather than only to the final output. With $N_{\text{sup}} = 16$ supervision steps, this allows the small architecture to emulate the effective computational depth of a much deeper model.

More recently, Tiny Autoregressive Recursive Models (Raubu et al., 2026) adapted TRM-style refinement to causal next-token prediction under controlled, compute-matched settings. Although simpler two-level refinement baselines performed competitively, the full autoregressive TRM architecture did not provide reliable additional gains. In contrast, our work investigates recursive refinement in a non-autoregressive masked-prediction setting, where a latent reasoning state is refined before answer tokens are predicted in parallel.

Instead of learned halting, TRLM uses explicitly specified, trajectory-conditioned recursion schedules.

Our objective is to adapt this architecture, originally validated on puzzles such as ARC-AGI (Chollet, 2019), to generative, text-based reasoning tasks. We discard the fixed-point assumptions and biological constraints of earlier formulations and adopt a simpler interpretation: y represents the answer state used for prediction, while z serves as a persistent latent reasoning state. Deliberately keeping the model small substantially reduces its parameter count and per-recursion computational cost. Rather than providing constant memory efficiency, the architecture offers a flexible compute–memory trade-off: resource usage increases with the number of recursive steps, but trajectory conditioning enables the model to achieve performance close to that of much deeper recursive trajectories even when only a small number of steps is used. Thus,

TRLM increases computational depth through recursion rather than additional parameterized layers. This makes it well suited to small-model settings because the parameter budget remains fixed while inference-time computation scales with recursive steps. Our main contributions are summarized as follows:

- **Adaptation of recursive architectures for non-autoregressive language modeling:** We introduce TRLM, which adapts the Tiny Recursive Model idea from static puzzle-solving to text reasoning.
- **Trajectory-conditioned recursive refinement for masked language modeling:** Building on trajectory-conditioned looped computation (Jeddi et al., 2026), we adapt normalized trajectory-time and step-size conditioning to a masked non-autoregressive architecture with separate answer and latent reasoning states. These signals are injected into the shared Transformer blocks through adaptive normalization, allowing the same reasoning module to behave differently across refinement stages.
- **Shared-weight computation for parameter-efficient reasoning:** TRLM increases effective computation depth by repeatedly applying the same Transformer reasoning module rather than adding independently parameterized layers. This allows a small model to trade additional inference-time computation for effective depth without increasing its learned parameter count.
- **Controlled downstream architecture comparison:** We fine-tune TRLM on GSM8K and SVAMP and compare it with scratch-pretrained GPT-2-style (Radford et al., 2019), Gemma-4-Text-style (Team et al., 2026), Diffusion-Gemma-style (Google DeepMind, 2026), and Mamba-2-style (Dao and Gu, 2024) baselines trained with the same tokenizer, data mixture, token budget, and evaluated with the same numeric exact-match metric.
- **Analysis of inference-time recursion depth:** We evaluate multiple recursion depths under a fixed single-pass parallel decoding procedure and observe non-monotonic accuracy across depths, with different depths attaining

the highest point estimates on GSM8K and SVAMP.

2 Related Work

2.1 Parameter Sharing and Recurrent Depth

A central line of work related to TRLM obtains effective depth by repeatedly applying a shared computation module rather than stacking independently parameterized layers. The Universal Transformer recurrently applies one transition function across depth, refining token states over multiple steps and thereby decoupling computation depth from parameter count (Dehghani et al., 2019). This connects layer sharing to recurrent computation and backpropagation through time (Werbos, 1974; Lillicrap and Santoro, 2019), while inheriting the optimization difficulties of long recurrent trajectories (Pascanu et al., 2013). Adaptive halting allows a model to learn when to stop iterative computation, while conditional computation adjusts the amount or path of computation based on the input. Both approaches therefore treat depth as an input- or position-dependent inference-time resource. However, discrete or stochastic routing can complicate training (Graves, 2017; Dehghani et al., 2019; Bengio et al., 2013).

Recent recursive and looped language models extend this idea to test-time reasoning. Relaxed Recursive Transformers combine shared-depth computation with layer-specific low-rank adaptations, balancing exact weight tying with per-layer specialization (Bae et al., 2025). LoopFormer introduces elastic-depth looped Transformers trained over variable-length trajectories, conditioning each loop on normalized trajectory time and step size and using shortcut-consistency training to align representations produced by trajectories of different lengths (Jeddi et al., 2026). Other recurrent-depth approaches repeatedly apply shared Transformer modules to increase internal computation without enlarging the base model (Geiping et al., 2025; Saunshi et al., 2025). In contrast to autoregressive test-time scaling, which typically lengthens visible reasoning traces, recurrent depth increases the transformations applied before prediction (Snell et al., 2024). TRLM follows this shared-weight tradition but adapts trajectory-conditioned recursive computation to masked language modeling through separate answer and latent reasoning states.

