

Scaling Latent Reasoning via Looped Language Models

Rui-Jie Zhu^{*1,2,†}, Zixuan Wang^{*1,3}, Kai Hua^{*1}, Tianyu Zhang^{*4,5}, Ziniu Li^{*1}, Haoran Que^{*1,6}, Boyi Wei^{*3}, Zixin Wen^{*1,7}, Fan Yin^{*1}, He Xing^{*11}, Lu Li⁸, Jiajun Shi¹, Kaijing Ma¹, Shanda Li^{1,7}, Taylor Kergan^{2,9}, Andrew Smith^{2,9}, Xingwei Qu^{1,10}, Mude Hui², Bohong Wu¹, Qiyang Min¹, Hongzhi Huang¹, Xun Zhou¹, Wei Ye⁶, Jiaheng Liu¹¹, Jian Yang¹¹, Yunfeng Shi¹¹, Chenghua Lin¹⁰, Enduo Zhao¹¹, Tianle Cai¹, Ge Zhang^{*1,†}, Wenhao Huang^{1,†}, Yoshua Bengio^{4,5}, Jason Eshraghian^{2,†}

¹ByteDance Seed, ²UC Santa Cruz, ³Princeton University, ⁴Mila - Quebec AI Institute, ⁵University of Montreal, ⁶Peking University, ⁷Carnegie Mellon University, ⁸University of Pennsylvania, ⁹Conscium, ¹⁰University of Manchester, ¹¹M-A-P

*Core Contributors, †Corresponding authors

Abstract

Modern LLMs are trained to “think” primarily via explicit text generation, such as chain-of-thought (CoT), which defers reasoning to post-training and under-leverages pre-training data. We present and **open-source** Ouro, named after the recursive Ouroboros, a family of pre-trained Looped Language Models (LoopLM) that instead build reasoning into the pre-training phase through (i) iterative computation in latent space, (ii) an entropy-regularized objective for learned depth allocation, and (iii) scaling to **7.7T** tokens. Ouro 1.4B and 2.6B models enjoy superior performance that match the results of up to 12B SOTA LLMs across a wide range of benchmarks. Through controlled experiments, we show this advantage stems not from increased knowledge capacity, but from superior knowledge manipulation capabilities. We also show that LoopLM yields reasoning traces more aligned with final outputs than explicit CoT. We hope our results show the potential of LoopLM as a novel scaling direction in the reasoning era.

Correspondence: ridger@ucsc.edu, zhangge.eli@bytedance.com, huang.wenhao@bytedance.com, jsn@ucsc.edu
Project Page & Base / Reasoning Models: <http://ouro-llm.github.io>

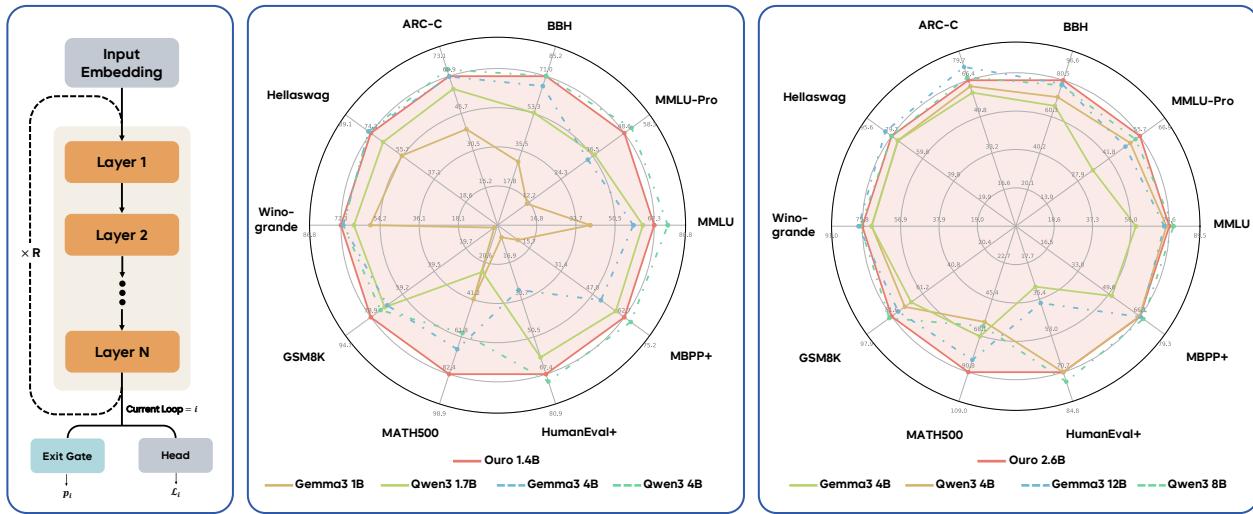


Figure 1 Ouro Looped Language Model performance. (Left) The parameter-shared looped architecture. (Middle & Right) Radar plots comparing the Ouro 1.4B and 2.6B models, both with 4 recurrent steps (red), against individual transformer baselines. Our models demonstrate strong performance comparable to or exceeding much larger baselines.

1 Introduction

The advancement of Large Language Models (LLMs) has historically relied on scaling up model size as the primary driver, accompanied by increases in data and compute [1–4]. However, deploying models with hundreds of billions of parameters requires extensive infrastructure, increasing latency and cost while limiting accessibility. These factors make *parameter efficiency* critical: achieving better model capability within a fixed parameter budget. Such models not only mitigate overfitting on finite datasets with fewer trainable parameters, but also enable more practical deployment with lighter infrastructure. To achieve such parameter efficiency, two main avenues have been explored. The first expands the training corpus regardless of model size [5], though data scarcity increasingly limits this path. The second leverages inference-time compute through Chain-of-Thought (CoT) reasoning [6], allowing models to spend more compute on complex problems via extended token generation.

We explore a third pathway based on architectural innovation: achieving dynamic computation within a fixed parameter budget. This is accomplished by recursively reapplying shared parameters, where a group of weight-tied layers are iteratively reused during the forward pass. We call this the Looped Language Model (LoopLM). The LoopLM paradigm has gained wide attention and shown strong promise recently, originating with the seminal Universal Transformer [7]. Plenty of works on looped transformers [8–13], recursive transformers [14], and latent reasoning frameworks [15, 16] have demonstrated the benefits of deeper computational processing on equivalent training data, indicating the potential benefits at a larger scale. These approaches share a common principle: through iterative reuse of weight-shared layers, computational scaling is decoupled into the depth of computation and the number of parameters involved.

The LoopLM design yields several advantages. First, LoopLM allows adaptive computation via a learned early exit mechanism: simple inputs may terminate after fewer recurrent steps, while complex ones naturally allocate more iterations, enabling flexible computational depth without increasing the parameter count. Moreover, unlike inference-time methods like CoT, LoopLM achieves scalability by deepening its internal computational graph rather than extending the output sequence, thereby avoiding excessive context growth. Finally, LoopLM potentially improves capacity per parameter and achieves stronger performance than standard transformers of comparable size when trained on the same data. While the prior studies have demonstrated the benefits of LoopLM at modest scales, there is little evidence that Looped Language Models can perform on par with non-looped architecture at practically meaningful scales. To this end, we ask the following question:

Does LoopLM exhibit more favorable scaling behavior (in loss, efficiency, safety, or capability growth), compared to existing non-recursive transformer models?

In this work, we aim to answer this question affirmatively. We carefully investigate the LoopLM scaling trajectory to understand its point of saturation, illustrating that LoopLM represents a fundamentally more efficient path towards higher performance. While prior works have shown promise on modest scales (e.g., hundreds of billions of tokens), we study whether these benefits persist or even amplify when scaled to the multi-trillion token regimes of training that define SOTA foundation models. In particular, we study the *mechanisms* of LoopLM by further asking the following questions:

1. Does the recursive application of weight-shared layers enhance the LoopLM’s capabilities in a manner analogous to increasing the number of unshared layers?
2. Are the performance gains LoopLM monotonic in the number of loops? Are there more factors involved in the use of the adaptive computation of LoopLM that are different with prior empirical results in smaller scale experiments?

To resolve these uncertainties, we move beyond small-scale proofs of concept and undertake a comprehensive empirical investigation. Our results are presented below.

Our Contribution

In this work, we attempt to address the above research questions through a multi-faceted approach. We scale up the pretraining of LoopLM to a total of 7.7T tokens and thoroughly investigate the scaling behavior of LoopLM in various aspects. To achieve adaptive computation with LoopLM, we develop novel training

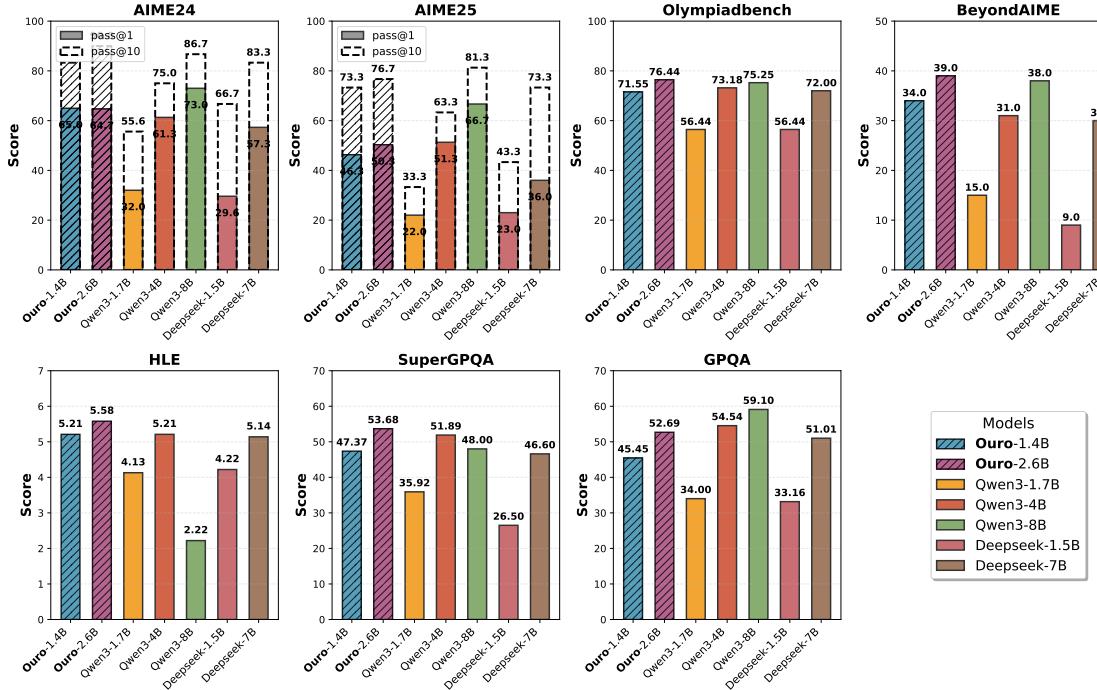


Figure 2 Performance on advanced reasoning benchmarks. Ouro-Thinking models compared with strong baselines such as Qwen3 and DeepSeek-Distill. **Ouro-1.4B-Thinking R4** is competitive with 4B models, and **Ouro-2.6B-Thinking R4** matches or exceeds 8B models across multiple math and science datasets.

objectives that enable computationally efficient recurrent computation while maintaining peak performance. We further conducted several controlled experiments on synthetic tasks to understand the mechanism of LoopLM superiority. Specifically, our contributions are:

- **Exceptional parameter efficiency at scale.** By pre-training on 7.7 trillion tokens, we demonstrate that 1.4B and 2.6B parameter LoopLMs match the performance of 4B and 8B standard transformers respectively across nearly all benchmarks, achieving 2-3× parameter efficiency improvements critical for deployment under resource constraints. As illustrated in Figure 1 and Figure 2, our Ouro and Ouro-Thinking models demonstrate exceptional parameter efficiency across a suite of benchmarks, with our 1.4B and 2.6B parameter models matching the performance of larger non-recurrent LLMs.
- **Entropy regularized adaptive computation.** We develop an entropy regularized training objective with a uniform prior over exit steps that enables unbiased exploration of all recurrent depths. Moreover, we add a focused training stage for adaptive gates training, specifically to optimize the tradeoff between computation efficiency and loop performance, allowing learned adaptive allocation of recurrent steps based on input difficulty.
- **Mechanistic understanding via synthetic tasks.** Using controlled experiments inspired by the physics of language models framework, we show that recurrence does not increase raw knowledge storage (approximately 2 bits per parameter for both looped and non-looped models) but dramatically enhances knowledge manipulation capabilities on tasks requiring fact composition and multi-hop reasoning.
- **Improved safety and faithfulness.** LoopLM architectures reduce harmfulness on HEx-PHI [17], with safety improving as recurrent steps increase, including extrapolated steps. In contrast to CoT, our iterative latent updates yield a causally faithful reasoning process rather than post hoc rationalization.

Our study of LoopLM establishes the number of recursion as a third scaling axis beyond model size and data, and we publicly release the Ouro model family (1.4B and 2.6B parameters) to demonstrate the benefits of LoopLM at scale.

2 Related Works

We define notation and confine background only to what is needed for presenting our method. A standard L -layer Transformer is a composition of layer functions. It processes an input sequence of hidden states $H^{(0)} \in \mathbb{R}^{N \times d}$ through a series of unique layers, parameterized by $\theta_1, \dots, \theta_L$:

$$H^{(L)} = \text{TransformerLayer}_{\theta_L}(\dots \text{TransformerLayer}_{\theta_2}(\text{TransformerLayer}_{\theta_1}(H^{(0)})) \dots)$$

In contrast, the Universal Transformer [7] replaces L distinct layers with a recurrent application of a single Transformer block repeatedly for T updates:

$$H^{(t)} = \text{TransformerLayer}_{\theta}(H^{(t-1)}), \quad \text{for } t = 1, \dots, T, \text{ with } H^{(0)} \text{ as input}$$

The core ideas of this architecture have resurfaced in recent literature, with recurrent-depth structures used to improve the efficiency and reasoning capabilities of modern LLMs. For example, Geiping et al. [15] adopts a “recurrent depth” to scale test-time computation in latent space. Similarly, Saunshi et al. [8] demonstrates that “looped transformers” can match the performance of much deeper non-looped models on reasoning tasks, formally connecting looping to the generation of latent thoughts. The approach is refined by converting standard models into “Relaxed Recursive Transformers” with a common base block while injecting unique LoRA adapters across recursive steps [14]. Similar concepts have emerged under different terms, such as “pondering” in continuous space [16] and “inner thinking” for adaptive computation [18]. More advanced variants, such as Mixture-of-Recursions [19] combine recursive parameter efficiency with adaptive, token-level routing.

Across all these works, from the original Universal Transformer to its modern descendants, this emerging line of architectures can be understood in two complementary ways. From one perspective, it behaves like a deep Transformer where the weights of all layers are tied. From another, iteration functions as latent reasoning, where the hidden states $H^{(1)}, \dots, H^{(T)}$ form a latent chain of thought that progressively refines the representation to solve a task. Taken together, these results suggest that models can improve their ability to reason by reusing computation internally without having to increase parameter count, shifting scale to substance.

Perspective 1: Parameter Sharing for Model Efficiency. This view treats LoopLM as parameter sharing: one or more Transformer blocks, or even submodules (e.g., attention, FFN), are reused across the depth of the model, reducing parameters without changing the computation. The most prominent example in the modern transformer era is ALBERT [20], which combines parameter re-use with embedding factorization to drastically reduce the total parameter count. Prior to the widespread adoption of LLMs, parameter sharing was explored extensively in machine translation [21]; Takase et al. [22] systematically studied sharing strategies to balance compression and accuracy. Interest in parameter reuse dropped as models grew larger, but it has resurged to shrink the memory footprint of LLMs. For example, Megrez2 [23] reuses experts across layers in a standard Mixture-of-Experts (MoE) model, and shows a viable path forward for edge LLM deployment with limited memory.

Perspective 2: Latent Reasoning and Iterative Refinement. Here, the LoopLM’s iteration is viewed as latent reasoning where each step is a non-verbal “thought” that refines the model’s internal representation. Empirically, increasing the number of recurrent steps improves performance on complex reasoning tasks [8, 15]. Some models make this process explicit by feeding hidden states back into the input. Coconut inserts a “continuous thought” token, which is derived from the previous step’s last-layer hidden state, so the model can “ponder” in a continuous latent space [24]. CoTFormer interleaves activations back into the input before reapplying this augmented sequence to the shared layers [25]. These explicit feedback loops contrast with implicit LoopLM variants, where the entire thought process is contained within the evolution of hidden states from $H^{(t-1)}$ to $H^{(t)}$. Thus, both Perspective 1 (model compression) and Perspective 2 (latent reasoning) leverage shared-parameter iteration to improve parameter efficiency, and are being explored for enhanced reasoning and efficient sequence-length expansion (e.g., PHD-Transformer [26]).

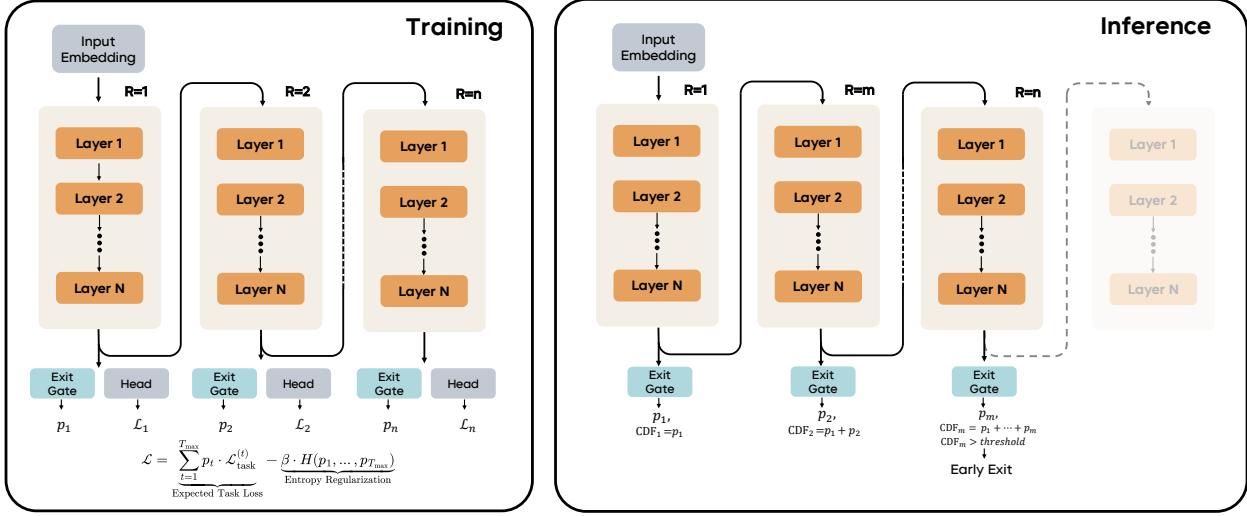


Figure 3 Overview of Looped Language Model (LoopLM) architecture. **Left (Training):** During training, the model applies a stack of N shared-weight layers for n recurrent steps ($R = 1$ to $R = n$). At each recurrent step i , an exit gate predicts the probability p_i of exiting, and a language modeling head \mathcal{L}_i computes the task loss. The training objective combines the expected task loss across all steps with an entropy regularization term $H(p_1, \dots, p_n)$ to encourage exploration of different computational depths. **Right (Inference):** At inference time, the model can exit early based on the cumulative distribution function (CDF) computed from exit probabilities. When $CDF_i = \sum_{k=1}^i p_k$ exceeds a threshold, the model terminates at step i , enabling adaptive computation that allocates more steps to harder inputs while maintaining efficiency on simpler ones. The dashed line indicates potential future steps that may be skipped through early exit.

3 Learning Adaptive Latent Reasoning with LoopLM

In this section, we shall formally define the LoopLM architecture based on (causal) transformers and explain how we train LoopLMs that are able to perform latent reasoning with adaptive computation. Figure 3 illustrates our architecture during both training and inference phases. Our objective is to let the model decide how many recurrent steps to use for each token and each example, spending less compute on easy inputs and more compute on hard inputs, without sacrificing accuracy when many steps are available.