2.2 Latent Reasoning

Latent reasoning methods move part of reasoning from explicit text tokens into continuous hidden representations. This addresses a limitation of chain-of-thought generation: intermediate reasoning must be serialized into natural language and is exposed to autoregressive error accumulation (Wei et al., 2023; You et al., 2025). Coconut reuses a final hidden state as a continuous thought representation, feeding it back into the model without first decoding a discrete token (Hao et al., 2025). Related work treats reasoning as iterative computation over hidden states, either by increasing recurrent depth at test time or by looping Transformer computation in latent space (Geiping et al., 2025; Saunshi et al., 2025; Zhu et al., 2025; Tan et al., 2026). TRLM shares this shift from visible rationales to internal refinement, but organizes the latent process differently: instead of inserting continuous thoughts into an autoregressive chain, it maintains separate answer and reasoning states, denoted by y and z , and predicts masked answer tokens in parallel after recursive refinement.

2.3 Masked and Non-Autoregressive Language Generation

Autoregressive language models factorize generation from left to right, so each new token becomes conditioning context for later decisions (Radford et al., 2019). This is natural for open-ended continuation, but it makes decoding sequential and exposes later predictions to prefix-level errors. Masked and non-autoregressive generation instead treats the target span as an unknown canvas: observed context remains fixed, and hidden positions are predicted from a bidirectional representation of the available sequence, allowing multiple target tokens to be scored in parallel.

Masked and diffusion-based language models show that text generation need not follow strict left-to-right autoregression. Discrete diffusion language models, such as DiffusionGemma, generate text through a multi-step denoising process over discrete token representations (Google DeepMind, 2026). TRLM, however, is not a diffusion language model: it does not define a diffusion/noising process or a multi-step denoising trajectory over the output tokens. Instead, TRLM uses a masked non-autoregressive interface while performing recursive computation in a continuous latent reasoning state. After the latent state has been refined for a fixed

number of recursive steps, all masked answer positions are predicted once in parallel.

3 Method

3.1 Tiny Recursive Language Model

Figure 1 provides an overview of the TRLM architecture and its recursive latent-refinement procedure. The Tiny Recursive Language Model (TRLM) is a masked non-autoregressive model that increases effective depth by recursively reusing a shared Transformer reasoning module. Increasing recursion depth adds computation but does not add parameters. For a masked input sequence

$$\mathbf{s} = (s_1, \dots, s_L),$$

the token embedding layer produces a fixed input representation

$$\mathbf{x} = E(\mathbf{s}) \in \mathbb{R}^{B \times L \times d}.$$

TRLM also maintains an answer state $\mathbf{y}^{(0)}$ and a persistent latent reasoning state $\mathbf{z}^{(k)}$, initialized as

$$\mathbf{y}^{(0)} = \mathbf{0}, \quad \mathbf{z}^{(0)} = \mathbf{0}.$$

During recursive reasoning, $\mathbf{x}$ remains fixed while $\mathbf{z}$ is refined; the answer state is updated only after the final reasoning state has been obtained.

At recursive step k , the model concatenates the input, answer, and reasoning states and projects them back to the model dimension:

$$\mathbf{h}_0^{(k)} = W_c [\mathbf{x}; \mathbf{y}^{(0)}; \mathbf{z}^{(k)}] + b_c.$$

This hidden sequence is processed by the shared Transformer reasoning module,

$$\mathbf{h}_{\ell}^{(k)} = \mathcal{T}_{\theta, \ell} (\mathbf{h}_{\ell-1}^{(k)}; \mathbf{c}^{(k)}), \quad \ell = 1, \dots, N.$$

The same layer stack $\{\mathcal{T}_{\theta, \ell}\}_{\ell=1}^N$ is reused for all recursion steps, and the reasoning state is updated residually:

$$\mathbf{z}^{(k+1)} = \mathbf{z}^{(k)} + \text{Dropout} (\mathbf{h}_N^{(k)}).$$

We use externally specified recursion trajectories rather than a learned halting variable. Consequently, the number and sizes of recursive updates are known during each forward pass. This avoids the implementation overhead of a separate q_{halting} prediction head and the additional optimization burden of learning when to stop recurrent computation.

The compute budget is instead controlled directly by the chosen trajectory: the number of step sizes determines recursion depth, and their values determine how the normalized refinement interval is traversed. Trajectory time t_k and step size Δt_k also tell the shared module which refinement stage it is executing, allowing the same weights to specialize their behavior across early and late updates.

Following the trajectory-conditioning formulation used in LoopFormer (Jeddi et al., 2026), each recursive step is conditioned on its normalized trajectory time t_k and step size Δt_k :

$$\mathbf{e}^{(k)} = \text{MLP}_t(t_k) + \text{MLP}_{\Delta t}(\Delta t_k).$$

A conditioning projection maps $\mathbf{e}^{(k)}$ to adaptive normalization parameters for the attention and feed-forward branches of the shared Transformer. In this way, the same pre-normalized Transformer stack, implemented with rotary positional embeddings, grouped-query attention, and SwiGLU feed-forward layers, can behave differently at early and late refinement stages without introducing step-specific weights. Unlike LoopFormer, which applies trajectory conditioning to the hidden states of a decoder-only autoregressive looped language model, TRLM applies it to recursive latent reasoning before a single parallel masked-token prediction step.

After R recursive reasoning steps, the final reasoning state $\mathbf{z}^{(R)}$ is used to update the answer state:

$$\mathbf{a} = W_a [\mathbf{y}^{(0)}; \mathbf{z}^{(R)}] + b_a.$$

This representation is passed once more through the shared Transformer layer stack and added residually to the initial answer state:

$$\mathbf{y}^* = \mathbf{y}^{(0)} + \mathcal{T}_{\theta} (\mathbf{a}).$$

The model then applies final normalization and a vocabulary projection,

$$\mathbf{o} = \text{LayerNorm} (\mathbf{y}^*), \quad \mathbf{l} = \mathbf{o} E^{\top}.$$

The input embedding matrix E is tied to the language-model output projection. Although logits are produced for all sequence positions, supervision is applied only to the masked target answer positions:

$$\mathcal{L}_{\text{LM}} = - \sum_{i \in \mathcal{M}} \log p_{\theta} (s_i^* | \mathbf{s}_{\setminus \mathcal{M}}),$$

where $\mathcal{M}$ denotes the masked and supervised answer positions.

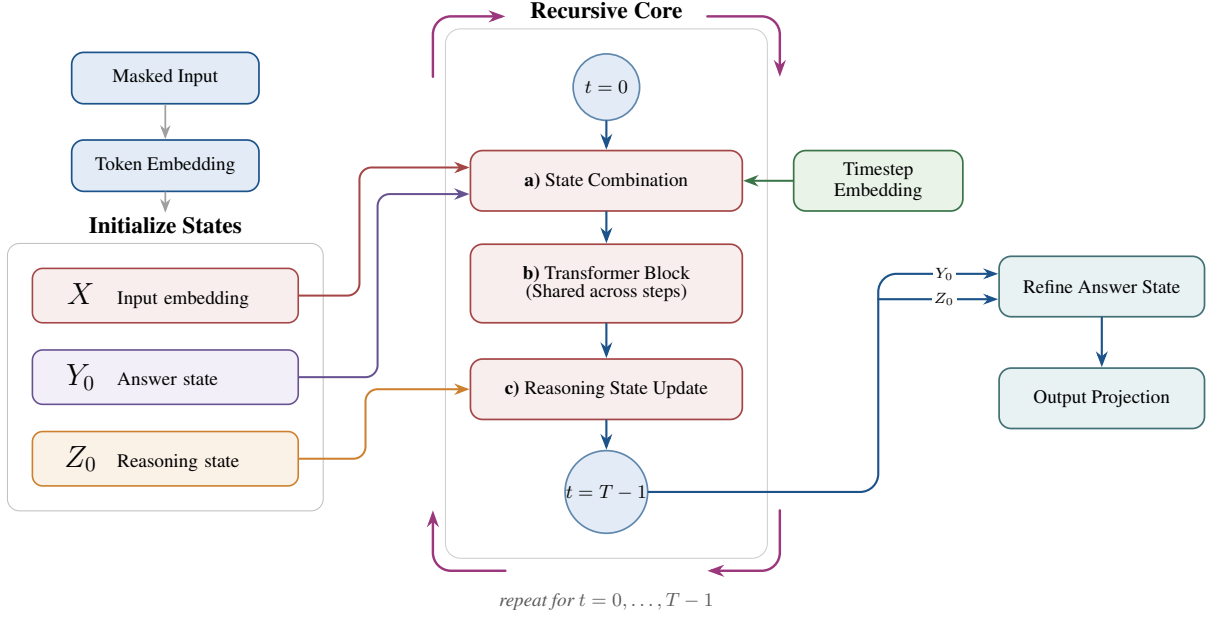


Figure 1: Overview of TRLM. The model receives a masked prompt–answer sequence, recursively refines a latent reasoning state with a shared Transformer module, and predicts the masked answer tokens in parallel.