3.1 LoopLM Architecture

Let $\text{Tr}_\theta(\cdot) : \mathbb{R}^{M \times d} \rightarrow \mathbb{R}^{M \times d}$ denote a causal transformer layer equipped with parameter θ , with hidden dimension d and input length M . Moreover we let $\text{lmhead}(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}^{|V|}$ denote the language-modeling head where V is the vocabulary of tokens, and $\text{emb}(\cdot) : \mathbb{R}^{|V|} \rightarrow \mathbb{R}^d$ denote the embedding layer. A typical non-looped causal language model is defined by stacking L layers as follows:

$$F(\cdot) := \text{lmhead} \circ H^L \circ \text{emb}(\cdot), \quad \text{where } H^L(\cdot) := \text{Tr}_{\theta_L} \circ \dots \circ \text{Tr}_{\theta_1}(\cdot) \text{ is the hidden layers.}$$

Let $t \in \mathbb{N}, t \leq T_{\max}$ be the number of loop steps, which we also call the number of **recurrent steps** or **recurrent depth**, we define the looped language model $F^{(t)}$ by:

$$F^{(t)}(\cdot) = \text{lmhead} \circ \underbrace{H^L \circ H^L \circ \dots \circ H^L}_{t \text{ times}} \circ \text{emb}(\cdot). \quad (1)$$

Apparently $F^{(1)} \equiv F$ is the non-loop model. Given a sequence of tokens $x_{1:M} = (x_1, x_2, \dots, x_M)$ of length M , a LoopLM model $F^{(t)}$ acts as a sequence to sequence model and produces M tokens $y_{1:M} = (y_1, y_2, \dots, y_M)$ that is consistent with the causal dependency condition that $y_{1:m} = F^{(t)}(x_{1:m})$ and for any $m \leq M$. The

Algorithm 1 Q-exit Early Stopping Criterion

Require: Input x ; maximum steps T_{\max} ; threshold $q \in [0, 1]$;

- 1: $cdf \leftarrow 0$; $surv \leftarrow 1$ ▷ $surv = \prod_{j=1}^{t-1} (1 - \lambda_j)$
- 2: **for** $t = 1$ **to** T_{\max} **do**
- 3: $\lambda_t \leftarrow \sigma(\text{Gate}(F^{(t)}(x)))$
- 4: $q_\phi(t|x) \leftarrow \lambda_t \cdot surv$
- 5: $cdf \leftarrow cdf + p_t$ ▷ $\text{CDF}(t | x)$
- 6: **if** $cdf \geq q$ **then**
- 7: **return** t ▷ $t_{\text{exit}}(x)$
- 8: $surv \leftarrow surv \cdot (1 - \lambda_t)$
- 9: **return** T_{\max} ▷ fallback if the threshold is never reached

next-token-prediction objective is simply the cross-entropy loss on all $y_i, 1 \leq i \leq M$,

$$\mathcal{L}^{(t)} = \mathbb{E}_{x_{1:M}} \left[\sum_{1 \leq i \leq M-1} -\log \mathbf{Pr}(F^{(t)}(x_{1:i}) = x_{2:i+1}) \right], \quad \text{for } t \in [1, T_{\max}] \quad (2)$$

where $\mathbf{Pr}(F^{(t)}(x_{1:i}) = x_{i+1})$ is computed by taking a softmax over the lmhead output. The loss $\mathcal{L}^{(t)}$ measures the precision of a t -step LoopLM. Prior literature such as [8, 12] have shown that scaling up the loop count t is beneficial for reasoning tasks in smaller scale. However, increasing the loop count t costs computation, and not all language tokens are reasoning-heavy and need those computation to be predicted correctly. In fact, many tokens in the next-token-prediction tasks are simple or can be learned with high certainty [27, 28]. Thus it is crucial to spend the computation budget on the right tokens for pareto-optimal allocation between performance and efficiency. This is achieved by the **gating mechanism** we shall describe below.

3.2 Adaptive Computation via Gating Mechanism

The early exit gate at loop step $t \leq T_{\max}$ is defined by $\lambda_t(x) = \sigma(\text{Gate}(F^{(t)}(x)))$ for an input sequence x through a learned linear projection followed by sigmoid activation σ . After that, LoopLM uses the gates $\{\lambda_t(x)\}_{t \in [1, T_{\max}]}$ to compute a distribution $q_\phi(\cdot|x) \in \Delta^{T_{\max}}$ over $\{1, \dots, T_{\max}\}$ ¹ to decide whether to continue the recursive computation or stop to yield the output. Following [29], we adopt a deterministic **Q-exit criterion** based on the cumulative distribution function (CDF), described in [Algorithm 1](#). At each step t , we compute the following:

$$\text{CDF}(t|x) = \sum_{i=1}^t q_\phi(i|x) = \sum_{i=1}^t \lambda_i(x) \prod_{j=1}^{i-1} (1 - \lambda_j(x))$$

and $q_\phi(i|x)$ represents the probability of exiting exactly at step i given input x . The product term $\prod_{j=1}^{i-1} (1 - \lambda_j)$ ensures that we can only stop at step i if we did not exit at any earlier step $j < i$. For input x , we exit early when the CDF exceeds a set threshold $q \in [0, 1]$:

$$t_{\text{exit}}(x) = \min\{t : \text{CDF}(t|x) \geq q\}$$

The threshold q controls the compute-accuracy tradeoff, where lower values encourage earlier exits and higher values allow deeper computation. The gating function λ_t shall be learned in the two-stage training:

- **Stage I:** during pre-training, the gates $\lambda_t(\cdot)$ are learned by optimizing an entropy-regularized objective. In this stage, the exit distribution aims to widen the coverage of the optimal exit step.
- **Stage II:** we focus on training only the gates in this stage. The objective exploits the pattern learned in stage I and sharpens the distribution by choosing the optimal exit step with a soft cross-entropy loss.

We shall immediately explain the two stages below.

¹ $\Delta^d = \{x \in [0, 1]^d \mid \sum_{i=1}^d x_i = 1\}$ is the probability simplex.

3.3 Stage I: Learning An Entropy-Regularized Objective

To obtain a LoopLM that can exit early while maintaining performance, we need an objective that jointly optimizes accuracy and computational efficiency. Let $\mathcal{L}^{(t)}$ follow (2), then our training objective combines the next-token prediction loss with an entropy regularization term over q_ϕ :

$$\mathcal{L} = \underbrace{\sum_{t=1}^{T_{\max}} q_\phi(t|x) \cdot \mathcal{L}^{(t)}}_{\text{expected task loss}} - \underbrace{\beta \cdot H(q_\phi(\cdot|x))}_{\text{entropy regularization}} \quad H(q_\phi(\cdot|x)) = -\sum_{t=1}^{T_{\max}} q_\phi(t|x) \log q_\phi(t|x) \quad (3)$$

where $\mathcal{L}^{(t)}$ is the next token prediction loss at loop step t , and $H(q_\phi(\cdot|x))$ is the entropy of the exit step distribution. The hyperparameter β controls the exploration-exploitation trade-off: larger β encourages more uniform distributions (higher entropy), allowing the model to explore different depths, while smaller β allows more concentrated distributions when the model is confident about the optimal depth for a given input.

Alternative perspective: variational inference with uniform prior. The entropy-regularized objective can be equivalently viewed through the lens of variational inference. If we treat the exit step as a latent variable z with a prior distribution π , we can derive an Evidence Lower Bound (ELBO) objective:

$$\mathcal{L}_{\text{ELBO}} = \sum_{t=1}^{T_{\max}} q_\phi(t|x) \cdot \mathcal{L}^{(t)} + \beta \cdot \text{KL}(q_\phi(\cdot|x) \| \pi)$$

When we choose a uniform prior $\pi_t = 1/T_{\max}, \forall t$, the KL divergence simplifies to:

$$\text{KL}(q_\phi(\cdot|x) \| \pi) = -H(q_\phi(\cdot|x)) + \log T_{\max}$$

Since $\log T_{\max}$ is constant, minimizing the ELBO with a uniform prior is equivalent to our entropy-regularized objective. This connection reveals that our approach aligns with adaptive computation methods like PonderNet [30], which also optimize an ELBO objective for dynamic halting.

Why uniform prior? While our formulation is similar to PonderNet [30], a critical difference lies in the choice of prior. PonderNet and other adaptive computation methods typically employ geometric priors:

$$\pi_t^{\text{geo}} = \frac{\lambda (1-\lambda)^{t-1}}{1 - (1-\lambda)^{T_{\max}}}, \quad t = 1, \dots, T_{\max}, \quad \lambda \in (0, 1)$$

Similarly, methods like Recurrent Depth [15] use Poisson-lognormal priors that also favor shallow computation. We argue that such priors conflate two distinct goals: (1) learning when to exit based on input difficulty, and (2) minimizing average computation cost. In fact, these priors contain a strong inductive bias toward shallow computation by placing more mass on early steps, explicitly encouraging the model to exit early. Therefore, they risk under-exploring deeper steps and may fail to fully exploit the benefits of recurrent depth.

In contrast, the uniform prior makes no assumptions about the optimal exit step distribution. It optimizes all depths uniformly, enabling the model to infer the computational requirements of different inputs without relying on predefined inductive biases. This is particularly important for complex reasoning tasks, where the optimal depth should be learned from the data rather than being constrained by the prior. We provide a detailed empirical validation of this choice against geometric priors in [Section A](#).

3.4 Stage II: Focused Adaptive Gate Training

Unlike traditional approaches that treat the gating mechanism as an auxiliary component, we directly optimize the exit gate for effective adaptive computation. The core idea of our gate training approach is to teach the model to make termination decisions based on actual performance improvements. We adopt a greedy approach that optimizes the trade-off between improvement via recurrent step and computation efficiency below.

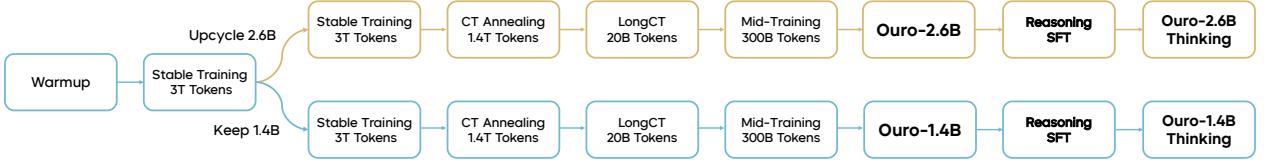


Figure 4 The Ouro model training pipeline. The process starts with a common Warmup and an initial 3T token Stable Training phase. The model is then split into two streams: one ‘Keep 1.4B’ (resulting in Ouro-1.4B) and one ‘Upcycle 2.6B’ (resulting in Ouro-2.6B). Both streams independently undergo an identical subsequent four-stage training process: a second Stable Training (3T tokens), CT Annealing (1.4T tokens), LongCT (20B tokens), and Mid-Training (300B tokens). This 7.7T token pre-training pipeline produces the base models (Ouro-1.4B and Ouro-2.6B), which are then passed through a final Reasoning SFT stage to create the Ouro-Thinking models.

In this stage, the adaptive exit objective must exclusively optimize the gating mechanism without interfering with the language model’s learned representations. We compute the detached loss $\mathcal{L}_{i,\text{stop}}^{(t)}$ at each token position i and define the loss improvement of advancing from step $t - 1$ to step t by

$$I_i^{(t)} = \max(0, \mathcal{L}_{i,\text{stop}}^{(t-1)} - \mathcal{L}_{i,\text{stop}}^{(t)}) \quad (4)$$

Intuitively, when $I_i^{(t)}$ is small, the improvement has stagnated and LoopLM should opt for an early exit. We implement this intuition by computing the **ideal continuation probability** $w_i^{(t)} = \sigma(k \cdot (I_i^{(t)} - \tau))$ using $k = 50.0$ as the slope and $\tau = 0.005$ as the improvement threshold. When $I_i^{(t)} > \tau$, $w_i^{(t)}$ is large and the model performs the next recurrent step; When $I_i^{(t)} \ll \tau$, early exit is favored in step t . The adaptive exit loss at step t takes the form of a weighted cross-entropy averaged over the sequence length M between this ideal behavior and the actual gate prediction.

$$\mathcal{L}_{\text{adaptive}}^{(t)} = -\frac{1}{M} \sum_{i \leq M} \left[w_i^{(t)} \log(1 - \lambda_i^{(t)}) + (1 - w_i^{(t)}) \log(\lambda_i^{(t)}) \right] \quad (5)$$

The total adaptive training loss is averaged across all recurrent steps:

$$\mathcal{L}_{\text{adaptive}} = \frac{1}{T_{\text{max}}} \sum_{t=2}^{T_{\text{max}}} \mathcal{L}_{\text{adaptive}}^{(t)}$$

Significance of our adaptive loss. The adaptive loss (5) trains the gate in step t by matching its weights $\lambda_i^{(t-1)}$ with the ideal probability $w_i^{(t)}$. This formulation penalizes two failure modes simultaneously:

- **Underthinking**, where the gate wants to stop when it should continue (when the ideal continuation probability $w_i^{(t)}$ is large, but the early stop gate $\lambda_i^{(t-1)}$ is also large);
- **Overthinking**, where the gate wants to continue when it should stop (when the ideal continuation probability $w_i^{(t)}$ is small, but $\lambda_i^{(t)}$ is also small).

By optimizing (5), the gates learn to greedily search for the optimal exit step, to tradeoff computation efficiency for better performance. For empirical evaluations, see [Section 5.4.1](#).

4 Training Looped Language Models

Our training pipeline for the Ouro model family is a multi-stage process, as illustrated in Figure 4. The process begins with a common warmup stage, followed by an initial Stable Training phase on 3T tokens. After this, the model is split into two variants: a 1.4B parameter model and a 2.6B model created via upcycling. Both variants then undergo an identical series of four subsequent training stages: a second Stable Training phase (3T tokens), CT Annealing (1.4T tokens), LongCT (20B tokens), and Mid-Training (300B tokens). This comprehensive pipeline, totaling 7.7T tokens of training data, produces our base models, **Ouro-1.4B** and

Ouro-2.6B. Finally, these base models are further refined through a specialized Reasoning SFT (Supervised Fine-Tuning) stage to create the final, reasoning-focused models: **Ouro-1.4B-Thinking** and **Ouro-2.6B-Thinking**. This section details the model architecture, data composition, and specific configurations used in each of these training stages.

4.1 Transformer Architecture

Our Ouro models are built upon the standard decoder-only Transformer architecture [31], prioritizing a clean implementation of the looped computation mechanism without extraneous modifications. The core architecture consists of a stack of identical Transformer blocks, which are applied recurrently.

Each block uses Multi-Head Attention (MHA) with Rotary Position Embeddings (RoPE) [32] to handle sequence order. For computational efficiency, The feed-forward network (FFN) in each block utilizes a SwiGLU activation [33]. To enhance training stability, which is especially critical for deep recurrent computation, we employ a **sandwich normalization** structure. This places an RMSNorm layer before both the attention and FFN sub-layers, an approach noted in prior literature to improve stability in loop transformers [15].

Table 1 Ouro model architecture configurations. Both models share the same vocabulary and core component types, differing in parameter count and layer depth.

Model	Parameters	Layers	Hidden Size (d_{model})	Attention	FFN	Pos. Embed.	Vocab Size
Ouro 1.4B	1.4B	24	2048	MHA	SwiGLU	RoPE	49,152
Ouro 2.6B	2.6B	48	2048	MHA	SwiGLU	RoPE	49,152

For both models, we use a shared vocabulary of 49,152 tokens, reused from the SmolM2 [34] model. This tokenizer is optimized for languages with a Latin alphabet and code and contains no Chinese tokens. Our inclusion of Chinese data in Stage 1 thus resulted in highly inefficient tokenization and poor performance. Consequently, we removed all Chinese data from Stage 2 onwards to focus our training budget on English and code. This limited vocabulary may also impose constraints on the model’s advanced mathematical reasoning capabilities due to a potential lack of specialized symbols. This shift is reflected in the data compositions detailed in the following sections.

4.2 Data

As data defines the capability boundaries of large foundational models, our model is trained on a diverse collection of datasets spanning multiple domains and stages, including web data, mathematical content, code, and long-context documents, enabling it to perform acquire fundamental language understanding and perform advanced reasoning, coding, and long-context understanding through a unified training pipeline. In addition to standard web crawl datasets, we adopt specialized datasets for mathematical reasoning and code generation to further enhance the model’s capabilities for complex problem-solving. In Table 2, we summarize the composition and quantity of our training data across different stages. In the following sections, we detail our dataset sources, preparation protocols, and data mixing strategies.

4.2.1 Data Composition

The capabilities of modern language models primarily stem from their training data, and this principle holds true for our model as well. To ensure reproducibility, our training corpus is entirely composed of open-source datasets, with data statistics summarized in Table 3. We partition the data into four distinct subsets, called stages, each employing different data construction strategies that align with the Warmup-Stable-Decay (WSD) [35] learning rate scheduler, which is widely adopted in modern language model pretraining.

Stage 1: Pre-training The pre-training stage supports the warmup and stable phases of training. The dataset is primarily composed of Web CommonCrawl (CC) data. Since our objective is to train the model on more than 2T tokens, many commonly used open-source datasets would not suffice: Fineweb-Edu [36] provides 1.3T tokens, and DCLM [37] offers 2.6T tokens. We select Nemotron-CC [38] (6.3T tokens) as the main

Table 2 Statistics of the training corpus. Since data are randomly sampled during pre-training, the dataset size does not directly correspond to the total number of seen tokens.

Data Source	Stage	# Tokens (B)	# Used Tokens (B)
Nemotron-CC (Web Data)	Stage 1	6386	4404
MAP-CC (Web Data)	Stage 1	800	780
Ultra-FineWeb-zh (Web Data)	Stage 1	120	120
OpenCoder-pretrain	Stage 1	450	450
MegaMath-web	Stage 1	247	246
MegaMath-high-quality	Stage 2	64	64
Nemotron-CC-Math-v1	Stage 2	210	210
Nemotron-Code	Stage 2	53	53
Nemotron-SFT-Code	Stage 2	48	48
Nemotron-SFT-General	Stage 2	87	87
OpenCoder-Annealing	Stage 2	7	7
ProLong-64K	Stage 3	20	20
Mid-training SFT Mix	Stage 4	182	90

Table 3 Data composition for Stage 1 (Pre-training Phase I & II). Total dataset size: 6T tokens.

Data Source	Nemotron-CC	MAP-CC	Ultra-FineWeb-zh	OpenCoder-pretrain	MegaMath-web
Proportion (%)	73.4	13.0	2.0	7.5	4.1

dataset for the stable phase due to its large scale and suitability for our training requirements. To provide the model with basic Chinese proficiency, we include Ultra-FineWeb-zh [39] and MAP-CC [40]. Additionally, to enhance coding and mathematical abilities, we incorporate OpenCoder [41] and MegaMath [42].

Stage 2: Continual Training (CT) Annealing The CT annealing stage incorporates higher-quality data to enhance the model under the annealing learning rate. We construct our dataset using the high-quality subset of Nemotron-CC as the base. To further strengthen mathematical, coding, and general capabilities, we incorporate the high-quality subset of MegaMath, Nemotron-CC-Math-v1 [43, 44], OpenCoder-Annealing [41], Nemotron-Pretraining-Code-v1 [44], and Nemotron-Pretraining-SFT-v1 [44].

Table 4 Data composition for Stage 2 (CT Annealing). Total dataset size: 1.4T tokens.

Data Source	Proportion (%)
Nemotron-CC-high-quality	66.5
Nemotron-CC-Math-v1	15.0
MegaMath-high-quality	4.6
OpenCoder-LLM/opc-annealing-corpus	0.5
Nemotron-Pretraining-Code-v1/Synthetic-Code	3.8
Nemotron-Pretraining-SFT-v1/Nemotron-SFT-Code	3.4
Nemotron-Pretraining-SFT-v1/Nemotron-SFT-General	6.2

Stage 3: Long Context Training (LongCT) The LongCT stage extends the long-context capabilities of the model. We adopt the 64K-length subset of ProLong [45], consisting of 20B tokens, to train the model on longer sequences and improve its ability to handle long contexts.

Stage 4: Mid-training The mid-training stage leverages a wide and diverse set of extremely high-quality data, consisting of both \langle Question, Answer \rangle and \langle Question, CoT, Answer \rangle samples, to further enhance the model’s advanced abilities. We integrate more than 20 open-source supervised fine-tuning (SFT) datasets to maximize data diversity, while conducting thorough decontamination to minimize potential overlaps with mainstream evaluation benchmarks. All samples are converted into ChatML format to reduce alignment

tax in the subsequent post-training stage. After processing the previous datasets, we obtain a total of 182B tokens, from which we sample 90B tokens to form the newly incorporated datasets. To ensure stable data distribution during training, 30B tokens from Stage 1 and 180B tokens from Stage 2 are replayed, resulting in an effective training volume of 300B tokens. Consequently, this stage is designed to push the model to the limits of its advanced abilities developed during pretraining.

4.3 Pre and Mid-Training

We adopt a multi-stage training strategy using a dynamic mixture of the curated data described, specifically: a Pre-training stage (split into two phases with different recurrent configurations), a CT Annealing stage for quality enhancement, a LongCT stage for context extension, and a Mid-training stage for advanced capability refinement. Throughout our training pipeline, we train the model with a maximum of **4 recurrent steps**.

4.3.1 Training Stability and Adaptive Configuration

During training, we prioritized stability over aggressive scaling, making several key adjustments based on empirical observations of training dynamics. These decisions were critical for achieving stable convergence with recurrent architectures, which exhibit different optimization characteristics compared to standard transformers.

Recurrent Step Reduction for Stability. Our initial experiments with 8 recurrent steps in Stage 1a reveal training instabilities, including loss spikes and gradient oscillations. We hypothesize this stems from the compounding gradient flow through multiple recurrent iterations, which can amplify small perturbations. Consequently, we reduced the recurrent steps from 8 to 4 in Stage 1b, finding this sweet spot balanced computational depth with training stability.

Batch Size Scaling. To further enhance stability, we progressively increased the batch size from 4M to 8M tokens. Larger batch sizes provide more stable gradient estimates, which is particularly important for recurrent architectures where gradient flow through multiple iterations can introduce additional variance.

KL Divergence Coefficient Reduction. We strategically reduced β from 0.1 in Stage 1a to 0.05 in subsequent stages. This reduction serves dual purposes: (1) it decreases the conflicting gradients between task loss and the KL penalty, leading to more stable optimization, and (2) it reduces the “pull” from the uniform

Table 5 Training recipe for both Ouro 1.4B and 2.6B.