3.2 Long–Short Trajectory Training

Our long–short trajectory training is based on the shortcut-consistency principle introduced in LoopFormer (Jeddi et al., 2026), which jointly trains long and shorter computation trajectories and encourages the shorter route to approximate a stop-gradient target produced by the longer route. We adapt this idea to masked non-autoregressive reasoning: rather than operating directly on a decoder-only language-model hidden state, TRLM applies the objective to its answer representations produced after recursive latent refinement.

Because TRLM does not learn a halting policy, it is trained with explicitly sampled long and short recursion schedules for the same masked input. A trajectory is a sequence of positive step sizes

$$\tau = (\Delta t_1, \dots, \Delta t_K), \quad \sum_{k=1}^K \Delta t_k = 1,$$

where t_k and Δt_k condition the recursive update. Applying TRLM with τ produces answer representation $\mathbf{r}_\tau$ and logits $\mathbf{l}_\tau$; the length of τ sets recursion depth while parameters remain shared.

During training, the long trajectory samples a teacher recursion length

$$K \sim \text{Uniform} \{K'_{\min}, \dots, K_{\max}\},$$

$$K'_{\min} = \max(2, K_{\min}).$$

and uses uniform increments

$$\tau_{\text{long}} = \left(\frac{1}{K}, \dots, \frac{1}{K} \right).$$

The short trajectory is a coarser path over the same interval: J is sampled uniformly from $\{1, \dots, K-1\}$, and step sizes are drawn from $\{m/K\}_{m=1}^K$ subject to summing to one.

For trajectory τ , the masked language-modeling objective over supervised answer positions $\mathcal{M}$ is

$$\mathcal{L}_{\text{LM}}(\tau) = - \sum_{i \in \mathcal{M}} \log p_\theta(s_i^* | \mathbf{s}_{\setminus \mathcal{M}}, \tau).$$

The short-path language-modeling loss is

$$\mathcal{L}_{\text{short}} = \mathcal{L}_{\text{LM}}(\tau_{\text{short}}).$$

Let $\mathbf{r}_{\text{long}}$ and $\mathbf{r}_{\text{short}}$ be the final normalized answer representations produced by τ_{long} and τ_{short} . The representation-consistency term is

$$\mathcal{L}_{\text{cons}} = \|\mathbf{r}_{\text{short}} - \text{stopgrad}(\mathbf{r}_{\text{long}})\|_2^2.$$

The resulting training objective is

$$\mathcal{L} = \mathcal{L}_{\text{LM}}(\tau_{\text{long}}) + \lambda_{\text{short}} \mathcal{L}_{\text{short}} + \lambda_{\text{cons}} \mathcal{L}_{\text{cons}},$$

where λ_{short} and λ_{cons} weight the short-path language-modeling and representation-consistency terms. The long and short trajectories receive the same masked-token supervision. The consistency term aligns the short-path representation with a stop-gradient copy of the long-path representation, encouraging lower-compute trajectories to approximate the higher-compute solution.

3.3 Recursive Masked Generation

The prompt remains visible and the answer span is replaced by mask tokens. Let $\mathcal{M}$ be the masked answer positions. For recursion depth R , TRLM uses the uniform trajectory

$$\tau_R = \left(\frac{1}{R}, \dots, \frac{1}{R} \right),$$

which spans the normalized interval $[0, 1]$. The model performs one forward pass with this trajectory and produces logits for all positions:

$$\mathbf{l} = f_{\theta}(\mathbf{s}, \tau_R).$$

Decoding is performed independently at the masked positions after invalid ids are filtered from the candidate set. The mask token is always forbidden, and other special tokens can also be suppressed when invalid. Each masked position receives the maximum-probability valid token:

$$\hat{s}_i = \arg \max_{v \in \mathcal{V}_{\text{valid}}} \text{softmax}(\mathbf{l}_i)_v, \quad i \in \mathcal{M}.$$

Unmasked positions are copied from the input:

$$\hat{s}_i = s_i, \quad i \notin \mathcal{M}.$$

Generation therefore consists of R latent refinement updates followed by one parallel token-selection step; R changes inference-time computation without changing parameter count or output length.