	Stage 1a Pre-train I	Stage 1b Pre-train II	Stage 2 CT Annealing	Stage 3 LongCT	Stage 4 Mid-training
Hyperparameters					
Learning rate (Final)	3.0×10^{-4}	3.0×10^{-4}	3.0×10^{-5}	3.0×10^{-5}	1.0×10^{-5}
LR scheduler	Constant	Constant	Cosine Decay	Constant	Cosine Decay
Weight decay			0.1		
Gradient norm clip			1.0		
Optimizer			AdamW ($\beta_1 = 0.9, \beta_2 = 0.95$)		
Batch size (tokens)	4M → 8M		8M		
Sequence length	4K	4K	16K	64K	32K
Training tokens	3T	3T	1.4T	20B	300B
Recurrent steps	8		4		
β for KL divergence	0.1		0.05		
RoPE base	10K	10K	40K	1M	1M
Data Focus					
Web data	High	High	Medium	Low	Low
Math & Code	Low	Low	High	Low	High
Long-context	None	None	Low	High	Medium
SFT-quality	None	None	Low	Low	High

prior, allowing the model greater freedom to explore beneficial depth patterns without being artificially constrained. This adjustment was crucial for maintaining stable training dynamics while enabling the model to learn effective depth allocation.

4.3.2 Stage-wise Training Details

We utilize the flame [46] framework for performing pretraining, built upon torchtitan [47]. To fully utilize our resources, we adopt an upcycling strategy that enables efficient scaling of model capacity during training.

- **Stage 1a: Pre-training Phase I (Exploration Phase).** We initially train the model on 3T tokens of web data from Nemotron-CC with 8 recurrent steps. The training uses the Warmup-Stable-Decay (WSD) learning rate scheduler with a peak learning rate of 3×10^{-4} . The sequence length is set to 4K tokens with an initial batch size of 4M tokens, gradually increased to 8M for stability. During this phase, we observed training instabilities that prompted our subsequent architectural adjustments.
- **Stage 1b: Pre-training Phase II with Stability-Driven Upcycling.** After identifying stability issues in Stage 1a, we implemented an architectural pivot: reducing recurrent steps from 8 to 4. To maintain computational efficiency while improving stability, we split our approach into two variants:
 - **Variant 1:** A 1.4B parameter model maintaining 24 layers with 4 recurrent steps
 - **Variant 2:** A 2.6B parameter model created through layer stacking (48 layers) with 4 recurrent steps

The recurrent nature of our architecture makes this upcycling process particularly smooth, as the shared weights across iterations naturally facilitate layer duplication without the typical instabilities seen in standard transformer upcycling. Both variants are trained on an additional 3T tokens with the stabilized configuration. The data composition is carefully balanced as shown in Table 3.

- **Stage 2: CT Annealing.** Building on the stable foundation from Stage 1b, we enhance the model with higher-quality data while annealing the learning rate to 3×10^{-5} . The training corpus comprises 1.4T tokens with increased emphasis on mathematical and coding capabilities. We extend the sequence length to 16K tokens, exceeding the length of most samples to minimize truncation and better utilize the enhanced data quality. The recurrent steps remain at 4, having proven optimal for the stability-performance trade-off. The data composition is carefully balanced as shown in Table 4.
- **Stage 3: LongCT.** This stage focuses on extending the model’s context window capabilities. We train on 20B tokens from the ProLong-64K dataset with sequences of 64K tokens, maintaining the batch size at 8M tokens. The reduced KL coefficient ($\beta = 0.05$) continues to provide stable training dynamics even with these extended sequences.
- **Stage 4: Mid-training.** The final stage leverages 90B tokens of extremely high-quality SFT data, consisting of both <Question, Answer> and <Question, CoT, Answer> samples. All SFT-style data is converted to ChatML format to facilitate subsequent post-training alignment. The learning rate is further reduced to 1×10^{-5} with a cosine scheduler to help the model better absorb on this diverse, high-quality dataset.

Optimization Configuration. Throughout all stages, we use AdamW optimizer with weight decay set to 0.1, $\beta_1 = 0.9$, $\beta_2 = 0.95$, and gradient clipping at 1.0. These conservative settings were chosen specifically to maintain stability with recurrent architectures.

Learning Rate Considerations. We empirically found that recurrent architectures require smaller learning rates compared to standard transformers of equivalent parameter count. While resource constraints prevented exhaustive hyperparameter search, our chosen rates represent conservative values that prioritized stable convergence over potentially faster but riskier optimization trajectories.

Sequence Length Progression. The sequence length is progressively increased across stages: 4K tokens for both pre-training phases, 16K for continual training with learning rate annealing, 64K for long-context training, and 32K for mid-training. This gradual progression helps maintain stability while enhancing the training throughput and expanding the model’s long-context capability.

4.4 Supervised Fine-Tuning

Data Composition. We perform supervised fine-tuning (SFT) on a diverse corpus of approximately 8.3M examples drawn from high-quality public datasets. As shown in Table 6, our training mixture emphasizes mathematical reasoning (3.5M examples) and code generation (3.2M examples), while also incorporating scientific reasoning (808K examples) and conversational abilities (767K examples).

For mathematical reasoning, we combine OpenThoughts3 [48] and AceReason-1.1-SFT [49] to provide comprehensive coverage of problem-solving strategies. Our code training data aggregates multiple sources including AceReason-1.1-SFT, OpenCodeReasoning [50], Llama-Nemotron-Post-Training-Dataset [51], and OpenThoughts3, ensuring broad exposure to diverse programming paradigms and reasoning patterns. Scientific reasoning capabilities are developed through OpenThoughts3 and Llama-Nemotron-Post-Training-Dataset, while conversational proficiency is enhanced using the OO1-Chat-747K² and DeepWriting-20K [52] datasets.

Training Configuration. We train for 2 epochs with a maximum sequence length of 32K tokens using the LlamaFactory codebase [53]. We employ the Adam optimizer with a learning rate of 2×10^{-5} and $\beta = (0.9, 0.95)$, applying a cosine decay schedule for stable convergence.³

Table 6 Supervised fine-tuning data composition. The training corpus comprises 8.3M examples across four key capability domains.

Topic	Data Source	Size
Math	OpenThoughts3, AceReason-1.1-SFT	3.5M
Code	AceReason-1.1-SFT, OpenCodeReasoning, Llama-Nemotron-Post-Training-Dataset, OpenThoughts3	3.2M
Science	OpenThoughts3, Llama-Nemotron-Post-Training-Dataset	808K
Chat	OO1-Chat-747K, DeepWriting-20K	767K

4.5 Reinforcement Learning Attempts

Following the SFT stage, we conducted exploratory RLVR (Reinforcement Learning with Verifiable Rewards) alignment experiments using algorithms such as DAPO [54] and GRPO [55] on the DAPO-17K dataset. However, these attempts did not yield significant performance gains over the final SFT checkpoint. We hypothesize two primary reasons for this. First, Model Saturation: the models are relatively small and had already undergone extensive SFT, which may have saturated their capabilities and left little room for further improvement via RL. Second, Infrastructure Challenges: the unique LoopLM architecture posed difficulties for our RL infrastructure. Specifically, our vLLM-based system could not efficiently perform rollouts with dynamic early exits and subsequently use that variable-depth information for the update step. This forced us to adopt a trade-off solution for training and inference.

We explored two main strategies based on these constraints:

1. **Fixed 4-Round RL:** We performed both rollouts and updates using a fixed 4 recurrent steps. In this setup, model performance increased normally but did not surpass the SFT checkpoint. Interestingly, we found that even when trained at a fixed 4-round depth, the model could still utilize fewer rounds at inference time, behaving as incentivized by the RL objective. The reason for this generalization is currently unknown.
2. **Adaptive RL:** We attempted to perform both rollouts and updates using the model’s native adaptive early exit mechanism. This approach failed to yield performance improvements, which we primarily attribute to the infrastructure challenges in handling the dynamic computational graphs.

²<https://huggingface.co/datasets/m-a-p/001-Chat-747K>

³Training was interrupted due to infrastructure issues; we resumed from the last saved checkpoint with a learning rate close to the original cosine decay schedule.

We will further explore RL alignment for this architecture as we continue to develop infrastructure that can fully support LoopLM’s dynamic computation.

5 Experiments

5.1 Base Model Evaluation

We conduct comprehensive evaluations of the Ouro base models trained on 7.7T tokens using the LoopLM architecture. The evaluation focuses on their performance across general knowledge, reasoning, mathematics, science, coding, and multilingual capabilities. All benchmarks are evaluated using `lm-eval-harness` [56] and `evalplus` [57] frameworks with settings detailed in Appendix. C.1.

For the base model baselines, we compare our Ouro models with leading open-source base models, including Qwen2.5 [2], Qwen3 [3], Gemma3 [4], Llama3.1 [5], and Llama3.2 [5] series base models. All models are evaluated using the same evaluation pipeline to ensure fair comparison.

Table 7 Comparison of 1.4B LoopLM model with 1-4B parameter baselines. The best score is **bolded**, and the second-best is underlined. LoopLM’s column is highlighted in gray.

	Gemma3 1B	Llama3.2 1.2B	Qwen2.5 1.5B	Qwen3 1.7B	Qwen2.5 3B	Llama3.2 3B	Qwen3 4B	Gemma3 4B	Ouro 1.4B R4
Architecture	Dense	Dense	Dense	Dense	Dense	Dense	Dense	Dense	LoopLM
# Params	1.0B	1.0B	1.5B	1.7B	3.0B	3.0B	4.0B	4.0B	1.4B
# Tokens	2T	9T	18T	36T	18T	9T	36T	4T	7.7T
<i>General Tasks</i>									
MMLU	39.85	45.46	60.99	62.46	65.62	59.69	<u>73.19</u>	58.37	67.35
MMLU-Pro	11.31	11.80	29.11	37.27	37.87	33.34	51.40	34.61	<u>48.62</u>
BBH	30.26	30.72	43.66	53.51	55.37	39.45	<u>70.95</u>	66.32	71.02
ARC-C	39.25	41.98	54.44	55.72	55.46	52.47	63.65	<u>60.92</u>	60.92
HellaSwag	56.12	59.35	67.73	67.09	74.54	73.09	75.66	<u>75.58</u>	74.29
Winogrande	58.72	62.75	66.77	66.30	70.17	69.14	<u>71.19</u>	71.07	72.30
<i>Math & Coding Tasks</i>									
GSM8K	2.05	7.05	60.73	70.28	<u>74.60</u>	67.20	72.86	68.69	78.92
MATH500	41.00	7.40	17.60	25.80	42.60	40.80	59.60	<u>68.60</u>	82.40
HumanEval	6.70	19.50	52.40	66.50	68.90	29.90	77.40	34.80	<u>74.40</u>
HumanEval+	5.50	17.40	46.30	59.80	62.20	26.20	70.70	29.30	<u>67.40</u>
MBPP	12.40	35.70	60.30	68.00	63.00	50.30	78.80	60.60	<u>73.00</u>
MBPP+	10.10	29.10	50.00	58.50	54.20	39.70	65.90	51.10	<u>62.70</u>

Summary of Evaluation Results Based on the overall evaluation results, we highlight key conclusions about our base models:

- (1) Our 1.4B parameter Ouro model (with 4 recurrent steps) achieves performance comparable to the 4B Qwen3-Base across most benchmarks. Notably, it matches or exceeds the 4B model on challenging reasoning tasks such as BBH (71.02 vs 70.95), GSM8K (78.92 vs 72.86) and MATH500 (82.40 vs 59.60)
- (2) The 2.6B parameter Ouro model outperforms dense models up to 8B parameters on reasoning-intensive benchmarks. It achieves 55.73 on MMLU-Pro, 80.46 on BBH and 90.85 on MATH500, surpassing the 8B Qwen3-Base (53.72, 77.65 and 62.30 respectively).
- (3) The recurrent architecture shows particular strength on tasks requiring multi-step reasoning and knowledge manipulation, with the most pronounced gains observed on MMLU-Pro, BBH, GSM8K and MATH500 benchmarks, validating our hypothesis that iterative computation enhances reasoning capabilities.

Table 8 Comparison of 2.6B LoopLM model with 3-12B parameter baselines. The best score is **bolded**, and the second-best is underlined. LoopLM’s column is highlighted in gray.

	Qwen2.5	Llama3.2	Qwen3	Gemma3	Qwen2.5	Llama3.1	Qwen3	Gemma3	Ouro
	3B	3B	4B	4B	7B	8B	8B	12B	2.6B R4
Architecture	Dense	Dense	Dense	Dense	Dense	Dense	Dense	Dense	LoopLM
# Total Params	3.0B	3.0B	4.0B	4.0B	7.0B	8.0B	8.0B	12.0B	2.6B
# Trained Tokens	18T	9T	36T	4T	18T	15T	36T	12T	7.7T
<i>General Tasks</i>									
MMLU	65.62	59.69	73.19	58.37	74.20	73.02	76.63	72.14	<u>74.60</u>
MMLU-Pro	37.87	33.34	51.40	34.61	43.55	43.24	<u>53.72</u>	49.21	55.73
BBH	55.37	39.45	71.14	66.32	53.72	71.56	<u>77.65</u>	78.41	80.46
ARC-C	55.46	52.47	63.65	60.75	63.65	60.75	66.10	72.44	<u>66.40</u>
HellaSwag	74.54	73.09	75.66	75.58	79.98	<u>81.97</u>	79.60	83.68	79.69
Winogrande	70.17	69.14	71.19	71.27	76.48	<u>77.11</u>	76.80	77.74	75.85
<i>Math & Coding Tasks</i>									
GSM8K	74.60	67.20	72.86	68.69	81.50	78.17	83.09	77.18	<u>81.58</u>
MATH500	42.60	40.80	59.60	68.60	61.20	52.90	62.30	<u>83.20</u>	90.85
HumanEval	68.90	29.90	77.70	34.80	79.30	38.40	84.80	46.30	<u>78.70</u>
HumanEval+	62.20	26.20	70.70	29.30	70.60	31.10	75.30	37.20	<u>70.70</u>
MBPP	63.00	50.30	78.80	60.60	73.80	62.40	<u>79.00</u>	73.50	80.40
MBPP+	54.20	39.70	65.90	51.10	63.50	51.60	67.90	<u>66.10</u>	<u>66.60</u>

Table 9 Performance comparison across different benchmarks. For AIME24 and AIME25, we report pass@1/pass@10 metrics. The best score is **bolded**, and the second-best is underlined.

Model	AIME24		AIME25		Olympiad bench	Beyond AIME	HLE	Super GPQA	GPQA
	pass@1	pass@10	pass@1	pass@10					
Ouro-1.4B-Thinking-R4	<u>65.0</u>	<u>83.3</u>	46.3	73.3	71.6	34.0	<u>5.21</u>	47.4	45.5
Ouro-2.6B-Thinking-R4	64.7	90.0	50.3	76.7	<u>76.4</u>	39.0	5.58	53.7	52.7
Qwen3-1.7B	32.0	55.6	22.0	33.3	56.4	15.0	4.13	35.9	34.0
Qwen3-4B	61.3	75.0	<u>51.3</u>	63.3	73.2	31.0	<u>5.21</u>	<u>51.9</u>	<u>54.5</u>
Qwen3-8B	73.0	<u>86.7</u>	66.7	81.3	<u>75.3</u>	<u>38.0</u>	2.22	48.0	59.1
Deepseek-Distill-Qwen-1.5B	29.6	66.7	23.0	43.33	56.44	9.0	4.2	26.5	33.2
Deepseek-Distill-Qwen-7B	57.3	83.3	36.0	73.3	72.0	30.0	5.14	46.6	51.0

5.2 Reasoning Model Evaluation

We evaluate the reasoning capabilities of our Ouro reasoning models (**Ouro-Thinking**) with 4 recurrent steps on challenging mathematical and scientific benchmarks that require multi-step problem solving and deep reasoning. The evaluation includes AIME 2024/2025 (American Invitational Mathematics Examination), OlympiadBench, GPQA, SuperGPQA, BeyondAIME, and HLE, representing some of the most challenging reasoning tasks in the field.

Benchmarks.

- **AIME 2024/2025** [58]. 30 questions per year from AIME I and II; integer answers 0–999.
- **OlympiadBench** [59]. Olympiad-level bilingual scientific problems; supports images for multimodal inputs.
- **GPQA** [60]. 448 graduate-level multiple-choice questions in biology, physics, and chemistry; search-resistant design.

- **SuperGPQA** [61]. GPQA scaled to about 285 graduate disciplines; curated to remain challenging.
- **BeyondAIME** [62]. Hard integer-answer math beyond AIME; emphasizes contamination resistance.
- **HLE** [63]. Multi-disciplinary closed-ended benchmark; expert-written with public splits and a private test set.

Models compared. We report results for Ouro-1.4B-Thinking and Ouro-2.6B-Thinking, which are LoopLM-based looped language models with iterative depth. As baselines we include Qwen3-1.7B, Qwen3-4B, Qwen3-8B, DeepSeek-Distill-Qwen-1.5B, and DeepSeek-Distill-Qwen-7B. We use size-matched baselines whenever available, otherwise we compare to the next larger widely used model.

Evaluation protocol. All systems are evaluated with a single in-house harness and identical prompting. We adopt an LLM-as-judge protocol across benchmarks with a fixed rubric and tie-breaking policy. Unless otherwise noted, decoding uses `temperature = 1.0` and `top_p = 0.7` for every model.

Evaluation results. Table 9 summarizes outcomes. Iterative reasoning in the LoopLM architecture provides consistent gains on these tasks. The 1.4B Ouro model with 4 recurrent steps reaches 71.55 on OlympiadBench (vs. 73.18 for Qwen3-4B) and 34.0 on BeyondAIME (vs. 31.0 for Qwen3-4B). The 2.6B with 4 recurrent steps variant scores 76.44 on OlympiadBench (vs. 75.25 for Qwen3-8B) and 39.0 on BeyondAIME (vs. 38.0 for Qwen3-8B).

5.3 Performance by Recurrent Depth and Extrapolation

Table 10 Performance of the Ouro 1.4B **base model** across different recurrent steps (C-QA is CommonsenseQA [64]). Steps 5-8 represent extrapolation, as the model was trained with a maximum of 4 steps. Performance peaks at the trained depth ($T = 4$) and then degrades.

UT Step	ARC-C (25-shot)	ARC-E (8-shot)	C-QA (10-shot)	HellaSwag (10-shot)	MMLU (5-shot avg)	Winogrande (5-shot)
1	37.63	63.85	44.64	55.24	41.21	56.99
2	54.86	80.30	67.98	71.15	60.43	66.69
3	59.47	83.33	74.37	74.07	66.71	71.35
4	60.92	83.96	75.43	74.29	67.45	72.30
<i>Extrapolation (Trained on T=4)</i>						
5	58.96	82.91	75.35	73.72	66.64	70.32
6	59.73	82.58	74.94	72.77	65.77	71.03
7	58.96	81.99	74.28	72.35	65.28	70.09
8	58.19	82.07	73.55	71.60	64.49	69.30

We analyze the Ouro model’s performance as a function of its recurrent computational depth. Our models were trained with a maximum of 4 recurrent steps ($T = 4$). We investigate this behavior for both our **base models** and our SFT **Ouro-Thinking** models.

Base Model Performance. Tables 10 and 11 present the performance of the Ouro 1.4B and 2.6B **base models**, respectively, evaluated at depths from $T = 1$ to $T = 8$.

For both base models, performance on standard benchmarks (e.g., MMLU, ARC-C) generally improves up to the trained depth of $T = 4$. Steps $T = 5$ through $T = 8$ represent extrapolation beyond the training configuration. As shown in both tables, benchmark performance sees a moderate degradation when extrapolating, with a noticeable drop compared to the peak at $T = 4$.

However, this degradation in task-specific performance contrasts sharply with the model’s safety alignment. As detailed in Section 7.1, the model’s safety improves as the number of recurrent steps increases, even into the

Table 11 Performance of the Ouro 2.6B **base model** across different recurrent steps (C-QA is CommonsenseQA [64]). Steps 5–8 represent extrapolation, as the model was trained with a maximum of 4 steps. Performance is strongest around the trained depth ($T = 4$) and shows varied degradation patterns during extrapolation.

UT Step	ARC-C (25-shot)	ARC-E (8-shot)	C-QA (10-shot)	HellaSwag (10-shot)	MMLU (5-shot avg)	Winogrande (5-shot)
1	47.95	72.39	57.58	68.94	51.55	61.48
2	62.37	85.23	76.90	77.61	67.63	70.48
3	65.36	87.33	79.77	79.12	73.57	74.35
4	66.38	86.95	81.65	79.56	74.60	75.53
<i>Extrapolation (Trained on $T=4$)</i>						
5	65.36	86.83	81.24	79.57	74.43	75.93
6	65.02	86.74	81.08	79.63	73.79	75.37
7	65.44	86.57	80.75	79.59	72.92	75.77
8	64.76	86.49	81.08	79.50	72.24	74.59

Table 12 Performance of Ouro-1.4B-Thinking model by recurrent step. The model was trained at $T = 4$. Performance peaks around $T = 4$ or $T = 5$. All scores are percentages (0-100).

Benchmark	T=1	T=2	T=3	T=4	T=5	T=6	T=7	T=8
OlympiadBench	2.22	59.70	70.67	71.55	72.30	69.48	69.04	66.81
SuperGPQA	2.03	33.07	44.50	47.37	48.73	46.15	45.29	42.88
AIME 2024	0.00	37.33	62.33	65.00	60.67	50.67	42.33	38.67
AIME 2025	0.33	25.00	43.33	46.30	47.00	43.00	41.00	38.00

extrapolated regime ($T > 4$). This suggests that while the model’s fine-grained knowledge for benchmarks may falter beyond its training depth, the iterative refinement process continues to enhance its safety alignment.

Reasoning Model (SFT) Performance. We conduct a similar analysis on our SFT models, **Ouro-Thinking**, to see how recurrent depth affects specialized reasoning tasks. Results for the 1.4B and 2.6B models are presented in Table 12 and Table 13, respectively.

We conduct a similar analysis on our SFT models, **Ouro-Thinking**, to see how recurrent depth affects specialized reasoning tasks. Results for the 1.4B and 2.6B models are presented in Table 12 and Table 13, respectively.

For both SFT models, performance at $T = 1$ is very low, confirming that iterative refinement is essential for these complex tasks. Performance generally peaks at or near the trained depth, but shows slightly different patterns. The 1.4B model (Table 12) peaks around $T = 4$ or $T = 5$. The 2.6B model (Table 13) tends to peak slightly earlier, at $T = 3$ or $T = 4$. Interestingly, neither model peaks strictly at $T = 4$ across all tasks, unlike the base model evaluations which are often logit-based. This may suggest that the longer decoding required for these reasoning tasks allows for a more active exploration of capabilities at different recurrent depths. For both models, performance degrades as they extrapolate to deeper, unseen recurrent steps ($T = 6 – 8$), reinforcing that performance is optimized for the depth seen during training.

Table 13 Performance of Ouro-2.6B-Thinking model by recurrent step. The model was trained at $T = 4$. Performance peaks at $T = 3$ or $T = 4$. All scores are percentages (0-100).

Benchmark	T=1	T=2	T=3	T=4	T=5	T=6	T=7	T=8
OlympiadBench	18.96	68.59	75.56	76.44	71.85	69.19	57.63	39.26
SuperGPQA	15.66	48.58	56.70	53.68	56.45	55.44	53.32	46.84
AIME 2024	3.00	52.00	70.33	64.70	57.00	56.33	49.67	39.00
AIME 2025	2.00	40.67	50.67	50.30	49.33	46.00	38.00	24.33

5.4 Early Exit and Adaptive Computation Efficiency

A defining advantage of the LoopLM architecture lies in its capacity for adaptive computation allocation. Unlike standard transformers with fixed computational budgets, our model can dynamically adjust the number of recurrent steps based on input complexity. This section investigates various strategies for implementing adaptive early exit, comparing their effectiveness in balancing computational efficiency with task performance.

5.4.1 Early Exit Strategies

We explore three distinct approaches to determining when the model should terminate its iterative computation and produce the final output.

Baseline: Static Exit. The simplest strategy forces the model to exit at a predetermined recurrent step, regardless of the input characteristics. While this approach provides predictable computational costs, it fails to leverage the model’s potential for adaptive resource allocation. We evaluate static exit at steps 1 through 4 to establish performance bounds and understand the relationship between computational depth and accuracy.

Hidden State Difference Threshold. This heuristic-based approach monitors the magnitude of representational changes between consecutive recurrent steps. At each step t , we compute $\Delta h_t = \|h_t - h_{t-1}\|_2$ and trigger early exit when $\Delta h_t < \epsilon$ for some threshold ϵ .

Learned Gating with Q-Exit Criterion. Our primary approach employs the learned exit gate described in Section 4, which produces step-wise halting probabilities λ_t based on the model’s current hidden states. During inference, we apply the Q-exit criterion: at each step t , we compute the cumulative distribution function $CDF(t) = \sum_{i=1}^t p(i|x)$ and exit when $CDF(t)$ exceeds the threshold $q \in [0, 1]$. The threshold q serves as a deployment-time hyperparameter that controls the compute-accuracy trade-off without requiring model retraining.

We evaluate this strategy under two training configurations. The *untrained* configuration uses the gate as trained during our standard pretraining pipeline with the entropy-regularized objective (uniform prior KL loss). This represents the gate’s behavior when jointly optimized with language modeling throughout Stages 1-4. The *trained* configuration additionally applies the specialized adaptive exit loss described in Section 3.4, which explicitly teaches the gate to base stopping decisions on observed task loss improvements.

Experimental Results. Figure 5 presents the accuracy-efficiency trade-off curves for all strategies on the MMLU benchmark. By varying the exit threshold (or static exit step for baseline), we obtain multiple operating points for each method, enabling direct comparison at equivalent computational budgets measured by average exit round.

Several key findings emerge from this analysis:

1. The Ponder gate with specialized adaptive exit training achieves the best accuracy at every computational budget, demonstrating that the loss improvement-based training signal described in Section 3.4 provides

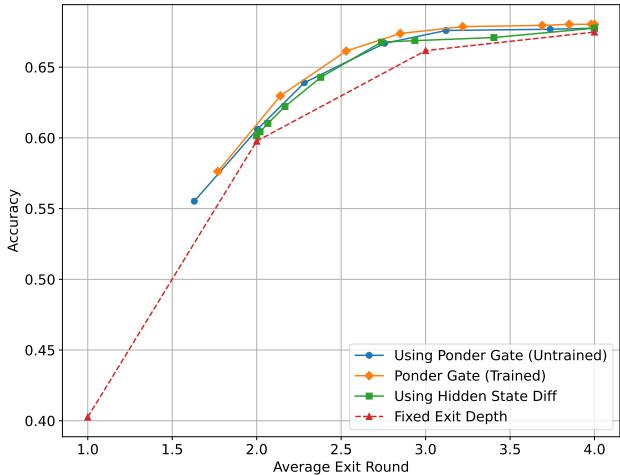


Figure 5 Comparison of early exit strategies on MMLU.

We evaluate four approaches across different average exit rounds: static baseline (red triangle), hidden state difference threshold (green squares), Ponder gate from standard pretraining (blue circles), and Ponder gate with specialized adaptive exit training from Section 3.4 (orange diamonds).

clear benefits over standard entropy regularization. At an average exit round of 2.5, the specialized training reaches 66% accuracy while the standard gate achieves approximately 64%;

2. Even without specialized training, the Ponder gate from standard pretraining substantially outperforms the static baseline, validating that the entropy-regularized objective with uniform prior successfully enables adaptive computation. The gate learns to differentiate input difficulty through the general training dynamics, though it lacks the explicit supervision to correlate stopping decisions with actual performance improvements. This demonstrates that our base training approach already captures useful signals for resource allocation;
3. The hidden state difference threshold strategy performs surprisingly competitively, closely tracking both gate configurations. At moderate computational budgets (2-3 average rounds), it achieves accuracy within 1%-2% of the specialized trained gate, suggesting that representation stability provides a reasonable proxy for computational convergence. However, the consistently superior performance of the specialized trained gate across all operating points confirms that explicit supervision via the adaptive exit loss captures information beyond what can be inferred from representational dynamics alone.
4. Comparing the untrained and trained gate configurations reveals the value proposition of the specialized training procedure. The gap between these curves, approximately 2%-3% accuracy at most operating points, represents the benefit of teaching the gate to explicitly monitor task loss improvements $I_t^{(n)}$ rather than relying solely on entropy regularization to discover stopping policies. This empirical result validates our design choice to introduce the adaptive exit loss as a specialized training objective.
5. The baseline’s monotonic improvement from 1 to 4 rounds confirms the “deeper is better” property while revealing diminishing returns. The dramatic jump from 1.0 to 2 rounds (40% to 60% accuracy) contrasts with the marginal gain from 3 to 4 rounds (67.35% accuracy). This pattern explains why adaptive methods prove effective: most examples achieve near-maximal performance at intermediate depths, with only a minority requiring full computational depth.

5.4.2 KV Cache Sharing for Inference Efficiency

The recurrent nature of our architecture introduces a challenge: naively, each recurrent step requires maintaining its own KV cache, leading to $4\times$ memory overhead for our 4-step model. We investigate strategies to reduce this overhead through KV cache reuse.

Prefilling Phase During the prefilling phase (processing the input prompt), we find that all four recurrent steps require their own KV caches, as each step transforms the representations in ways that cannot be approximated by earlier steps. Attempting to reuse KV caches during prefilling leads to performance degradation (>10 points on GSM8K).

Decoding Phase However, during the decoding phase (auto-regressive generation), we discover that KV cache reuse becomes viable. We explore two strategies:

1. **Last-step reuse:** Only maintain KV cache from the final (4th) recurrent step
2. **First-step reuse:** Only maintain KV cache from the first (1st) recurrent step.
3. **Averaged reuse:** Maintain an averaged KV cache across all four steps

As shown in Table 14, these strategies yield dramatically different outcomes. Reusing only the first step’s cache results in a catastrophic performance collapse (e.g., 18.73 on GSM8K, down from 78.92), indicating that the initial representations are insufficient for subsequent decoding steps. In contrast, both the last-step and averaged reuse strategies achieve nearly identical performance (within 0.3 points on GSM8K) to the full cache baseline, while successfully reducing memory requirements by $4\times$. The last-step strategy performs slightly better than the averaged approach on MATH-500, suggesting that the final recurrent step’s representations are most informative for subsequent token generation. This finding enables practical deployment of LoopLM models with memory footprints comparable to standard transformers of similar parameter count.

Table 14 KV cache sharing strategies during decoding. Both last-step and averaged strategies achieve minimal performance loss while reducing memory by 4 \times .

Strategy	GSM8K	MATH-500	Memory Reduction
Full (4 \times cache)	78.92	82.40	1.00 \times
First-step only	18.73	8.43	4.00 \times
Last-step only	78.85	80.40	4.00 \times
Averaged	78.73	78.52	4.00 \times

6 Understanding LoopLMs Superiority from a Parametric Knowledge Viewpoint

Why LoopLMs achieve far better performance when the parameter counts do not increase? Although potential enhanced reasoning capabilities were observed in [8], the source of the advantage remains unclear. Specifically, do LoopLMs perform better due to the models’ increased **knowledge capacity** with the same size of parameters? Or do they have a better capability in **extracting and composing the knowledge** encoded within the parameters? Toward understanding the improvement of the phenomenon, we explore what capabilities are exactly enhanced by simply looping more times. In this section, we perform experiments to test the model’s abilities to *memorize* factual knowledge in its parameters, and the capabilities of *manipulating and composing* existing knowledge encoded in the parameters based on a set of fully controllable synthetic tasks in [65–67].

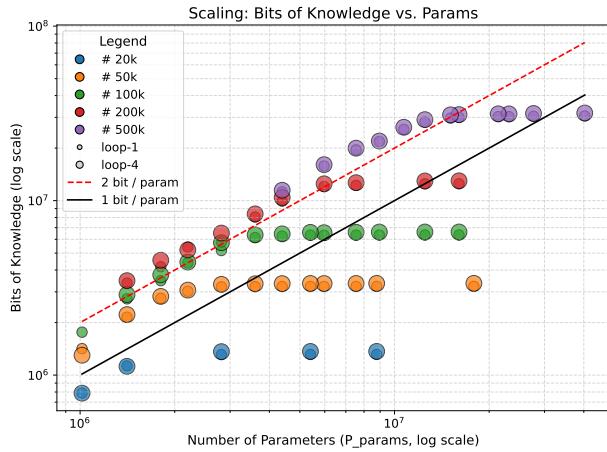
6.1 LoopLMs does not increase knowledge capacity

We first explore the *knowledge capacity*, i.e. the model’s storage capacity of facts in the parameters. We aim to answer the first question: do LoopLMs achieve better performance by **memorizing knowledge** when the parameter count is not increased?

Settings. Following the Capo task setting in Physics of language models [65, 66], we construct synthetic biographies to test **how much information the model memorizes**. Specifically, we generate several synthetic biographic datasets bioS(N) with different number of people N , and train a series of language models to memorize the information contained in the dataset. Each biography contains the individual’s name and five attributes a_1, a_2, \dots, a_5 of the person: gender, birth date, university, major, and employer. The names n and the attributes a_i are randomly selected from a pre-defined set \mathcal{N} and \mathcal{A}_i and combined together as a biography using a random template. Based on the random generation process, we have an information-theoretic lower bound for the model in the minimum bits required to encode all the names and attributes. To check whether the models memorize the biographic information accurately, we look at the probability of predicting the ground-truth attributes with the trained models given the biography context. Calculating the sum of cross-entropy loss on each attribute token positions, we can estimate how much information (estimated in bits) has already been memorized in the trained language model, which is our **knowledge capacity** metric.

With this metric, we can compare the knowledge capacity between the original model (with only one recurrent step) and the looped model (with 4 recurrent steps) with the same parameter count to investigate whether looping increases knowledge capacity. Moreover, as larger models should encode more information than smaller models, we also aim to investigate whether looped models have a better scaling effect when the size of the model grows. We thereby trained GPT-2 style models of different parameter numbers ranging from 1M to 40M (with depth and hidden dimension varied) and measured the number of bits of knowledge learned by each model. We trained on bioS(N) with N ranging from 20K to 500K individuals for 1000 exposures. More training details are provided in [Section B.1](#).

Results. The results are visualized in the plot “bits vs. # of parameters”, where we can observe the comparison between iso-parameter looped and non-looped models. Our results are shown in [Figure 6](#) (Left): **looping does not increase knowledge capacity nor improve capacity scaling**. Models with and without loops all attain around a similar capacity ratio ≈ 2 bits/parameter. Therefore, the number of parameters itself can be seen as a direct indicator of knowledge capacity, and **merely increasing looping does not help enhance knowledge capacity itself**.



	$L = 10$	$L = 16$	$L = 24$
Baseline model			
Base ($12 \otimes 1$)	93.6	94.4	34.8
2 layer model			
Base ($2 \otimes 1$)	21.5	8.4	7.5
Loop ($2 \otimes 6$)	98.1	96.3	78.0
3 layer model			
Base ($3 \otimes 1$)	75.4	29.8	11.0
Loop ($3 \otimes 4$)	97.9	95.8	92.2
6 layer model			
Base ($6 \otimes 1$)	84.7	59.5	20.0
Loop ($6 \otimes 2$)	93.4	88.5	35.1

Figure 6 Left. We trained both LoopLM and a standard transformer baseline with the same parameters on Capo task to compare the knowledge capacity gain by looping more times. With the same parameter count, the looped model and its non-looped baseline has almost the same knowledge capacity measured in bits of knowledge on Capo task. **Right.** Accuracy of looped/non-looped models on Mano task. Loop models are better than the iso-param ($\{2, 3, 6\} \otimes 1$) models. They also achieve better or comparable performance comparing to the iso-flop baseline ($12 \otimes 1$) model.

6.2 LoopLMS prevails in knowledge manipulation

We have already shown that reusing parameters cannot help the model memorize more atomic factual knowledge. However, natural language is not only about single-hop factual knowledge. In most of the scenarios, predicting the next token requires combining different piece of knowledge, which we called **knowledge manipulation** [65]. *Does looping and reusing parameters help LoopLMS in tasks that require flexible usage of knowledge?* We further consider two synthetic tasks to investigate the hypothesis on knowledge manipulation capacity: the synthetic Mano task in [66] based on modular arithmetic, and a multi-hop QA task in natural language [67] composing individual facts.

Mano Task. We first explore the knowledge manipulation task Mano in [66], based on a complex tree structure with restricted modular arithmetic knowledge. Models need to solve the task without intermediate thinking process. As illustration, an example could be $\langle \text{bos} \rangle + * \text{ a } \text{ b } \text{ c } \langle \text{eos} \rangle$ requires the model to directly output $(a * b) + c \bmod 23$. To solve this task, the model needs to (1) apply the arithmetic rules modulo 23 as the factual knowledge encoded in the parameters, and (2) parse the binary tree structure of the arithmetic to compose all calculations.

To evaluate the manipulation capability thoroughly, we consider the test accuracy across different difficulty levels based on maximum expression length L , which accounts for the number of operations in the sample. The model is trained with online samples with all possible expression lengths $\ell \in [1, L]$ and tested on the maximum expression length L . We prepare three levels of difficulties $L = [10, 16, 24]$ to test LoopLM’s superiority over non-looped models given fixed training budget. We train ($\{2, 3, 6, 12\} \otimes 1$) standard transformers as the baselines and several looped models ($k \otimes 12/k$) with $k = 2, 3, 6$. More details are included in Appendix B.2.

Results. The results in Figure 6 show that given the same parameters, looped models **always** outperform their non-looped counterpart for all possible $k \in \{2, 3, 6\}$. Even with the same number of FLOPs in the model, the looped models can often perform better. This indicates that **LoopLM has a better inductive bias towards knowledge manipulation**: with the same budget on training samples and computation, LoopLM can achieve comparable or even better performance after training when the task requires manipulation capability (e.g., parsing the arithmetic tree) given limited amount of required knowledge (e.g., modular arithmetic rules).

Multi-hop QA. Next, we corroborate our conjecture with a natural language multi-hop reasoning task proposed in [67], based on synthetic facts on relations \mathcal{R} between $|\mathcal{E}|$ different individuals, like *The instructor of A is B* and *The teacher of B is C*. The target is to answer multi-hop questions like ‘Who is the teacher of the instructor of A?’. We aim to study whether looping enables the original transformer better learn to perform

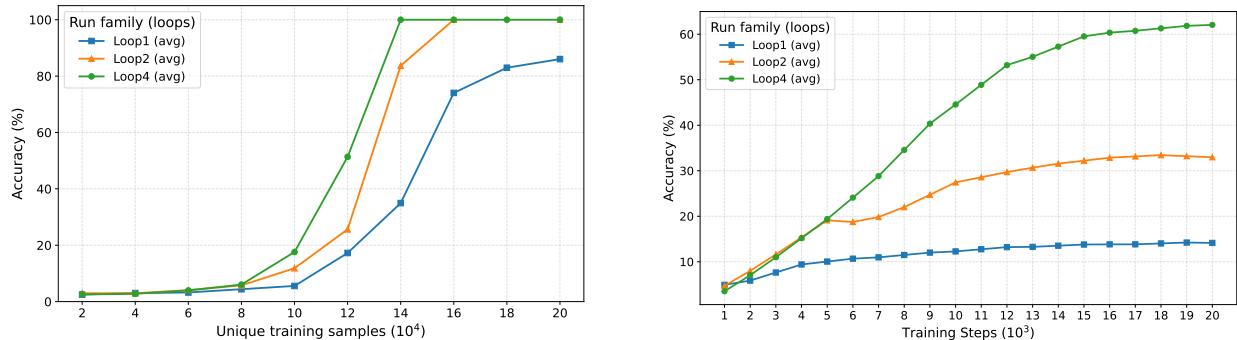


Figure 7 We trained LoopLMs and standard transformer baselines with the same parameters on Multi-hop QA tasks. To investigate the *sample efficiency* of LoopLMs, we **vary the number of unique training samples** (from 2.5% to 25% all possible QA pairs) for models with different loops. We compare the final performance using **the same compute budget in total training tokens**. **Left.** As shown, models with more loops requires fewer samples to learn the 3-hop QA task. **Right.** As an example, we train with 15% of all possible QA pairs (12000 unique samples) for 20000 steps with context length 1024 and batch size 2048. Models with more loops learn faster and achieve better performance comparing with models without loops.

internal multi-hop reasoning in a natural language setting. Compared to the Mano task, the task requires the model to memorize more factual knowledge with layer-wise data structure, which is closer to practical natural language multi-hop reasoning.

Multi-hop QA tasks require huge amount of samples to learn according to [67] when training standard transformers. To study whether LoopLMs accelerate the learning process of this multi-hop knowledge manipulation task, we consider *sample efficiency* in learning. Specifically, we study **how many different QA pairs are necessary for the trained model to achieve 100% accuracy, as well as the performance after training on a fixed budget of unique training samples**. For simplicity, we focus on the task with 3-hop QA pairs. We separate all possible QA pairs into training subsets of different sizes, and compare when each model perfectly generalizes on the leave-out test set. Similarly to the Mano task, we train a standard ($6 \otimes 1$) transformer as the baseline, and compare it with looped models ($6 \otimes \{2, 3, 4\}$) to study the effect of the universal transformer. We also train an iso-flop model ($24 \otimes 1$) for comparison. More details are included in Appendix B.3.

Results. The results in Figure 7 show that looped models generally learn the multi-hop QA task with fewer examples compared to both the non-looped iso-parameter model when the training budget is the same. Moreover, LoopLMs learn the multi-hop task much faster than the non-looped model with the same number of unique QA samples. The improved *sample efficiency* on the multi-hop reasoning task further demonstrates that LoopLM has a better ability to learn to compose and manipulate atomic factual knowledge.

Based on the results in both Mano and multi-hop QAs, we can conclude that LoopLMs have a better inductive bias towards more **flexible manipulation** of learned knowledge, instead of increasing the knowledge capacity. The conclusion holds for both synthetic tasks regardless of whether the task is more reasoning-heavy (Mano) or knowledge-heavy (Multi-hop QA). This also corresponds to the analysis (see Appendix B.4) on the existing benchmarks (e.g. MMLU): adding more recurrent steps significantly improves the performance on more reasoning-heavy categories, while the improvements on more knowledge-heavy tasks is limited.

6.3 Discussion: towards understanding why LoopLM helps knowledge manipulation

Why does LoopLM naturally bias towards better manipulation of the knowledge encoded in the parameter space? We conjecture that the reason lies in the inherent recurrent structure of LoopLM. Given that the knowledge capacity is limited by the parameter counts, looping enables LoopLM to better utilize the knowledge encoded in the parameters. LoopLM can reuse the knowledge in each looped block, **retrieve** new necessary factual information, or apply **structured procedures** to obtain the final prediction.