4 Experiments

4.1 Tasks and Evaluation Protocol

We evaluate all models on two mathematical reasoning benchmarks, GSM8K (Cobbe et al., 2021) and SVAMP (Patel et al., 2021). GSM8K consists of grade-school mathematical word problems that typically require multi-step arithmetic reasoning, while SVAMP evaluates arithmetic word-problem reasoning under controlled variations designed to reduce reliance on shallow linguistic or structural cues. We use evaluation sets containing 1,319 GSM8K examples and 300 SVAMP examples. Representative examples from both evaluation suites are provided in Appendix A.

Each example is formatted as a natural-language prompt followed by an answer target ending with a numeric final answer. For TRLM, prompt tokens remain visible while the answer span is replaced by mask tokens and predicted non-autoregressively.

We evaluate recursion depths $r \in \{1, 2, 4, 8, 16\}$ using the uniform trajectories defined in Section 3.3. The GPT-2-style, Gemma-4-Text-style, and Mamba-2-style baselines use greedy autoregressive decoding without sampling, whereas the Diffusion-Gemma-style baseline uses its native iterative denoising procedure with a fixed evaluation seed. The same prompt formatting, numeric answer extraction, and numeric exact-match metric are used across all models. All experiments and latency measurements were conducted on a 95.6 GB G4 GPU environment.

4.2 Compared Models

We compare TRLM against four scratch-pretrained baselines representing autoregressive Transformer, Gemma-style attention, recurrent state-space, and diffusion-style generation paradigms. These are compact architecture-family baselines rather than official released checkpoints. Appendix B summarizes the controlled configurations; TRLM differs by reusing a shared Transformer reasoning module recursively and predicting masked answer tokens in parallel.

4.3 Main Results

Table 1 summarizes the fixed-length generation results on GSM8K and SVAMP. Among the non-TRLM baselines, the Gemma-4-Text-style model achieves the strongest GSM8K accuracy, reaching 2.88%, while the GPT-2-style model obtains the highest baseline result on SVAMP with 7.33%. The Diffusion-Gemma-style and Mamba-2-style baselines perform lower in this small scratch-pretrained setting, despite representing alternative non-autoregressive denoising and recurrent state-space generation paradigms.

TRLM is competitive with the strongest evaluated baseline across the two benchmarks. On GSM8K, TRLM with $r = 8$ reaches 2.88%, matching the highest baseline point estimate obtained by the Gemma-4-Text-style model, while $r = 16$ remains close at 2.81%. On SVAMP, all evaluated TRLM depths yield higher point estimates than the strongest non-TRLM baseline; the highest observed accuracy is 9.00% at $r = 4$, compared with 7.33% for the GPT-2-style baseline.

These differences should be interpreted cautiously given the relatively small number of correct predictions, particularly on SVAMP. Nevertheless, the results show that recursive masked generation remains competitive with similarly sized scratch-

Model	Params	GSM8K Acc.	SVAMP Acc.	Avg. Gen. Time
Diffusion-Gemma-style (scratch)	74.3M	0.99% (13/1319)	3.00% (9/300)	728.31 ms
Gemma-4-Text-style (scratch)	77.6M	2.88% (38/1319)	6.33% (19/300)	53.47 ms
GPT-2-style (scratch)	73.5M	2.58% (34/1319)	7.33% (22/300)	21.15 ms
Mamba-2-style (scratch)	76.0M	1.29% (17/1319)	1.33% (4/300)	75.19 ms
TRLM $r = 1$	74.5M	2.73% (36/1319)	8.33% (25/300)	11.06 ms
TRLM $r = 2$	74.5M	2.65% (35/1319)	8.67% (26/300)	16.25 ms
TRLM $r = 4$	74.5M	2.65% (35/1319)	9.00% (27/300)	25.86 ms
TRLM $r = 8$	74.5M	2.88% (38/1319)	8.00% (24/300)	46.21 ms
TRLM $r = 16$	74.5M	2.81% (37/1319)	8.67% (26/300)	85.72 ms

Table 1: Main generation results on GSM8K and SVAMP. Accuracy is numeric exact match. Average generation time is the per-example mean computed over both test sets from total generation time. Best accuracy values and the lowest average generation time are shown in bold.

pretrained architectures while providing a direct inference-time control over additional latent computation.