Search on the parametric knowledge graph. During pretraining, language models often obtain an enormous

amount of factual knowledge and learn analysis procedures with a rather shallow thinking depth. To perform more challenging tasks, the model needs to use multiple pieces of knowledge in the parameter space, which requires the model to search in-depth in the knowledge graph with directional dependencies formed by the atomic facts or knowledge. LoopLM naturally support an efficient reuse of the knowledge and algorithms stored in the parameter spaces: even though the knowledge piece is not retrieved or used in the previous calculations, the recurrent structure enables LoopLM to redo the procedure and extract necessary information.

Based on the abstraction above, we try to understand why LoopLMs are able to search on knowledge graph without adding more parameters. Specifically, we study the expressivity of LoopLM on a synthetic task. We consider the extensively studied search problem in the literature of **latent reasoning** [24, 68, 69]: *graph reachability* on a knowledge graph. Here, we consider that only part of the knowledge graph G_{ctx} is included in the context, and most of the knowledge relations G must be encoded in the parameters. The model must learn to compose the context knowledge G_{ctx} and the learned knowledge G . Compared to traditional CoT and recent proposed latent CoT [24, 68], we show that LoopLM is a parallelizable latent reasoning paradigm that requires fewer sequential reasoning steps.

Theorem 1 (Informal). *Fix n as the maximum size of the combined knowledge graph G . Given the adjacency matrix of the context graph G_{ctx} and a query pair (s, t) , there exists a one-layer transformer independent of G_{ctx} with loops $O(\log_2 D)$ times that checks whether there exists a path from s to t in the combined knowledge graph $(G + G_{ctx})$, where D is the diameter of $(G + G_{ctx})$.*

Latent reasoning method		Discrete CoT	Continuous CoT	Universal Transformer
Sequential computation steps		$O(n^2)$	$O(D)$	$O(\log D)$

The proof and the discussion on LoopLM’s efficiency are deferred to Appendix B.5. We claim that the universal transformers maximize the parallelism in exploring all-pair connectivity and reduce the sequential computation steps exponentially from $O(n^2)$ to $O(\log D)$, making the latent reasoning much more efficient than the traditional CoT view of looping [8] and continuous CoT [68]. The potential efficient latent reasoning ability may account for the superiority of LoopLM in knowledge manipulation, which also may contribute to the superior performance in reasoning-heavy tasks.

Recurrence improves sample efficiency. The expressiveness result of LoopLM does not explain why the transformers with loops often learns knowledge manipulation tasks with samples much fewer than its iso-FLOP counterpart. We conjecture that the reason lies again in the **recurrent structure** of LoopLM. Assuming the reasoning tasks require multiple manipulation and recursion using learned parametric knowledge or algorithmic procedure, the models have to learn a *repeated structure* across layers of different depth. For deep transformer models without looping, they potentially have to explore a large function class where each block of parameters are not tied. The parameter-sharing layers may help the model explore a much smaller realizable hypothesis class, thus reducing the sample complexity of learning those manipulation tasks. It could be a possible statistical reason that LoopLM enjoys a better sample complexity on those reasoning/manipulation tasks.

7 Safety, Faithfulness and Consistency

7.1 Safety

We assess model safety using HEx-PHI dataset [17], which contains 330 examples covering 11 prohibited categories. HEx-PHI employs GPT-4o as a judge to assign each model response a harmfulness score from 1 to 5; a higher score indicates a less safe output. Additionally, we compute the harmfulness rate, defined as the proportion of the test cases that receive the highest harmfulness score of 5. For Ouro Base models, we use greedy decoding with `max_new_tokens=128`; For Ouro Thinking models, we sample with `temperature=1.0`, `top_p=0.7` with `max_new_tokens=8192`. We evaluate Ouro 1.4B and 2.6B models with recurrent steps ranging from 1 to 8, and report the result in Figure 8a. Notably, while our models were trained with only 4 recurrent steps, both models show their **extrapolation capability** by extending recurrence steps to 5-8 during inference. This demonstrates the model’s ability to generalize to deeper computation than seen during training. The Ouro Thinking checkpoints further enhance safety alignment, reducing harmful rates to 0.009 for Ouro

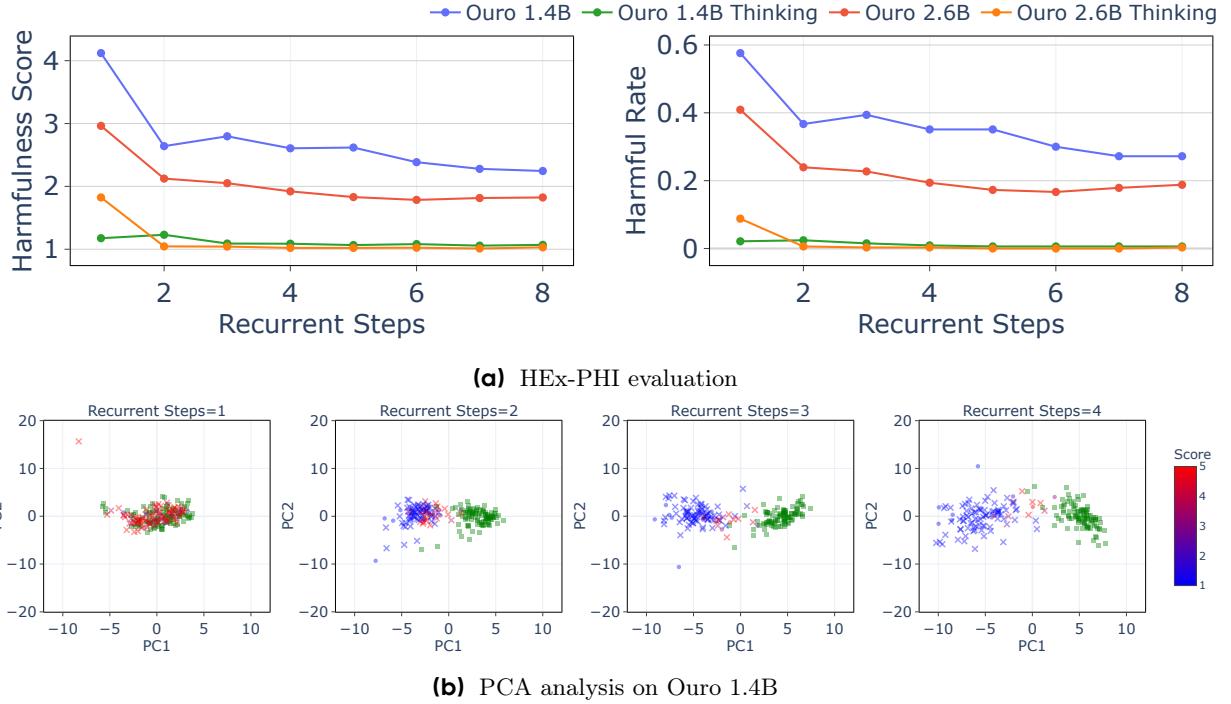


Figure 8 (a) For both 1.4B and 2.6B models, Ouro demonstrates improved safety alignment on HEx-PHI as the recurrent steps increase. Note that models were trained with 4 recurrent steps; evaluations at steps 5-8 demonstrate successful extrapolation beyond the training configuration. (b) **As the recurrent steps increase, Ouro 1.4B can better distinguish the benign prompts and harmful prompts, leading to safer responses.** We perform PCA on the hidden representation of the last input token from the model’s top layer. Harmful prompts with a harmfulness score of 4 or 5 at recurrent step 1 are marked with \times , while other harmful prompts are shown as circles. The color of each point reflects the harmfulness score of the corresponding response. Benign prompts are shown as green squares.

1.4B Thinking and 0.003 for Ouro 2.6B Thinking at 4 recurrent steps, comparable to Qwen3-4B-Thinking (0.009).

To further investigate how increasing recurrent steps affects the model’s safety alignment, we conduct Principal Component Analysis (PCA) on the hidden representation of the last input token from the top model layer. For a controlled analysis, we select 100 benign and 100 harmful questions with identical formats (all the examples are the questions starting with ‘How to’) from Zheng et al.(2024) [70]⁴. Additionally, we evaluate the model’s responses to the 100 harmful questions and compute a 5-level harmfulness score (same as the one used in HEx-PHI) for each response. We plot our PCA analysis on Ouro 1.4B in Figure 8b, from which we have the following observations. First, as the number of recurrent steps increases, the model becomes more capable of separating benign and harmful prompts, resulting in safer responses, as indicated by the decreasing number of red points. Furthermore, most points associated with unsafe responses appear near the middle of the plot, which represents the boundary between the ‘benign’ and ‘harmful’ clusters. This suggests that difficulty in distinguishing harmfulness may lead to unsafe responses, which can be alleviated by increasing the number of recurrent steps.

7.2 Faithfulness

We call a model’s thinking process faithful if it is (i) procedurally correct and (ii) causally coupled to the final answer. Concretely, a faithful process should satisfy a counterfactual criterion: if the justification is intervened on (e.g., altered to a different intermediate state), the final prediction should change accordingly. A growing

⁴Harmful questions: <https://github.com/chujiezheng/LLM-Safeguard/blob/main/code/data/custom.txt>; Benign questions: https://github.com/chujiezheng/LLM-Safeguard/blob/main/code/data_harmless/custom.txt

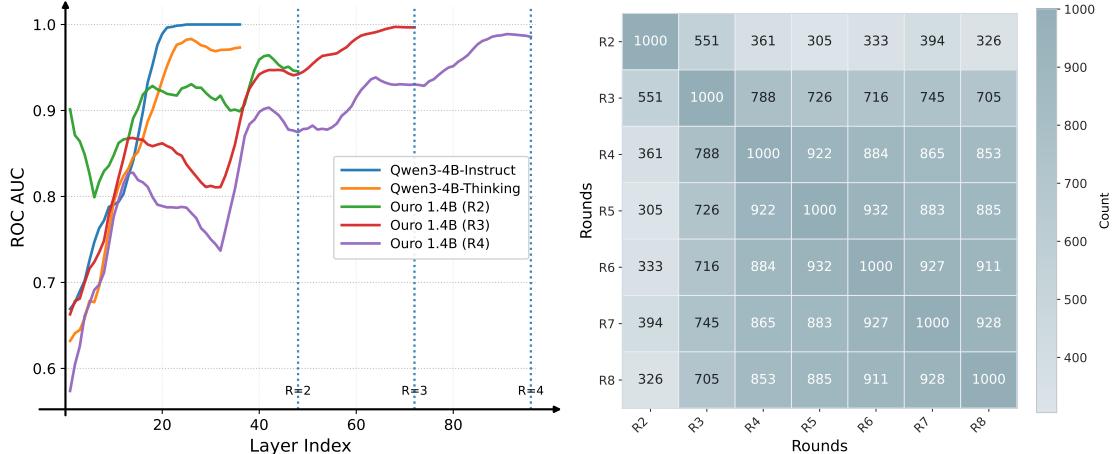


Figure 9 Left. ROCAUC of linear probes by layer on Quora Question Pairs. Each colored curve shows a probe trained on hidden states within a given 2 to 8 recurrent steps to predict that loop’s answer; Qwen3-4B models are the baselines. Vertical dotted lines mark loop boundaries. In recurrent step $i = 2, 3, 4$, the ROC AUC rises quickly within a recurrent step, then partially resets at the next loop, indicating that intra-step answers are determined early while cross-step updates modify the provisional answer. **Right. Agreement across recurrent steps.** Heat map (A) over 1,000 Quora Question Pairs. Entry $A[i, j]$ is the number of items for which steps (i) and (j) assign the same label.

body of work [71–74] shows that standard LLMs often appear to decide on an answer before generating chain-of-thought text and then use that text to rationalize the already-formed decision.

In LoopLM, the reasoning substrate is the sequence of latent states $H^{(1)} \rightarrow H^{(2)} \rightarrow \dots \rightarrow H^{(T)}$. Each transition $H^{(k)} \rightarrow H^{(k+1)}$ performs non-trivial computation using the same shared-weight block, and each step is trained to improve the task objective. Thus, the causal path to the answer is this latent trajectory, not any optional natural-language trace. When we decode intermediate text $\text{Text}(R_k)$ from $H^{(k)}$ via the LM head, we treat it as an instrumented readout of the internal state rather than the mechanism itself. Because $H^{(k)}$ is directly supervised by the LM loss, its projection into token space provides a faithful snapshot of what the model currently represents.

Standard evaluation of faithfulness is often based on the manipulation of the reasoning process, CoT, and check if the average treatment effect of CoT is significant. In our case, we cannot manipulate the latent reasoning process. Instead, we adopt an observational proxy for mediation: we read out intermediate hidden representations and test whether predictions change as recurrence deepens on inputs that admit multiple plausible labels. Concretely, we assess whether intermediate “thinking” genuinely mediates decisions by measuring step-by-step predictability and agreement patterns. We use the Quora Question Pairs dataset [75], which asks whether two short questions are semantically equivalent: a setting with ambiguity and weakly-defined decision boundaries. There are a lot of ambiguous questions in this dataset:

Example: Ambiguous questions in Quora dataset

Question: does the following two questions have the same intent?

Pair 1:

1. What are the questions should not ask on Quora?
2. Which question should I ask on Quora?

Answer: False

Pair 2:

1. How do we prepare for Union Public Service Commission?
2. How do I prepare for civil service?

Answer: True

If a thinking process merely rationalizes the pre-committed answer, even if the questions are very ambiguous, the answers will not change after the reasoning process. This has been reported for Gemma-2 9B and reproduced by us on Qwen-3-4B-Thinking. As shown in the left part of Figure 9, the simple linear probe on the final-token logits on the Qwen3-4B-Thinking model shows 0.99 ROC AUC predicting the model’s eventual

answer, which means the thinking process almost does not affect the results.

In our model, the situation is very different. Our $1.4B \times 4$ model uses 24 layers per recurrent step. We train linear probes on hidden states from layers 1 through $24i$ to predict the step- i answer, for $i \in \{2, 3, 4\}$. Within a single recurrent step, the step- i answer is well predicted by a probe on representation within layer $24i$, indicating strong intra-step alignment between state and decision, which is similar to the non-reasoning model Qwen-4B-Instruct, showing in left part of Figure 9. Crucially, probes on the preceding representation (layer $24(i-1)$) do not reliably predict the step- i decision for $i \in \{2, 3, 4\}$, showing that the new recurrent pass performs additional computation that can revise a provisional choice.

To further examine the consistency between the results of different rounds. We also compute a step-by-step agreement matrix A over 1,000 Quora Question Pairs, where $A[i, j]$ counts identical labels between step i and step j (diagonal = 1000 by construction). See the right side of Figure 9. Adjacent steps never reach full agreement; for example, $A[3, 4] = 361$ indicates only 36.1% of step-3 answers match step-4. $A[2, 3] = 551$ indicates only 44.9% of step-2 answers match step-3. We also notice that when $i \geq 4$, the overlap consistency between step- i and step- $i+1$, $A[i, i+1]$, is close to 1000. We think this phenomenon comes from: (1) the model does not learn to reason recursively when $i > 4$. The model is trained within 4 loops; (2) as the number of loops increases, the answer gradually converges to a fixed point.

All in all, this systematic disagreement across steps when $i \leq 4$ is precisely what a faithful latent process should exhibit: the model is updating its decision as recurrence deepens, and intermediate predictions are not frozen rationalizations of the final output.

7.3 More Discussion

The practical barrier for safety-critical deployment is that a model’s articulated reasoning and its final answer may diverge. The LoopLM architecture reduces this gap by exposing a sequence of intermediate predictors that are strongly aligned with the final predictor and can be used both for acceleration and for pre-emptive control. We summarize three deployment advantages.

Built-in draft model for speculative decoding. Let $\text{Text}(R_t)$ denote the language-model head attached to the latent state after recurrent step t , and let T be the maximum step used at deployment. The pair

$$\left(\underbrace{\text{Text}(R_s)}_{\text{proposal}}, \underbrace{\text{Text}(R_T)}_{\text{verifier}} \right), \quad 1 \leq s < T.$$

forms a native proposal–verification decomposition for speculative decoding without training an external draft model. Proposals are sampled from $\text{Text}(R_s)$ and verified under $\text{Text}(R_T)$ using standard acceptance tests; rejected tokens are rolled back as usual. Because both heads share the same parameters up to step s , cached activations and KV states can be reused, reducing verifier overhead. This turns the recurrent structure into an architectural primitive for draft–verify decoding rather than an add-on.

Joint acceleration and pre-emptive safety. Using the same proposal–verification split, safety checks can be interleaved with speculative decoding without extra models. At step s :

1. Generate draft tokens with $\text{Text}(R_t)$ and compute their acceptance under $\text{Text}(R_T)$.
2. Run safety screening on the draft distribution or sampled drafts before any token is surfaced to the user. Screening can operate on logits, beams, or short candidate spans.
3. If a violation is detected, halt or reroute the response before streaming; otherwise, accept tokens that pass both verification and safety checks.

Because $\text{Text}(R_s)$ and $\text{Text}(R_T)$ share the latent trajectory, intermediate predictions are well-aligned with the final answer distribution. This alignment makes the step- s output a reliable proxy for the step- T output for the purpose of early screening, while the verifier maintains final quality. The Q-exit threshold q further provides a single deployment knob that simultaneously adjusts compute, consistency, and safety strictness by shifting the average exit depth.

Anytime generation with monotone refinement. The training objective in Section 3.4 optimizes the expected task loss across steps while preserving the deeper-is-better property. Consequently, for next-token prediction loss,

$$\mathbb{E}[\mathcal{L}^{(t+1)}] \leq \mathbb{E}[\mathcal{L}^{(t)}], \quad 1 \leq t < T,$$

so each additional loop refines the distribution toward higher-quality predictions. This yields an anytime algorithm: decoding may begin from any intermediate step s and continue streaming while later steps continue to verify or revise. Unlike chain-of-thought pipelines, which often require completing a reasoning prefix before emitting answers, LoopLM exposes a single predictive interface at every step, enabling immediate fallback to a smaller compute budget when latency constraints apply.

8 Conclusion

In this work, we introduced Ouro, a family of Looped Language Models that demonstrate exceptional parameter efficiency by integrating iterative computation and adaptive depth directly into pre-training on 7.7T tokens. Our 1.4B and 2.6B models consistently match or exceed the performance of 4B and 8B standard transformers, showcasing a 2-3× efficiency gain. We demonstrated this advantage stems not from increased knowledge storage, but from a fundamentally superior capability for knowledge manipulation, supported by synthetic experiments and theoretical analysis. We also presented a practical training objective using entropy regularization with a uniform prior to learn adaptive depth, and validated efficient KV cache sharing strategies that make LoopLMs viable for real-world deployment.

Beyond performance, the LoopLM architecture exhibits unique properties: its iterative refinement process provides a causally faithful reasoning trace, mitigating the post-hoc rationalization issues seen in standard CoT, and its safety alignment uniquely improves with increased recurrent steps, even when extrapolating. This work establishes iterative latent computation as a critical third scaling axis beyond parameters and data. Future research should focus on enhancing performance extrapolation at greater depths and exploring more complex recurrent mechanisms, solidifying this parameter-efficient approach as a necessary direction in a data-constrained era.

Acknowledgement

We sincerely thank Zeyuan Allen-Zhu for his in-depth discussion on the physics of language model part and his enlightening insights on knowledge manipulation. We also thank Yonghui Wu, Guang Shi, Shu Zhong, Tenglong Ao, Chen Chen, Songlin Yang, Wenhao Chai, and Yuhong Chou for their insightful discussions. Special thanks to Wenjia Zhu – his words opened our eyes to what the real problems are in current models, and inspired us to explore this direction.

Contributions

Project Lead

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Ge Zhang

Core Contributors

Rui-Jie Zhu: Proposes the project and leads the pre-training of Ouro. Optimizes pre-training and inference infrastructure, develops the initial vLLM implementation, and explores RLVR.

Zixuan Wang: Leads the analysis on understanding LoopLM superiority and is responsible for related experiments. He contributes to the design of adaptive early exit strategies, training, and the safety analysis.

Kai Hua: Designs and curates all pre-training data mixtures and provides key insights during the pre-training process.

Ge Zhang: Co-leads and supervises the Ouro. Provides several key insights during the pre-training and post-training process.

Tianyu Zhang: Leads the analysis of Ouro on consistency, safety, and faithfulness. He designs the pipeline evaluation on faithfulness. He contributes to post-training, probing and efficient KV cache design.

Ziniu Li: Leads the post-training phase, developing supervised fine-tuning and providing key contributions to RLVR exploration.

Haoran Que: Leads the scaling law analysis for LoopLM, investigating the relationship between performance, model size, and recurrent depth.

Boyi Wei: Contributes to the safety analysis, conducting evaluations on the HEx-PHI benchmark and performing PCA on model representations.

Zixin Wen: Contributes to the theoretical analysis, Physics of LLMs experiments, paper writing, and RLVR.

Fan Yin: Optimizes the vLLM and SGLang implementations for Ouro, contributing core pull requests to improve inference efficiency.

He Xing: Contributes to the vLLM infrastructure development and optimization.

Contributors

Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Xun Zhou, Qiyang Min, Hongzhi Huang, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai

Supervision

Ge Zhang, Wenhao Huang, Yoshua Bengio, Jason Eshraghian

References

- [1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [2] Qwen Team et al. Qwen2 technical report. [arXiv preprint arXiv:2407.10671](https://arxiv.org/abs/2407.10671), 2:3, 2024.
- [3] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengan Huang, Chenxu Lv, et al. Qwen3 technical report. [arXiv preprint arXiv:2505.09388](https://arxiv.org/abs/2505.09388), 2025.
- [4] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. [arXiv preprint arXiv:2503.19786](https://arxiv.org/abs/2503.19786), 2025.

- [5] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. [arXiv e-prints](#), pages arXiv–2407, 2024.
- [6] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. [Advances in neural information processing systems](#), 35:24824–24837, 2022.
- [7] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. [arXiv preprint arXiv:1807.03819](#), 2018.
- [8] Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. [arXiv preprint arXiv:2502.17416](#), 2025.
- [9] Khashayar Gatmiry, Nikunj Saunshi, Sashank J Reddi, Stefanie Jegelka, and Sanjiv Kumar. Can looped transformers learn to implement multi-step gradient descent for in-context learning? [arXiv preprint arXiv:2410.08292](#), 2024.
- [10] Khashayar Gatmiry, Nikunj Saunshi, Sashank J Reddi, Stefanie Jegelka, and Sanjiv Kumar. On the role of depth and looping for in-context learning with task diversity. [arXiv preprint arXiv:2410.21698](#), 2024.
- [11] Jianhao Huang, Zixuan Wang, and Jason D Lee. Transformers learn to implement multi-step gradient descent with chain of thought. [arXiv preprint arXiv:2502.21212](#), 2025.
- [12] William Merrill and Ashish Sabharwal. A little depth goes a long way: The expressive power of log-depth transformers. [arXiv preprint arXiv:2503.03961](#), 2025.
- [13] William Merrill and Ashish Sabharwal. Exact expressive power of transformers with padding. [arXiv preprint arXiv:2505.18948](#), 2025.
- [14] Sangmin Bae, Adam Fisch, Hravir Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise lora. [arXiv preprint arXiv:2410.20672](#), 2024.
- [15] Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. [arXiv preprint arXiv:2502.05171](#), 2025.
- [16] Boyi Zeng, Shixiang Song, Siyuan Huang, Yixuan Wang, He Li, Ziwei He, Xinbing Wang, Zhiyu Li, and Zhouhan Lin. Pretraining language models to ponder in continuous space. [arXiv preprint arXiv:2505.20674](#), 2025.
- [17] Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. Fine-tuning aligned language models compromises safety, even when users do not intend to! In [The Twelfth International Conference on Learning Representations](#).
- [18] Yilong Chen, Junyuan Shang, Zhenyu Zhang, Yanxi Xie, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. Inner thinking transformer: Leveraging dynamic depth scaling to foster adaptive internal thinking. [arXiv preprint arXiv:2502.13842](#), 2025.
- [19] Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hravir Harutyunyan, Ziwei Ji, Aaron Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. [arXiv preprint arXiv:2507.10524](#), 2025.
- [20] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. [arXiv preprint arXiv:1909.11942](#), 2019.
- [21] Raj Dabre and Atsushi Fujita. Recurrent stacking of layers for compact neural machine translation models. In [Proceedings of the AAAI Conference on Artificial Intelligence](#), volume 33, pages 6292–6299, 2019.
- [22] Sho Takase and Shun Kiyono. Lessons on parameter sharing across layers in transformers. [arXiv preprint arXiv:2104.06022](#), 2021.
- [23] Boxun Li, Yadong Li, Zhiyuan Li, Congyi Liu, Weilin Liu, Guowei Niu, Zheyue Tan, Haiyang Xu, Zhuyu Yao, Tao Yuan, et al. Megrez2 technical report. [arXiv preprint arXiv:2507.17728](#), 2025.
- [24] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. [arXiv preprint arXiv:2412.06769](#), 2024.

- [25] Amirkeivan Mohtashami, Matteo Pagliardini, and Martin Jaggi. Cotformer: More tokens with attention make up for less depth. In *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ NeurIPS 2023)*, 2023.
- [26] Bohong Wu, Shen Yan, Sijun Zhang, Jianqiao Lu, Yutao Zeng, Ya Wang, and Xun Zhou. Efficient pretraining length scaling. *arXiv preprint arXiv:2504.14992*, 2025.
- [27] Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of language models: Part 2.1, grade-school math and the hidden reasoning process. *arXiv preprint arXiv:2407.20311*, 2024.
- [28] Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, et al. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939*, 2025.
- [29] Nikita Balagansky and Daniil Gavrilov. Palbert: Teaching albert to ponder. *Advances in Neural Information Processing Systems*, 35:14002–14012, 2022.
- [30] Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.
- [31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [32] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.
- [33] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.
- [34] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, et al. Smollm2: When smol goes big-data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025.
- [35] Kaiyue Wen, Zhiyuan Li, Jason Wang, David Hall, Percy Liang, and Tengyu Ma. Understanding warmup-stable-decay learning rates: A river valley loss landscape perspective. *arXiv preprint arXiv:2410.05192*, 2024.
- [36] Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin A Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.
- [37] Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, Hritik Bansal, Etaash Guha, Sedrick Scott Keh, Kushal Arora, et al. Datacomp-lm: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 37:14200–14282, 2024.
- [38] Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset. *arXiv preprint arXiv:2412.02595*, 2024.
- [39] Yudong Wang, Zixuan Fu, Jie Cai, Peijun Tang, Hongya Lyu, Yewei Fang, Zhi Zheng, Jie Zhou, Guoyang Zeng, Chaojun Xiao, et al. Ultra-fineweb: Efficient data filtering and verification for high-quality llm training data. *arXiv preprint arXiv:2505.05427*, 2025.
- [40] Xinrun Du, Zhouliang Yu, Songyang Gao, Ding Pan, Yuyang Cheng, Ziyang Ma, Ruibin Yuan, Xingwei Qu, Jiaheng Liu, Tianyu Zheng, et al. Chinese tiny llm: Pretraining a chinese-centric large language model. *arXiv preprint arXiv:2404.04167*, 2024.
- [41] Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, J. H. Liu, Chenchen Zhang, Linzheng Chai, Ruijing Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. Opencoder: The open cookbook for top-tier code large language models. 2024.
- [42] Fan Zhou, Zengzhi Wang, Nikhil Ranjan, Zhoujun Cheng, Liping Tang, Guowei He, Zhengzhong Liu, and Eric P. Xing. Megamath: Pushing the limits of open math corpora. *arXiv preprint arXiv:2504.02807*, 2025. Preprint.
- [43] Rabeeh Karimi Mahabadi, Sanjeev Satheesh, Shrimai Prabhumoye, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-cc-math: A 133 billion-token-scale high quality math pretraining dataset. 2025.
- [44] NVIDIA, Aarti Basant, Abhijit Khairnar, Abhijit Paithankar, Abhinav Khattar, Adithya Renduchintala, Aditya Malte, Akhiad Bercovich, Akshay Hazare, Alejandra Rico, Aleksander Ficek, Alex Kondratenko, Alex Shaposhnikov,

Alexander Bukharin, Ali Taghibakhshi, Amelia Barton, Ameya Sunil Mahabaleshwarkar, Amy Shen, Andrew Tao, Ann Guan, Anna Shors, Anubhav Mandarwal, Arham Mehta, Arun Venkatesan, Ashton Sharabiani, Ashwath Aithal, Ashwin Poojary, Ayush Dattagupta, Balaram Buddharaaju, Banghua Zhu, Barnaby Simkin, Bilal Kartal, Bita Darvish Rouhani, Bobby Chen, Boris Ginsburg, Brandon Norick, Brian Yu, Bryan Catanzaro, Charles Wang, Charlie Truong, Chetan Mungekar, Chintan Patel, Chris Alexiuk, Christian Munley, Christopher Parisien, Dan Su, Daniel Afrimi, Daniel Korzekwa, Daniel Rohrer, Daria Gitman, David Mosallanezhad, Deepak Narayanan, Dima Rekesh, Dina Yared, Dmytro Pykhtar, Dong Ahn, Duncan Riach, Eileen Long, Elliott Ning, Eric Chung, Erick Galinkin, Evelina Bakhturina, Gargi Prasad, Gerald Shen, Haifeng Qian, Haim Elisha, Harsh Sharma, Hayley Ross, Helen Ngo, Herman Sahota, Hexin Wang, Hoo Chang Shin, Hua Huang, Iain Cunningham, Igor Gitman, Ivan Moshkov, Jaehun Jung, Jan Kautz, Jane Polak Scowcroft, Jared Casper, Jian Zhang, Jiaqi Zeng, Jimmy Zhang, Jinze Xue, Jocelyn Huang, Joey Conway, John Kamalu, Jonathan Cohen, Joseph Jennings, Julien Veron Vialard, Junkeun Yi, Jupinder Parmar, Kari Briski, Katherine Cheung, Katherine Luna, Keith Wyss, Keshav Santhanam, Kezhi Kong, Krzysztof Pawelec, Kumar Anik, Kunlun Li, Kushan Ahmadian, Lawrence McAfee, Laya Sleiman, Leon Derczynski, Luis Vega, Maer Rodrigues de Melo, Makesh Narsimhan Sreedhar, Marcin Chochowski, Mark Cai, Markus Kliegl, Marta Stepniewska-Dziubinska, Matvei Novikov, Mehrzad Samadi, Meredith Price, Meriem Boubdir, Michael Boone, Michael Evans, Michal Bien, Michal Zawalski, Miguel Martinez, Mike Chrzanowski, Mohammad Shoeybi, Mostafa Patwary, Nimit Dhameja, Nave Assaf, Negar Habibi, Nidhi Bhatia, Nikki Pope, Nima Tajbakhsh, Nirmal Kumar Juluru, Oleg Rybakov, Oleksii Hrinchuk, Oleksii Kuchaiev, Oluwatobi Olabiyi, Pablo Ribalta, Padmavathy Subramanian, Parth Chadha, Pavlo Molchanov, Peter Dykas, Peter Jin, Piotr Bialecki, Piotr Januszewski, Pradeep Thalasta, Prashant Gaikwad, Prasoon Varshney, Pritam Gundecha, Przemek Tredak, Rabeeh Karimi Mahabadi, Rajen Patel, Ran El-Yaniv, Ranjit Rajan, Ria Cheruvu, Rima Shahbazyan, Ritika Borkar, Ritu Gala, Roger Waleffe, Ruoxi Zhang, Russell J. Hewett, Ryan Prenger, Sahil Jain, Samuel Kriman, Sanjeev Satheesh, Saori Kaji, Sarah Yurick, Saurav Muralidharan, Sean Narendhiran, Seonmyeong Bak, Sepehr Sameni, Seungju Han, Shanmugam Ramasamy, Shaona Ghosh, Sharath Turuvekere Sreenivas, Shelby Thomas, Shizhe Diao, Shreya Gopal, Shrimai Prabhuloye, Shubham Toshniwal, Shuoyang Ding, Siddharth Singh, Siddhartha Jain, Somshubra Majumdar, Soumye Singhal, Stefania Alborghetti, Syeda Nahida Akter, Terry Kong, Tim Moon, Tomasz Hliwiak, Tomer Asida, Tony Wang, Tugrul Konuk, Twinkle Vashishth, Tyler Poon, Udi Karpas, Vahid Noroozi, Venkat Srinivasan, Vijay Korthikanti, Vikram Fugro, Vineeth Kalluru, Vitaly Kurin, Vitaly Lavrukhin, Wasi Uddin Ahmad, Wei Du, Wonmin Byeon, Ximing Lu, Xin Dong, Yashaswi Karnati, Yejin Choi, Yian Zhang, Ying Lin, Yonggan Fu, Yoshi Suhara, Zhen Dong, Zhiyu Li, Zhongbo Zhu, and Zijia Chen. Nvidia nemotron nano 2: An accurate and efficient hybrid mamba-transformer reasoning model, 2025.

- [45] Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. How to train long-context language models (effectively). [arXiv preprint arXiv:2410.02660](#), 2024.
- [46] Yu Zhang and Songlin Yang. Flame: Flash language modeling made easy, January 2025.
- [47] Wanchao Liang, Tianyu Liu, Less Wright, Will Constable, Andrew Gu, Chien-Chin Huang, Iris Zhang, Wei Feng, Howard Huang, Junjie Wang, Sanket Purandare, Gokul Nadathur, and Stratos Idreos. Torch titan: One-stop pytorch native solution for production ready LLM pretraining. In [The Thirteenth International Conference on Learning Representations](#), 2025.
- [48] Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, et al. Openthoughts: Data recipes for reasoning models. [arXiv preprint arXiv:2506.04178](#), 2025.
- [49] Zihan Liu, Zhuolin Yang, Yang Chen, Chankyu Lee, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Acereason-nemotron 1.1: Advancing math and code reasoning through sft and rl synergy. [arXiv preprint arXiv:2506.13284](#), 2025.
- [50] Wasi Uddin Ahmad, Sean Narendhiran, Somshubra Majumdar, Aleksander Ficek, Siddhartha Jain, Jocelyn Huang, Vahid Noroozi, and Boris Ginsburg. Opencodereasoning: Advancing data distillation for competitive coding. [arXiv preprint arXiv:2504.01943](#), 2025.
- [51] Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, et al. Llama-nemotron: Efficient reasoning models. [arXiv preprint arXiv:2505.00949](#), 2025.
- [52] Haozhe Wang, Haoran Que, Qixin Xu, Minghao Liu, Wangchunshu Zhou, Jiazhan Feng, Wanjun Zhong, Wei Ye, Tong Yang, Wenhai Huang, et al. Reverse-engineered reasoning for open-ended generation. [arXiv preprint arXiv:2509.06160](#), 2025.

- [53] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyuan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. [arXiv preprint arXiv:2403.13372](#), 2024.
- [54] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. [arXiv preprint arXiv:2503.14476](#), 2025.
- [55] 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. [arXiv preprint arXiv:2402.03300](#), 2024.
- [56] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024.
- [57] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In [Thirty-seventh Conference on Neural Information Processing Systems](#), 2023.
- [58] HuggingFaceH4. Aime 2024. https://huggingface.co/datasets/HuggingFaceH4/aime_2024, 2024. 30 problems from AIME I & II 2024.
- [59] Chaoqun He, Renjie Luo, Yuzhuo Bai, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. [arXiv preprint arXiv:2402.14008](#), 2024.
- [60] 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. [arXiv preprint arXiv:2311.12022](#), 2023.
- [61] M-A-P Team, Xinrun Du, Yifan Yao, et al. Supergpqa: Scaling llm evaluation across 285 graduate disciplines. [arXiv preprint arXiv:2502.14739](#), 2025.
- [62] ByteDance-Seed. Beyondaime. <https://huggingface.co/datasets/ByteDance-Seed/BeyondAIME>, 2025. CC0-1.0 license.
- [63] Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, et al. Humanity's last exam. [arXiv preprint arXiv:2501.14249](#), 2025.
- [64] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In [Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 \(Long and Short Papers\)](#), pages 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.
- [65] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of Language Models: Part 3.3, Knowledge Capacity Scaling Laws. In [Proceedings of the 13th International Conference on Learning Representations, ICLR '25](#), April 2025. Full version available at <https://ssrn.com/abstract=5250617>.
- [66] Zeyuan Allen-Zhu. Physics of Language Models: Part 4.1, Architecture Design and the Magic of Canon Layers. [SSRN Electronic Journal](#), May 2025. <https://ssrn.com/abstract=5240330>.
- [67] Yuekun Yao, Yupei Du, Dawei Zhu, Michael Hahn, and Alexander Koller. Language models can learn implicit multi-hop reasoning, but only if they have lots of training data. [arXiv preprint arXiv:2505.17923](#), 2025.
- [68] Hanlin Zhu, Shibo Hao, Zhiting Hu, Jiantao Jiao, Stuart Russell, and Yuandong Tian. Reasoning by superposition: A theoretical perspective on chain of continuous thought. [arXiv preprint arXiv:2505.12514](#), 2025.
- [69] Shu Zhong, Mingyu Xu, Tenglong Ao, and Guang Shi. Understanding transformer from the perspective of associative memory. [arXiv preprint arXiv:2505.19488](#), 2025.
- [70] Chujie Zheng, Fan Yin, Hao Zhou, Fandong Meng, Jie Zhou, Kai-Wei Chang, Minlie Huang, and Nanyun Peng. On prompt-driven safeguarding for large language models. In [Proceedings of the 41st International Conference on Machine Learning](#), pages 61593–61613, 2024.
- [71] Kyle Cox. Post-hoc reasoning in chain of thought, December 2024. Blog post.

- [72] Iván Arcuschin, Jett Janiak, Robert Krzyzanowski, Senthooran Rajamanoharan, Neel Nanda, and Arthur Conmy. Chain-of-thought reasoning in the wild is not always faithful. [arXiv preprint arXiv: 2503.08679](#), 2025.
- [73] Fazl Barez, Tung-Yu Wu, Iván Arcuschin, Michael Lan, Vincent Wang, Noah Siegel, Nicolas Collignon, Clement Neo, Isabelle Lee, Alasdair Paren, Adel Bibi, Robert Trager, Damiano Fornasiere, John Yan, Yanai Lazar, and Yoshua Bengio. Chain-of-thought is not explainability. 2025.
- [74] Tomek Korbak, Mikita Balesni, Elizabeth Barnes, Yoshua Bengio, Joe Benton, Joseph Bloom, Mark Chen, Alan Cooney, Allan Dafoe, Anca Dragan, Scott Emmons, Owain Evans, David Farhi, Ryan Greenblatt, Dan Hendrycks, Marius Hobbahn, Evan Hubinger, Geoffrey Irving, Erik Jenner, Daniel Kokotajlo, Victoria Krakovna, Shane Legg, David Lindner, David Luan, Aleksander Mądry, Julian Michael, Neel Nanda, Dave Orr, Jakub Pachocki, Ethan Perez, Mary Phuong, Fabien Roger, Joshua Saxe, Buck Shlegeris, Martín Soto, Eric Steinberger, Jasmine Wang, Wojciech Zaremba, Bowen Baker, Rohin Shah, and Vlad Mikulik. Chain of thought monitorability: A new and fragile opportunity for ai safety. [arXiv preprint arXiv: 2507.11473](#), 2025.
- [75] Quora. Quora question pairs. <https://www.kaggle.com/competitions/quora-question-pairs/>, 2017. Kaggle competition.
- [76] Clayton Sanford, Bahare Fatemi, Ethan Hall, Anton Tsitsulin, Mehran Kazemi, Jonathan Halcrow, Bryan Perozzi, and Vahab Mirrokni. Understanding transformer reasoning capabilities via graph algorithms. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, [Advances in Neural Information Processing Systems](#), volume 37, pages 78320–78370. Curran Associates, Inc., 2024.
- [77] Clayton Sanford, Daniel Hsu, and Matus Telgarsky. Transformers, parallel computation, and logarithmic depth. [arXiv preprint arXiv:2402.09268](#), 2024.
- [78] Bingbin Liu, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. [arXiv preprint arXiv:2210.10749](#), 2022.
- [79] Zixuan Wang, Eshaan Nichani, Alberto Bietti, Alex Damian, Daniel Hsu, Jason D Lee, and Denny Wu. Learning compositional functions with transformers from easy-to-hard data. [arXiv preprint arXiv:2505.23683](#), 2025.
- [80] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In [International Conference on Learning Representations \(ICLR\)](#), 2021.
- [81] Yizhong Wang, Yada Pruksachatkun, Sheng Chen, Zexuan Zhong, Pengfei Chen, et al. MMLU-Pro: A more challenging and reliable evaluation for massive multitask language understanding. [arXiv preprint arXiv:2406.01574](#), 2024.
- [82] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. [arXiv preprint arXiv:2210.09261](#), 2022.
- [83] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. [arXiv preprint arXiv:1803.05457](#), 2018.
- [84] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In [Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics](#), pages 4791–4800, Florence, Italy, 2019. Association for Computational Linguistics.
- [85] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. [arXiv preprint arXiv:1907.10641](#), 2019.
- [86] 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. [arXiv preprint arXiv:2110.14168](#), 2021.
- [87] 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 [NeurIPS 2021 Datasets and Benchmarks Track](#), 2021.
- [88] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov,

- Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebbgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. [arXiv preprint arXiv:2107.03374](#), 2021.
- [89] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. [arXiv preprint arXiv:2108.07732](#), 2021.
- [90] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. [arXiv:1803.05457v1](#), 2018.
- [91] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset, Aug 2016.
- [92] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. [arXiv preprint arXiv:1809.02789](#), 2018.
- [93] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In [Proceedings of the AAAI conference on artificial intelligence](#), volume 34, pages 7432–7439, 2020.
- [94] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. [arXiv preprint arXiv:2203.15556](#), 2022.
- [95] Haoran Que, Jiaheng Liu, Ge Zhang, Chenchen Zhang, Xingwei Qu, Yinghao Ma, Feiyu Duan, Zhiqi Bai, Jiakai Wang, Yuanxing Zhang, et al. D-cpt law: domain-specific continual pre-training scaling law for large language models. In [Proceedings of the 38th International Conference on Neural Information Processing Systems](#), pages 90318–90354, 2024.