Accuracy varies non-monotonically with recursion depth in our experiments. The highest GSM8K point estimate occurs at $r = 8$, whereas the highest SVAMP point estimate occurs at $r = 4$. Thus, within the evaluated range, increasing recursion depth does not consistently increase accuracy. In contrast, latency increases monotonically with depth, making r a direct test-time compute control.

4.4 Analysis of Trajectory Conditioning

Table 2 compares TRLM with and without trajectory-time and step-size conditioning across the evaluated recursion depths. We remove trajectory-time and step-size conditioning while keeping the remaining architecture and evaluation protocol fixed. The conditioned variant yields higher accuracy point estimates at four of the five evaluated depths on GSM8K and matches the unconditioned model at $r = 4$. On SVAMP, conditioning yields higher point estimates at four depths, while the unconditioned model is higher at $r = 8$. These differences are generally small and should therefore be interpreted as descriptive rather than as evidence of statistically significant improvements.

Conditioning consistently increases latency because of the additional conditioning projections and adaptive-normalization computations. Overall, the ablation suggests that trajectory conditioning can be beneficial across several recursion depths, although the magnitude and consistency of the accuracy effect require further evaluation.

5 Discussion

The results suggest that the main benefit of TRLM is not a monotonic gain from increasing recursion depth, but the ability to expose a controllable compute–depth trade-off while keeping the learned parameter count fixed. On GSM8K, TRLM matches the highest baseline point estimate at $r = 8$, whereas on SVAMP all evaluated recursion depths yield higher point estimates than the strongest non-TRLM baseline. Notably, $r = 1$ already reaches 8.33% accuracy on SVAMP, compared with 7.33% for the strongest non-TRLM baseline. This indicates that the observed SVAMP advantage cannot be attributed solely to increasing recursive depth; characteristics of the shared-weight masked architecture and its training objective may also contribute. Because the present evaluation covers only two reasoning benchmarks, this interpretation should remain tentative.

The relationship between recursion depth and accuracy is clearly non-monotonic in our experiments. GSM8K reaches its highest point estimate at $r = 8$, while SVAMP peaks at $r = 4$, and deeper trajectories do not consistently improve either benchmark. One possible explanation is that additional recursive updates can eventually become redundant or perturb useful intermediate representations rather than further refining them. Another possibility is that different problems require different amounts of latent computation. The current experiments do not isolate these mechanisms, but the observed pattern suggests that a fixed recursion depth is unlikely to be optimal for every input.

The trajectory-conditioning ablation provides additional evidence that the behavior of the shared module depends on its position in the refinement

r	With trajectory conditioning			Without trajectory conditioning		
	GSM8K Acc.	SVAMP Acc.	Avg. Time	GSM8K Acc.	SVAMP Acc.	Avg. Time
1	2.73% (36/1319)	8.33% (25/300)	11.06 ms	2.20% (29/1319)	8.00% (24/300)	9.74 ms
2	2.65% (35/1319)	8.67% (26/300)	16.25 ms	2.58% (34/1319)	8.00% (24/300)	14.13 ms
4	2.65% (35/1319)	9.00% (27/300)	25.86 ms	2.65% (35/1319)	8.00% (24/300)	22.06 ms
8	2.88% (38/1319)	8.00% (24/300)	46.21 ms	2.35% (31/1319)	8.33% (25/300)	38.66 ms
16	2.81% (37/1319)	8.67% (26/300)	85.72 ms	2.50% (33/1319)	8.00% (24/300)	71.38 ms

Table 2: Trajectory-conditioning ablation for TRLM. The conditioned results are the TRLM rows from Table 1. The unconditioned model is trained and fine-tuned without trajectory-time and step-size conditioning, while keeping the same 74.5M parameter scale and evaluation protocol.

trajectory. Conditioning yields higher accuracy point estimates at four of five evaluated depths on both benchmarks, although the differences are generally small and not uniform. This suggests that providing trajectory time and step size can help a shared module distinguish early from late refinement stages, while also introducing a modest computational overhead.