A Empirical Validation of Prior Choice

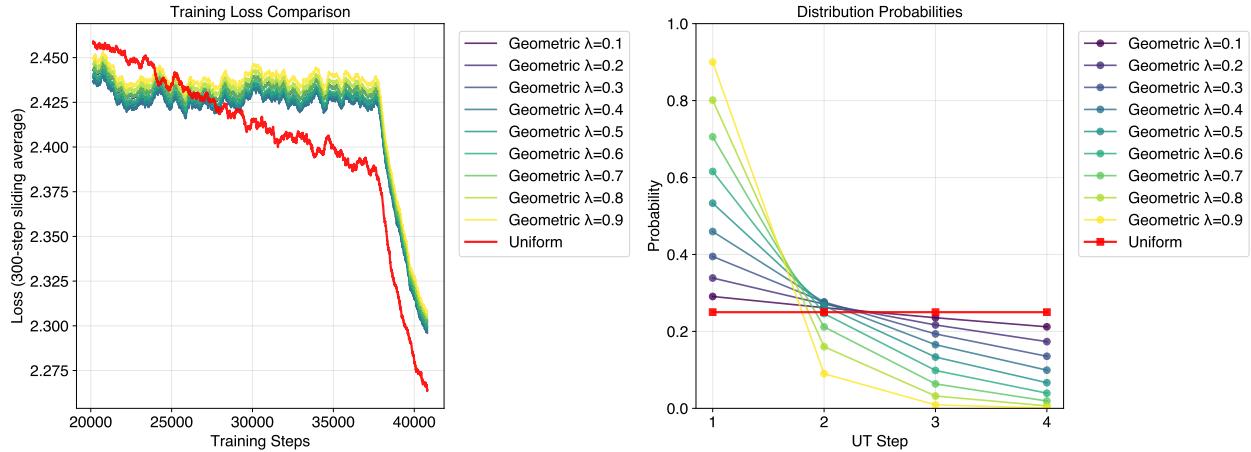


Figure 10 Effect of the prior over exit steps. Left: training loss (300-step sliding average) for a LoopLM with $T_{\max} = 4$ under different priors on z . Colored curves correspond to geometric priors with parameter $\lambda \in \{0.1, \dots, 0.9\}$; the red curve uses a uniform prior. Shaded regions indicate variability across runs. Right: prior probability over LoopLM steps induced by each λ (uniform shown in red). Stronger geometric bias (larger λ) concentrates mass on shallow steps, reducing credit assignment to deeper computation.

Experimental setup. Unless otherwise noted, we keep the model, data, optimizer, and schedule identical across conditions and only change the prior π used in the KL term of the loss. All results are obtained on a 776M-parameter LoopLM with $T_{\max} = 4$ recurrent steps. Training is performed on the FineWeb-Edu corpus [36] for a total of 20B tokens with a global batch of 50K tokens per optimization step, i.e., roughly 40K steps in total.⁵ For geometric priors we sweep $\lambda \in \{0.1, 0.2, \dots, 0.9\}$; the uniform prior assigns equal mass to all steps. To assess variability, we repeat each condition with multiple random seeds; shaded areas in Figure 10 denote the variability across runs. All other hyperparameters follow our training recipe, keeping β fixed across prior choices.

Convergence and final loss. As shown on the left of Figure 10, the uniform prior consistently achieves lower training loss and cleaner convergence on the 776M LoopLM. Geometric priors plateau higher, with the gap widening as λ grows (i.e., stronger bias toward early exit), reflecting weaker supervision for deeper iterations.

Stability and exploration. Geometric priors exhibit larger late-training oscillations, consistent with premature collapse of $q_\phi(z | x)$ onto shallow steps and reduced entropy. The uniform prior imposes no structural depth preference, so the KL term behaves as pure entropy regularization: exploration is maintained longer, and the model can allocate probability mass across multiple depths until it has learned which examples benefit from deeper computation.

Depth utilization. The right panel of Figure 10 visualizes the priors. Large- λ geometric priors concentrate mass at $t=1-2$, starving deeper steps ($t \geq 3$) of credit assignment; this undermines the “deeper is better” property. With a uniform prior, all depths receive comparable signal, enabling later iterations to specialize and deliver higher accuracy when maximum depth is allowed at inference.

Compute–accuracy trade-off. Although the uniform prior does not explicitly favor early exit, it does not preclude efficient inference: at test time we can still cap steps or apply a halting threshold. For a fixed average

⁵The loss curves plot a 300-step sliding average over the training trajectory.

step budget, models trained with a uniform prior achieve a strictly better accuracy–compute Pareto frontier than those trained with geometric priors, indicating that unbiased depth exploration during pretraining turns into better deployment trade-offs.

B Physics of LoopLMs

In this appendix, we conclude all the experimental settings and details in Section 6. Section B.1 includes the experiments on knowledge capacity; section B.2 includes the settings on knowledge manipulation synthetic tasks. Section B.3 introduces the detailed setting on the synthetic QA task following [67]. Finally, Section B.5 provides the theoretical results, detailed proof, and the discussion with the current theoretical results.

B.1 Capo: knowledge capacity

In this section, we introduce the knowledge capacity proposed in [65, 66]. The task evaluates models’ efficiency in memorizing factual knowledge within its parameters, which is measured by *bits per parameter*. We tested different sizes of models and visualize the knowledge scaling law through plotting *bits v.s. parameter number*.

Dataset: Synthetic Biographies We synthesize fake biographies following the bioS(N) dataset in [65]. Specifically, we generate N biographies of a random generated person together with their date of birth, city of birth, university, major, and employer. In our work, we online sample the individual attributes and generate the biographies in natural language using a random selected fixed template. An illustrative example is:

Layla Jack Beasley celebrates their birthday on January 24, 1914. They spent formative years in Portland, ME. They focused on Business Analytics. They supported operations for Delta Air Lines Inc. in Atlanta, GA. They received their education at Pepperdine University.

Model We use original GPT2 architecture and replace the positional encoding with RoPE [32]. In the Capo task, we tie the LM head and the embedding layer. To test the capability of universal transformer, we also added looping module s.t. the transformer blocks can be looped several times. We explore a broad range of model sizes varying in hidden dimension and depth. The notation $a\text{-}b\text{-}lc$ represents the model with $64a$ hidden dimensions (a attention heads with each head 64 dimensions), b layers, and c LoopLM steps (loops). The context length is set to 512.

Training details We use AdamW optimizer by setting $(\beta_1, \beta_2) = (0.9, 0.98)$, $\epsilon = 10^{-6}$ with 1000 steps of warmup followed by a cosine learning rate schedule from 1 to $0.1 \times$ of the original learning rate. We use bf16 training and packing is used during training. We masked different pieces of biographies from each other in each concatenated chunk.

We pass each data piece for 1000 times (similar to the 1000-exposure in [65]) during training. Since the final performance is not sensitive to learning rate choices, we consider learning rate $\eta = 0.001$, $wd = 0.02$, and total batch size 192. We pick $N \in \{20K, 50K, 100K, 200K, 500K\}$.

Evaluation: Knowledge Capacity Ratio After pretraining on the bioS(N) dataset, we assess a model’s knowledge capacity, defined as the number of bits of information it can reliably store. To make this measure comparable across models of different sizes, the raw bit count is normalized by the number of model parameters, yielding a “bits per parameter” metric. The derivation and motivation of the metric in discussed in [65]. For readers, we refer the detailed setting to Section 2.1 of [65].

Definition 1. Given a model F with P parameters trained over the bioS(N) dataset Z , suppose it gives $p_1 = loss_{name}(Z)$ and $p_2 = loss_{value}(Z)$, which are the sum of cross entropy loss on the name tokens and attribute tokens, respectively. The capacity ratio and the maximum achievable capacity ratio are defined as

$$R(F) \stackrel{\text{def}}{=} \frac{N \log_2 \frac{N_0}{e^{p_1}} + N \log_2 S_0 e^{p_2}}{P}, \quad R_{\max}(F) \stackrel{\text{def}}{=} \frac{N \log_2 N_0 \cdot N + N \log_2 S_0}{P},$$

for $N_0 = 400 \times 400 \times 1000$, $S_0 = 2 \times (12 \cdot 28 \cdot 200) \times 200 \times 300 \times 100 \times 263$ as all possible configurations.

Ignoring names, each person encodes approximately $\log_2(S_0) \approx 47.6$ bits of knowledge. The evaluation accounts for partial correctness. For instance, if a model recalls the year of a person’s birth but not the exact date, the partially correct information still contributes to the overall bit-level computation. This approach allows for a fine-grained measurement of knowledge retention, rather than relying on a strict all-or-nothing scoring.

B.2 Mano: knowledge manipulation

We followed [66] and used the Mano task to investigate the models’ capability of manipulating stored knowledge within the parameters without intermediate thoughts.

Dataset The dataset consists of modular arithmetic instances with tree structures of ℓ operations, where the number of operations $\ell \leq L$ as the maximum length. ℓ is uniformly sampled from $[1, L]$. The expressions are presented in prefix notation. For example, a length-3 instance is:

```
<bos> <len_3> - * a b + c d <ans> ans
```

which corresponds to $(a * b) + (c - d) \bmod 23$. All the operations are on \mathbb{F}_{23} . The task only involves $(+, -, *)$. The only tokens we use are the operations, numbers from 0 to 22, and the special `<bos>`, `<ans>` and length tokens `len_{i}` with $i \in [0, L]$.

Training details We use AdamW optimizer with $(\beta_1, \beta_2) = (0.9, 0.98)$, $\epsilon = 10^{-6}$ and gradient clipping with maximum norm 1.0. We employ 1000 steps of warmup followed by a cosine learning rate schedule to minimal learning rate 0.1 of the peak learning rate. We use bf16 training with packing and set the context length to 1024 tokens. Different pieces of mano problems are masked from each other in each concatenated chunk during training.

We conduct hyperparameter search over learning rates $lr \in \{0.00005, 0.0001, 0.0002, 0.0005\}$ with weight decay 0.1 and global batch size 128. We experiment with model depths $L \in \{10, 16, 24\}$ layers and hidden dimension 1024. Training is performed for $\{80K, 110K, 200K\}$ steps respectively for different difficulties. We run all experiments across 3 random seeds and report the best performance.

Evaluation During evaluation, we only use the expressions with the hardest length $\ell = L$. Accuracy is computed separately due to the masks. We consider exact match accuracy since the final answer is single-token.

B.3 Multi-hop question answering on synthetic relations

We followed [67] to construct the natural language multi-hop QA task. Comparing with Mano, the QA task is more knowledge-heavy and with a slightly simpler structure. [67] found that the model needs exponential many k -hop data for traditional transformer to learn. We chose this task to investigate if recursive structure in the reused-parameters can improve the sample efficiency of the task, showing better manipulation capability of LoopLM.

Dataset The dataset contains $|\mathcal{E}|$ entities—each with a unique name—and N relation types. We created 500 distinct single-token person names (e.g., Jennifer) and 20 single-token relation names (e.g., instructor) to serve as namespaces for entities and relations. We reused the name list in [67]. The complete list of relation names and a partial list of entity names appear in Tables 5 and 6 in [67]. The multi-hop questions are generated through a $K = 5$ hierarchical layers, where each layer has 100 individuals. Each entity is connected to $|\mathcal{R}|$ randomly chosen person in the next layer. This structure naturally generates $|\mathcal{E}|/5 \times |\mathcal{R}|^k$ k -hop questions. In our setting, since we only consider 3-hop questions, the number should be 8×10^5 .

For training, we use part of all the 3-hop training set and test on the leave-out 3000 test questions. For each test instance, we greedy decode the single token answer given the question prompt (e.g. ‘Who is the instructor of the teacher of Bob? \n Answer:’). We evaluate the exact match accuracy.

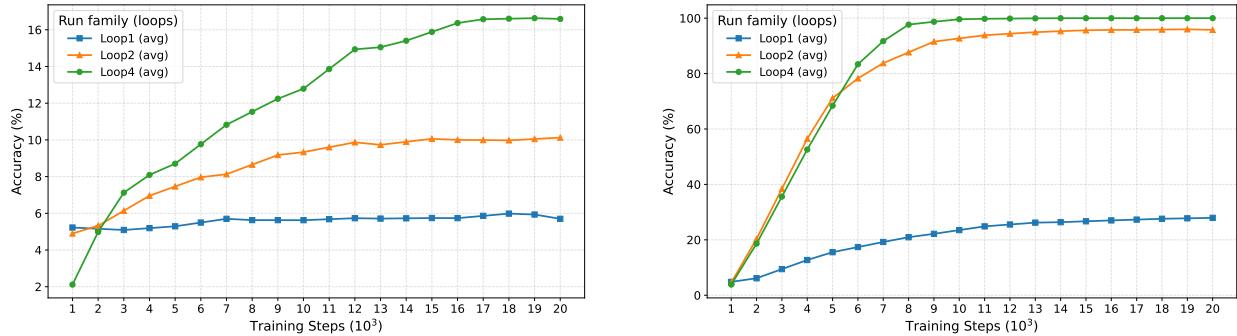


Figure 11 Left & Right. We further train with 100000 and 140000 unique QA pairs for 20000 steps with context length 1024 and batch size 2048. Similar to the main text, models with more loops learn faster and achieve better performance comparing with models without loops.

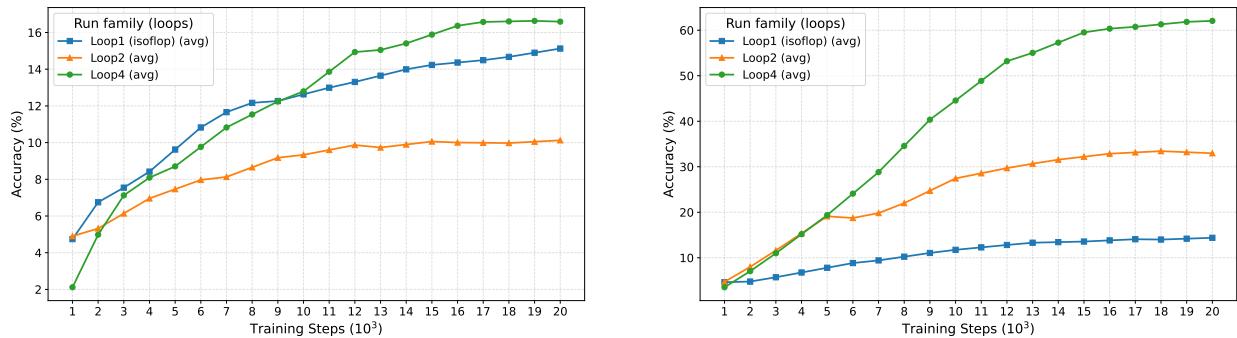


Figure 12 Left & Right. We further train with 100000 and 120000 unique QA pairs for 20000 steps with context length 1024 and batch size 2048. We train the baseline with 24 layers, which is equivalent flops with the loop 4 transformers. Similar to the main text, models with more loops learn faster and achieve better performance comparing with models without loops, even with iso-flop transformers. The loop 2 average performance is weaker than the iso-flop version transformer since it has less equivalent depth when $N = 10^5$, but it surpasses the baseline with more data provided.

Training details We use AdamW optimizer with $(\beta_1, \beta_2) = (0.9, 0.98)$, $\epsilon = 10^{-6}$ and gradient clipping 1.0. We run 1000 steps of linear warmup followed by a cosine learning rate schedule to minimal learning rate 0.1 of the peak learning rate. We use bf16 training with packing with context length 1024 tokens. QA pairs from distinct samples are masked from each other during training.

We use a base model architecture with 1024 hidden dimensions, 16 attention heads, and 6 layers. We allow it to loop in $\{1, 2, 3, 4\}$ times. Following the experimental setup in [67], we set the learning rate to 0.0005 with 1000 warmup steps and train for a total of 20,000 steps using batch size 2048. We run all experiments across 4 random seeds and report the average performance.

B.3.1 Additional experimental results

As the supplement of the main text, we present additional experiments to show that the superiority of LoopLM is general across different number of unique samples. For presentation, we only consider the interval of $\{10^5, 1.2 \times 10^5, 1.4 \times 10^5\}$ to exhibit the difference between looped models and non-looped baselines. We also checked iso-flop baseline models with the same hidden dimension and 24 layers⁶. The results are presented below in Figure 11 and Figure 12.

⁶We note that in Figure 12, the iso-flop baseline with $N = 1.2 \times 10^5$ does not perform significantly better than the shallower version in the main paper. We conjecture that it could be because of the randomness, or insufficient hyperparameter tuning. We believe further follow-up experiments should be necessary to further validate this conclusion here in the appendix.

B.4 Case study: improvements across different categories in MMLU

To validate our findings from synthetic tasks on a broad, real-world benchmark, we conducted a granular analysis of performance gains across all 57 sub-categories of MMLU. Our hypothesis is that if LoopLMs primarily enhance knowledge manipulation and reasoning, the largest performance gains should appear in procedural, reasoning-heavy tasks, while knowledge-heavy, retrieval-based subjects should see less improvement.

We measured the relative improvement by comparing the accuracy at the single recurrent step (Loop 1) against the accuracy at our fully trained depth (Loop 4). The detailed results for all 57 categories are available in **Table 15**. The analysis strongly supports our hypothesis. The categories with the most significant improvements are those requiring logical, maths, or procedural reasoning; conversely, categories that depend more on retrieving specific, memorized facts or nuanced world knowledge showed the most modest gains.

Example: Categories with the most significant/modest improvements

With most significant improvements:	With most modest improvements:
<ul style="list-style-type: none"> Elementary Mathematics: +155.6% Formal Logic: +143.3% Logical Fallacies: +127.8% High School Statistics: +126.9% 	<ul style="list-style-type: none"> Moral Scenarios: +7.8% Global Facts: +8.3% Virology: +13.7% Anatomy: +21.4%

This stark contrast indicates that the iterative computation is not simply increasing the model’s accessible knowledge (as seen in the nearly flat ‘global_facts’ improvement) but is actively performing the multi-step symbolic manipulation required for complex subjects like logic and math. This real-world benchmark result corroborates our synthetic findings in Section 6.2, confirming that the LoopLM architecture’s primary advantage lies in enhancing knowledge manipulation, not raw storage.

B.5 Theory: latent thought with LoopLM

In this section, we prove that LoopLM can solve the graph reachability problem (with part of the graph knowledge learned in the parameters) in $O(\log n)$ steps. The results are closely related to the expressivity power of transformers and looped transformers with padding [12, 13, 76, 77]. It matches the expressiveness lower bound results in [76, 77], and also resembles the *state tracking* tasks [78, 79] which requires $O(\log n)$ depths. We first define the task rigorously and state our main theorem. We finally discuss the theoretical improvement, caveats of the results, and all related theoretical results.

We first define our task based on the intuition of knowledge manipulation. Challenging knowledge manipulation tasks often have multiple steps or hierarchical structures, which requires the model to search in the knowledge graph with directional dependencies formed by the atomic facts or knowledge. Moreover, the context also contains conditions or new facts necessary for the problem. Therefore, we consider the searching task that requires the model to both **encode the fixed hidden knowledge graph** G in the parameters and **utilize the contextual information (additional graph)** G_{ctx} . The goal is to check if two queried nodes are connected. The formal definition is as follows (modified from [68]):

Definition 2 (Graph reachability on knowledge graph). *Let $V = \{v_1, v_2, \dots, v_n\}$ is the set of vertices and $E = \{e_1, e_2, \dots, e_m\}$ is the set of edges. Let $G = (V, E)$ be a directed hidden knowledge graph, and $G_{ctx} = (V, E_{ctx})$ be an input additional knowledge graph. Given a source node s a target node t , the task is to output 1 when there exists a path from s to t on the combined graph $G + G_{ctx} := (V, E + E_{ctx})$, and output 0 when s cannot reach t on the combined graph.*

Transformer architecture In this setting, we consider a simple single-head transformer architecture. We only use one-head and a two-layer gated MLP layer. For clearer theoretical demonstration, we use a special normalization layer $\text{LN}()$ to threshold on H : $\text{LN}(H)_{i,j} = \mathbf{1}\{H_{i,j} > 0\}$. And the overall architecture for each loop is (where Q, K, V, W_1, W_2 are all shared through layers)

$$H_{i+0.5} = \text{LN}(H_i + \text{Attn}_{Q,K,V}(H_i)), \quad \text{Attn}_{Q,K,V}(H_i) = VH_i \text{softmax}(H_i^\top K^\top QH_i)$$

Table 15 Performance metrics across different depths by category in MMLU.