Shared-weight recursion keeps the number of learned parameters fixed as recursion depth increases, but additional computation is not free. In our measurements, latency increases from 11.06 ms at $r = 1$ to 85.72 ms at $r = 16$. It therefore remains unclear whether the same accuracy–compute trade-off will persist at larger model and data scales. The non-monotonic depth results also motivate adaptive computation as a natural extension of TRLM. Rather than selecting R externally, future variants could use a learned halting mechanism or a convergence criterion based on changes in the latent reasoning state or output distribution, allowing easier examples to terminate earlier while allocating additional computation to harder ones.

6 Conclusion

We introduced TRLM, a masked non-autoregressive language model that applies a shared Transformer reasoning module recursively before predicting answer tokens in parallel. This design separates parameter count from effective computation depth: changing the recursion depth changes inference-time compute, but not the number of learned parameters.

Across GSM8K and SVAMP, TRLM is competitive with similarly sized scratch-pretrained autoregressive, state-space, and diffusion-style baselines. It matches the highest GSM8K point estimate and obtains the highest SVAMP point estimate in our comparison, while the fastest TRLM setting also

gives the lowest observed average generation latency.

Accuracy varies non-monotonically with recursion depth: the highest observed GSM8K accuracy occurs at $r = 8$, whereas SVAMP reaches its highest point estimate at $r = 4$. These results suggest that additional recursive computation is not uniformly beneficial and motivate future work on adaptive recursion policies, answer-canvas selection, and larger-scale evaluation.

Limitations

This study was conducted under limited computational resources, which constrained both the scale of the evaluated models and the extent of their training. To keep pretraining, fine-tuning, and repeated evaluation computationally feasible, all models were maintained at a relatively small parameter scale and were trained with a limited token budget. Consequently, the reported results may not reflect the behavior of TRLM under substantially larger model sizes, longer training schedules, or more diverse pretraining corpora. It remains unclear whether the observed advantages of recursive parameter sharing will persist, diminish, or become stronger as model and data scale increase.

The evaluation is also restricted to GSM8K and SVAMP. These datasets were selected because they are widely used and well-established mathematical reasoning benchmarks, allowing TRLM to be compared with common baseline architectures under a consistent and reproducible evaluation protocol. However, both benchmarks consist primarily of relatively short mathematical word problems and therefore represent only a limited subset of language reasoning. The results should not be interpreted as evidence of general superiority across broader capabilities such as commonsense reasoning, long-context understanding, knowledge-

intensive question answering, or open-ended text generation. Future work should therefore evaluate larger models with more extensive training and include a broader range of benchmarks.

TRLM uses externally selected recursion depths rather than learning an input-dependent halting policy. Although this simplifies training and provides predictable compute, it may allocate too few or too many recursive updates to individual examples. A direct comparison with learned q_{halting} -based adaptive computation remains future work.

Acknowledgments

This work received support from the Secretariat of Defence Industries of the Republic of Türkiye (SSB) through the EVREN Platform.

References

- Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. 2025. [Relaxed recursive transformers: Effective parameter sharing with layer-wise lora](#). *Preprint*, arXiv:2410.20672.
- Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2019. [Deep equilibrium models](#). *Preprint*, arXiv:1909.01377.
- Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. [Estimating or propagating gradients through stochastic neurons for conditional computation](#). *Preprint*, arXiv:1308.3432.
- François Chollet. 2019. [On the measure of intelligence](#). *Preprint*, arXiv:1911.01547.
- 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. 2021. [Training verifiers to solve math word problems](#). *Preprint*, arXiv:2110.14168.
- Tri Dao and Albert Gu. 2024. [Transformers are ssms: Generalized models and efficient algorithms through structured state space duality](#). *Preprint*, arXiv:2405.21060.
- Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2019. [Universal transformers](#). *Preprint*, arXiv:1807.03819.
- Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. 2025. [Scaling up test-time compute with latent reasoning: A recurrent depth approach](#). *Preprint*, arXiv:2502.05171.
- Google DeepMind. 2026. [DiffusionGemma: Open-weights discrete diffusion mixture-of-experts language model](#). <https://huggingface.co/google/diffusionGemma-26B-A4B-it>. Accessed: 2026-07-17.
- Alex Graves. 2017. [Adaptive computation time for recurrent neural networks](#). *Preprint*, arXiv:1603.08983.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. [Deepseek-r1 incentivizes reasoning in llms through reinforcement learning](#). *Nature*, 645(8081):633–638.
- Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2025. [Training large language models to reason in a continuous latent space](#). *Preprint*, arXiv:2412.06769.
- Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. 2026. [Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation](#). *Preprint*, arXiv:2602.11451.
- Alexia Jolicoeur-Martineau. 2025. [Less is more: Recursive reasoning with tiny networks](#). *Preprint*, arXiv:2510.04871.
- Steven George Krantz and Harold R Parks. 2002. *The implicit function theorem: history, theory, and applications*, volume 202. Springer.
- Timothy P Lillicrap and Adam Santoro. 2019. Back-propagation through time and the brain. *Current opinion in neurobiology*, 55:82–89.
- Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. [On the difficulty of training recurrent neural networks](#). *Preprint*, arXiv:1211.5063.
- Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. [Are nlp models really able to solve simple math word problems?](#) *Preprint*, arXiv:2103.07191.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.
- Paulius Rauba, Claudio Fanconi, and Mihaela van der Schaar. 2026. [Tiny autoregressive recursive models](#). *Preprint*, arXiv:2603.08082.
- Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. 2025. [Reasoning with latent thoughts: On the power of looped transformers](#). *Preprint*, arXiv:2502.17416.
- KaShun Shum, Shizhe Diao, and Tong Zhang. 2024. [Automatic prompt augmentation and selection with chain-of-thought from labeled data](#). *Preprint*, arXiv:2302.12822.

- Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. [Scaling llm test-time compute optimally can be more effective than scaling model parameters](#). *Preprint*, arXiv:2408.03314.
- Wenhui Tan, Jiaze Li, Jianzhong Ju, Zhenbo Luo, Ruihua Song, and Jian Luan. 2026. [Think silently, think fast: Dynamic latent compression of llm reasoning chains](#). *Preprint*, arXiv:2505.16552.
- Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, Mayank Chaturvedi, Victor Cotruta, Alice Coucke, Phil Culliton, Robert Dadashi, Lucas Dixon, Mohamed Elhawaty, Utku Evci, Clément Farabet, and 282 others. 2026. [Gemma 4 technical report](#). *Preprint*, arXiv:2607.02770.
- Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. [Gemma 3 technical report](#). *Preprint*, arXiv:2503.19786.
- Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. 2025. [Hierarchical reasoning model](#). *Preprint*, arXiv:2506.21734.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. [Chain-of-thought prompting elicits reasoning in large language models](#). *Preprint*, arXiv:2201.11903.
- Paul Werbos. 1974. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA*.
- Weiqiu You, Anton Xue, Shreya Havaldar, Delip Rao, Helen Jin, Chris Callison-Burch, and Eric Wong. 2025. [Probabilistic soundness guarantees in llm reasoning chains](#). *Preprint*, arXiv:2507.12948.
- Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, and 14 others. 2025. [Scaling latent reasoning via looped language models](#). *Preprint*, arXiv:2510.25741.

A Benchmark Examples

Dataset	Example problem	Gold answer
GSM8K	Janet’s ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers’ market daily for \$2 per fresh duck egg. How much in dollars does she make every day at the farmers’ market?	18
SVAMP	There were 106 dollars in Olivia’s wallet. After she visited a supermarket and a showroom there were 26 dollars left. If she spent 49 dollars at the showroom, how much did she spend at the supermarket?	31

Table 3: Representative examples from the GSM8K and SVAMP evaluation sets used in our experiments.

B Model Configurations

Model	Params	Key configuration	Generation paradigm
GPT-2-style (scratch)	73.5M	12 layers, hidden size 512, 8 attention heads, FFN size 2816, context length 1024	Autoregressive decoder-only Transformer
Gemma-4-Text-style (scratch)	77.6M	12 layers, hidden size 512, 8 query heads, 4 key/value heads, head dimension 64, FFN size 2304, sliding window 512	Autoregressive attention-based Transformer with grouped-query attention
Mamba-2-style (scratch)	76.0M	19 layers, hidden size 512, expansion factor 2, state size 128, 8 groups, 16 heads, head dimension 64	Autoregressive recurrent state-space model
Diffusion-Gemma-style (scratch)	74.3M	12 layers, hidden size 512, 8 query heads, 4 key/value heads, head dimension 64, one expert, canvas length 256	Text-only block diffusion with iterative denoising
TRLM (ours)	74.5M	12 shared Transformer layers, hidden size 512, 8 attention heads, grouped-query attention, SwiGLU size 2048, maximum recursion depth 16	Masked non-autoregressive generation with recursive latent refinement

Table 4: Controlled 75M-class model configurations. All models are trained from scratch on the same OpenWebText/FineWeb-Edu mixture and use the same tokenizer and pretraining token budget. Names ending in “style” denote architecture-family baselines, not official pretrained checkpoints.