Category	Loop 1	Loop 2	Loop 3	Loop 4	Improvement%
elementary_mathematics	0.3095	0.6190	0.7460	0.7910	155.5556
formal_logic	0.2381	0.4841	0.5238	0.5794	143.3333
logical_fallacies	0.3313	0.7178	0.7546	0.7546	127.7778
high_school_statistics	0.3102	0.5833	0.6620	0.7037	126.8657
high_school_macroeconomics	0.3692	0.6564	0.7513	0.7718	109.0278
management	0.3883	0.6699	0.7864	0.7961	105.0000
high_school_government_and_politics	0.3938	0.7202	0.8238	0.7979	102.6316
high_school_microeconomics	0.4076	0.7143	0.8361	0.8193	101.0309
high_school_psychology	0.4183	0.7817	0.8294	0.8385	100.4386
high_school_biology	0.4129	0.7452	0.8129	0.8161	97.6563
college_chemistry	0.2600	0.5000	0.5500	0.5000	92.3077
conceptual_physics	0.3830	0.6340	0.7149	0.7319	91.1111
college_biology	0.3889	0.6806	0.7569	0.7431	91.0714
machine_learning	0.2500	0.4286	0.5089	0.4732	89.2857
miscellaneous	0.3870	0.6564	0.7101	0.7178	85.4785
high_school_geography	0.4444	0.7121	0.7980	0.8182	84.0909
high_school_physics	0.2649	0.3841	0.5033	0.4834	82.5000
high_school_chemistry	0.3054	0.5320	0.5665	0.5517	80.6452
college_medicine	0.3584	0.5607	0.6416	0.6358	77.4194
college_computer_science	0.3500	0.5100	0.6000	0.6100	74.2857
professional_accounting	0.2872	0.4539	0.4929	0.5000	74.0741
world_religions	0.4211	0.6374	0.6959	0.7251	72.2222
high_school_computer_science	0.4600	0.6800	0.7600	0.7800	69.5652
clinical_knowledge	0.4151	0.5887	0.6415	0.6943	67.2727
astronomy	0.4013	0.6184	0.6711	0.6711	67.2131
prehistory	0.3765	0.5586	0.6389	0.6235	65.5738
high_school_us_history	0.4314	0.6912	0.7304	0.7108	64.7727
professional_psychology	0.3709	0.5392	0.5850	0.6095	64.3172
philosophy	0.4405	0.6495	0.7042	0.7106	61.3139
business_ethics	0.4200	0.6400	0.6500	0.6700	59.5238
high_school_mathematics	0.3000	0.4444	0.5037	0.4778	59.2593
high_school_european_history	0.5030	0.6909	0.7273	0.8000	59.0361
medical_genetics	0.4500	0.6900	0.7100	0.7100	57.7778
human_sexuality	0.4580	0.6336	0.7023	0.7176	56.6667
computer_security	0.4500	0.6200	0.6700	0.7000	55.5556
college_physics	0.2549	0.3333	0.3922	0.3922	53.8462
international_law	0.5124	0.7107	0.7769	0.7851	53.2258
marketing	0.5726	0.8547	0.8803	0.8761	52.9851
nutrition	0.4510	0.6634	0.6863	0.6863	52.1739
college_mathematics	0.2900	0.3700	0.4200	0.4400	51.7241
econometrics	0.3421	0.4123	0.5088	0.5175	51.2821
sociology	0.5373	0.7413	0.7910	0.8010	49.0741
professional_medicine	0.3787	0.5368	0.5735	0.5625	48.5437
high_school_world_history	0.5274	0.7300	0.7595	0.7722	46.4000
human_aging	0.4484	0.6143	0.6457	0.6502	45.0000
security_studies	0.5265	0.6898	0.7510	0.7592	44.1860
professional_law	0.3246	0.4055	0.4596	0.4570	40.7631
public_relations	0.4636	0.6182	0.6727	0.6364	37.2549
us_foreign_policy	0.5900	0.7200	0.8100	0.8000	35.5932
electrical_engineering	0.4483	0.5655	0.5862	0.6069	35.3846
abstract_algebra	0.2700	0.3000	0.3700	0.3600	33.3333
moral_disputes	0.5491	0.6503	0.6850	0.6994	27.3684
anatomy	0.4148	0.5111	0.5333	0.5037	21.4286
jurisprudence	0.5926	0.6944	0.7593	0.7130	20.3125
virology	0.4398	0.5000	0.4880	0.5000	13.6986
global_facts	0.3600	0.3700	0.3600	0.3900	8.3333
moral_scenarios	0.2436	0.2693	0.2492	0.2626	7.7982

$$H_{i+1} = \text{LN}(H_{i+0.5} + W_2 \text{ReLU}(W_1 H_{i+0.5}))$$

Input Format We define the adjacency matrix of the graph G as $A = [a_1, a_2, \dots, a_n] \in \mathbb{R}^{n \times n}$. Similarly, we define $A_{ctx} = [a_{1,ctx}, a_{2,ctx}, \dots, a_{n,ctx}]$. We use one-hot embeddings v_i to denote the vertex embedding. We consider the following input sequence format with length $n + 1$ for this task (assuming we already have the embeddings):

$$H_0 = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \\ a_{1,ctx} & a_{2,ctx} & \cdots & a_{n,ctx} \end{bmatrix} \in \mathbb{R}^{2n \times n}$$

where the first n tokens are the input context graph adjacency matrix. We assume the LoopLM recurrent for L steps, and we denote the hidden state sequence for the i -th recurrence:

$$H_i = \begin{bmatrix} v_1^{(i)} & v_2^{(i)} & \cdots & v_n^{(i)} \\ a_1^{(i)} & a_2^{(i)} & \cdots & a_n^{(i)} \end{bmatrix}.$$

For simplicity, we ignore the encoding and decoding process and have a direct output protocol: the final output for query (s, t) is the t -th entry of $a_s^{(L)}$.

Now we state the main theorem given the previous setting.

Theorem 2 (LoopLM solves reachability in $\log D$ steps). *Fix n as the maximum size of the combined knowledge graph G . Taking the adjacency matrix of the context graph $G_{ctx} \in \mathbb{R}^{n \times n}$ fed in as a n -token sequence and given a query pair (s, t) , there exists a one-layer, single-head transformer independent of G_{ctx} , with recurrent $O(\log_2 D)$ times and a hidden dimension of $d_e = 2n$ that can check whether there exists a path from s to t in the combined knowledge graph $(G + G_{ctx})$, where D is the diameter of $(G + G_{ctx})$.*

We directly construct the attention and the MLP layer for the LoopLM to implement an algorithm that is similar to Warshall's algorithm, using Boolean matrix powers with repeated squaring. The proof idea is to do a parallel search on **all pairs connectivity**, doubling the reachable distance with each loop. Since the maximum distance (i.e. diameter) is D , we only need $O(\log D)$ rounds to decide whether two nodes are connected or not. The attention enables each loop to iterative square the current adjacency matrix, and the MLP stores the hidden encoded G 's adjacency matrix to help internal knowledge manipulation.

Proof. We assign the parameters Q, K, V, W_1, W_2 as follows ($\beta \rightarrow +\infty$ is a large scalar):

$$K = \beta \begin{bmatrix} I_n & 0_{n \times n} \\ 0_{n \times n} & 0_{n \times n} \end{bmatrix}, Q = V = \begin{bmatrix} 0_{n \times n} & 0_{n \times n} \\ 0_{n \times n} & I_n \end{bmatrix}, W_2 = I_{2n}, W_1 = \begin{bmatrix} 0_{n \times n} & 0_{n \times n} \\ A & 0_{n \times n} \end{bmatrix}.$$

Recall that the input sequence is

$$H_0 = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \\ a_{1,ctx} & a_{2,ctx} & \cdots & a_{n,ctx} \end{bmatrix} \in \mathbb{R}^{2n \times n},$$

which only contains the input context graph's adjacency matrix. We assume the LoopLM loops for L steps, and we denote the hidden state sequence for the i -th loop:

$$H_i = \begin{bmatrix} v_1^{(i)} & v_2^{(i)} & \cdots & v_n^{(i)} \\ a_1^{(i)} & a_2^{(i)} & \cdots & a_n^{(i)} \end{bmatrix}.$$

For simplicity, we directly output the t -th entry of $a_s^{(L)}$ for query (s, t) . The model should check whether (s, t) are connected, i.e. $(a_s^{(L)})_t = 1$ or 0. We are going to prove by induction that for each recursion, $a_j^{(i)}$ contains all the vertices v_k (i.e. $(a_j^{(i)})_k = 1$) that vertex v_j is connected to, and the distance between v_j and v are less than or equal to 2^{i-1} . Therefore, we only need $\log D + 1$ loops to get the final answer.

Base. When $i = 1$, the constructed parameters ensure that the j -th node v_j attend to all the nodes that are directly connected to v_j with the same attention score β . This guarantees that the attention layer will average

the nodes' tokens that v_j is connected to. The j -th column of the attention before the thresholding layer becomes ($|a_{j,ctx}|$ means the nodes that v_j connects to)

$$\begin{bmatrix} v_j \\ a'_j \end{bmatrix} = \begin{bmatrix} v_j \\ a_{j,ctx} \end{bmatrix} + \frac{1}{|a_{j,ctx}|} \sum_{k:(a_{j,ctx})_k=1} \begin{bmatrix} 0_n \\ a_{k,ctx} \end{bmatrix}$$

This updated adjacency vector naturally contains all nodes that has distance ≤ 2 to v_j in context graph G_{ctx} after the thresholding layer. It naturally includes nodes with distance 1.

Now we consider the output of the MLP layer, which only adds the adjacency matrix of hidden knowledge graph G to the residual stream.

$$\begin{bmatrix} v_j \\ a''_j \end{bmatrix} = \begin{bmatrix} v_j \\ a'_j \end{bmatrix} + W_2 \text{ReLU} \left(W_1 \begin{bmatrix} v_j \\ a'_j \end{bmatrix} \right) = \begin{bmatrix} v_j \\ a'_j \end{bmatrix} + \begin{bmatrix} 0_n \\ a_j \end{bmatrix},$$

which combines the adjacency matrices of the context graph G_{ctx} and the hidden knowledge graph G . After the thresholding in the end, all non-zero entries become 1, which already includes all distance 1 nodes for all nodes. Therefore, we have that $a_{j,ctx}^{(1)}$ contains all reachable nodes within distance 1 of node v_j after the first recursion.

Induction. Assume at the recursion step i ($i \geq 1$), all $a_j^{(i)}$ contains all reachable vertices within distance 2^{i-1} of v_j . Now the hidden state sequence is going through loop $i+1$. To finish the proof, we show that $a_j^{(i+1)}$ contains all reachable vertices within distance 2^i of v_j .

The attention for this stage looks at $a_j^{(i+1)}$ now, and the j -th node v_j uniformly attend to all the nodes that are connected to v_j within distance 2^{i-1} . The j -th column of the attention before the thresholding layer becomes ($|a_j^{(i)}|$ means the number of nodes)

$$\begin{bmatrix} v_j \\ a_j^{(i+0.5)} \end{bmatrix} = \begin{bmatrix} v_j \\ a_j^{(i)} \end{bmatrix} + \frac{1}{|a_j^{(i)}|} \sum_{k:(a_j^{(i)})_k=1} \begin{bmatrix} 0_n \\ a_k^{(i)} \end{bmatrix}$$

After thresholding, the adjacency vector aggregates $a_j^{(i)}$ and all $a_k^{(i)}$ where v_k has distance $\leq 2^{i-1}$ to v_j in combined graph $G_{ctx} + G$. Meanwhile, $a_k^{(i)}$ contains all vertices connected to v_k with distance $\leq 2^{i-1}$. Therefore, the combined vector includes all nodes within distance 2^i of v_j .

Finally, we consider the final MLP:

$$\begin{bmatrix} v_j \\ a_j^{(i+1)} \end{bmatrix} = \text{LN} \left(\begin{bmatrix} v_j \\ a'_j \end{bmatrix} + W_2 \text{ReLU} \left(W_1 \begin{bmatrix} v_j \\ a_j^{(i+0.5)} \end{bmatrix} \right) \right) = \text{LN} \left(\begin{bmatrix} v_j \\ a_j^{(i+0.5)} \end{bmatrix} + \begin{bmatrix} 0_n \\ a_j \end{bmatrix} \right),$$

which also contains all vertices connected to v_j with distance 2^i . That means $(a_s^{(i+1)})_t$ is a precise indicator of whether (s, t) is connected within 2^i distance. By induction, we finish the proof and with $L = \lceil \log_2 D \rceil + 1$ recursion steps, the model can correctly solve reachability on the combined graph $G + G_{ctx}$. \square

Discussion on related theoretical results We provided a modified construction from [12], which requires n^3 padding tokens that increase the computation complexity to $O(n^6)$. However, our construction requires $O(n)$ hidden dimension as the continuous CoT did in [68]. The $\Theta(n)$ requirement is necessary because the superposition in latent space needs to (in the worst case) encode $\Theta(n)$ nodes' information, which can be a theoretical limitation. The requirement can be relaxed when there is an upper bound for the maximum number of vertices that some vertex is connected to.

Input format In our construction, the input format is the adjacency matrix of the graph. As a natural alternative, [68] used a sequence with length $O(n^2)$ as the input to encode all the different edges. To make LoopLM also work on this setting, some additional induction head mechanism (similar to [68]) is needed to extract all edges and combine them into the adjacency matrix. With slight modification, we can still get the solution with sequential steps $O(\log D)$.

C Evaluations

C.1 Evaluation Settings

Base Model Evaluation Settings Table 16 details the evaluation settings and frameworks used for the Ouro base models.

Table 16 Evaluation settings and benchmark sources for base models.

Benchmark	Settings	Framework
General		
MMLU [80]	logprobs, 5-shot	lm-eval-harness
MMLU-Pro [81]	strict match, 5-shot CoT	lm-eval-harness
BBH [82]	strict match, 3-shot CoT	lm-eval-harness
ARC-C [83]	logprobs, 25-shot	lm-eval-harness
HellaSwag [84]	logprobs, 10-shot	lm-eval-harness
Winogrande [85]	logprobs, 5-shot	lm-eval-harness
Math		
GSM8k [86]	strict match, 3-shot CoT	lm-eval-harness
MATH500 [87]	strict match, 5-shot CoT	In-house
Code		
HumanEval [88]	pass@1	evalplus
HumanEval+ [57]	pass@1	evalplus
MBPP [89]	pass@1	evalplus
MBPP+ [57]	pass@1	evalplus

Reasoning Model Evaluation Settings Table 17 details the evaluation settings and protocol used for the Ouro-Thinking reasoning models, as described in Section 5.2. All reasoning benchmarks utilized an in-house evaluation harness and an LLM-as-judge protocol with a fixed rubric.

Table 17 Evaluation settings and protocol for reasoning models (Ouro-Thinking).

Benchmark	Protocol	Decoding Settings
AIME 2024/2025	In-house harness; LLM-as-judge	temp=1.0, top_p=0.7
OlympiadBench	In-house harness; LLM-as-judge	temp=1.0, top_p=0.7
GPQA	In-house harness; LLM-as-judge	temp=1.0, top_p=0.7
SuperGPQA	In-house harness; LLM-as-judge	temp=1.0, top_p=0.7
BeyondAIME	In-house harness; LLM-as-judge	temp=1.0, top_p=0.7
HLE	In-house harness; LLM-as-judge	temp=1.0, top_p=0.7

D Scaling Law for LoopLMS

To further explore the potential of LoopLM, we conduct a series of small-scale experiments to investigate the scalability and predictability inherent in LoopLM. Specifically, our work focuses on the following three research questions:

- **RQ1:** What is the performance gap between standard models and LoopLM?
- **RQ2:** How do recurrent steps impact the total loss and step-wise loss in the context of LoopLM?
- **RQ3:** What is the inherent connection between total loss and step-wise loss?

D.1 RQ1: What is the performance gap between standard models and LoopLM?

To understand the performance gap between standard models and LoopLM, we quantify this difference in terms of benchmark performance. We also observe how this gap varies with changes in recurrent step and model size to guide the further scaling and iteration of LoopLM.

Experimental Setup We prepare five model sizes: 53M, 134M, 374M, 778M, and 1.36B. For recurrent steps, we prepare four different depths: 1, 2, 4, and 8. It is worth noting that for standard models, different recurrent steps effectively multiply the number of layers in the model. We evaluate the performance on the following benchmarks: ARC-Challenge [90], ARC-Easy [90], HellaSwag [84], LAMBADA [91], OpenBookQA [92], and PIQA [93]. In all sub-experiments, we train on 20B tokens using the FineWeb-Edu corpus [36]. We present the benchmark performance from the final step. For LoopLM, the recurrent step used for evaluating is the maximum recurrent step.

By observing the trends in the curves, we derive the following observations:

1. Whether standard models or LoopLM, the model’s performance improves with increasing model size and recurrent step. As shown in Figure 13, for all recurrent steps, the benchmark performance of both the LoopLM and Standard models increases as the model size grows, which aligns with the principle of LLMs: larger is better. As shown in Figure 14, except for the LoopLM at 778M and 1.364B, both the LoopLM and Standard models show that benchmark performance increases as the recurrent step increases. This indicates that latent reasoning is indeed useful for both the LoopLM and Standard Transformer.

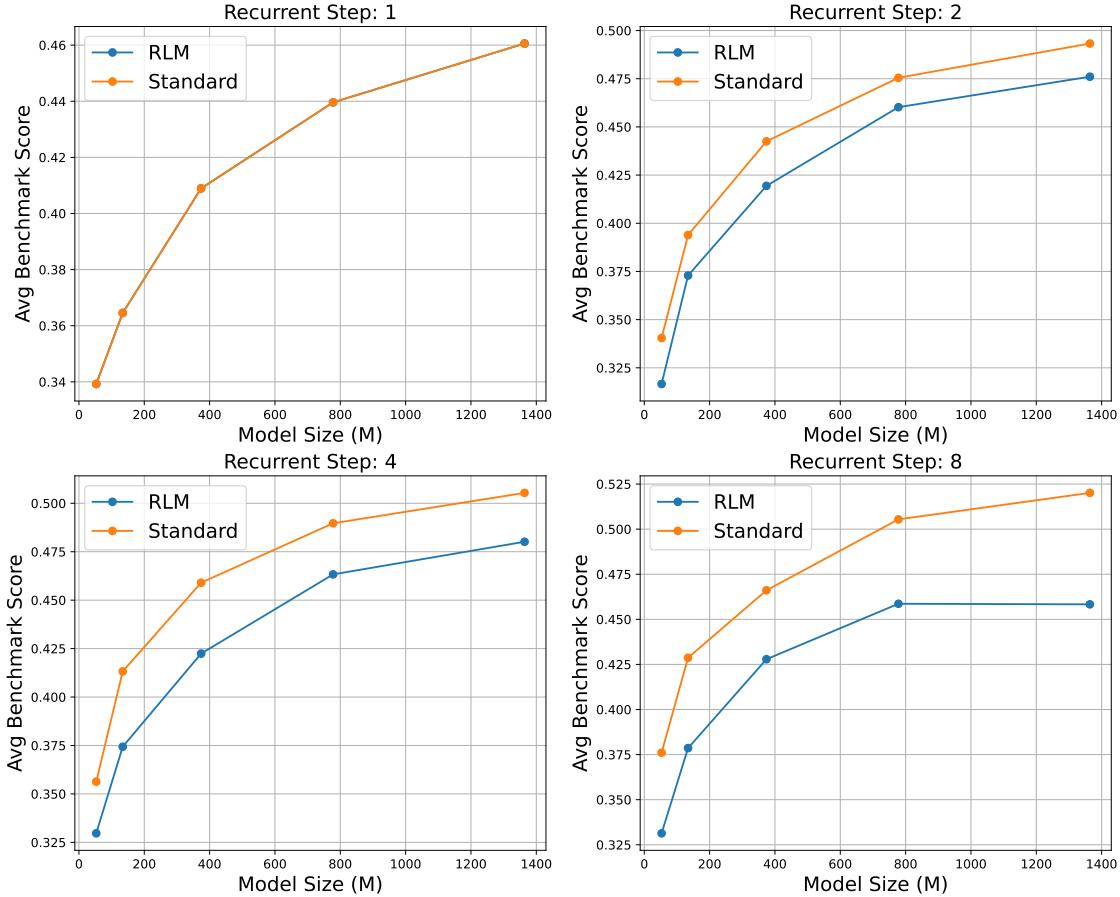


Figure 13 The average benchmark performance of LoopLM and Standard Transformer models under different recurrent steps as model size varies. With a recurrent step of 1 (*top left*), both models have identical architectures, resulting in overlapping curves. Overall, as the model size increases, the benchmark performance improves. The average benchmark score demonstrates the average results of the six benchmarks.

2. Overall, the performance of the standard model exceeds that of LoopLM under the same conditions. This gap increases with the recurrent step and decreases with the model size. Observing Figure 13 and Figure 14, it is clear that the benchmark performance of the Standard model is consistently higher than that of LoopLM, indicating that the Standard model has a scoring advantage without considering computational budget.

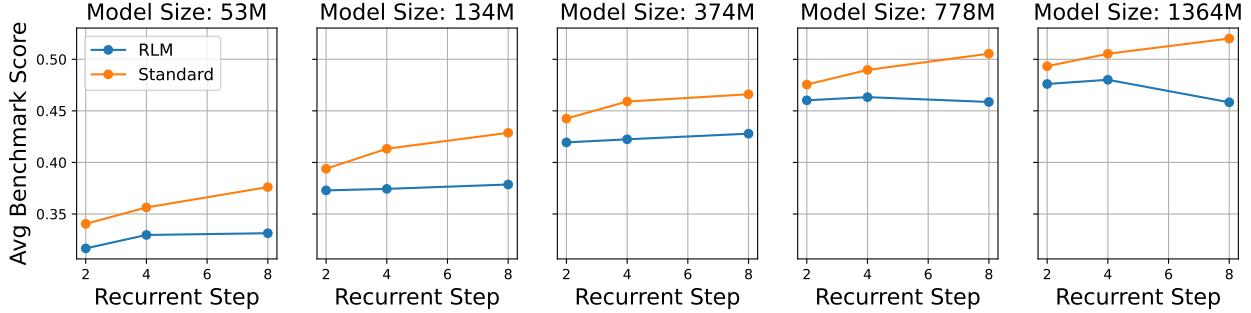


Figure 14 The average benchmark performance of LoopLM and Standard Transformer models under different model sizes as recurrent step varies. Except for the LoopLM at the model size of 778M and 1.364B, in all other cases, the benchmark performance of the model increases with the increase in recurrent steps.

Furthermore, we define the benchmark performance gap as the benchmark performance of the Standard model minus that of LoopLM, and this value is positive in all our experiments. As shown in Table 18, as the recurrent step increases, the benchmark performance gap also increases, suggesting that as the number of recurrences rises, the effect of models not sharing parameters gradually surpasses that of models sharing parameters. Besides, we find that the benchmark performance gap generally has a negative correlation with model size when the maximum recurrent step is relatively low, meaning that as the model size increases, the performance of LoopLM becomes closer to that of the Standard model, resulting in a smaller gap between the two. This trend is particularly consistent when the recurrent step is 4.

Table 18 The average benchmark performance gap between LoopLM and Standard models as the recurrent step varies at different model sizes. The gap is defined as (Standard model score - LoopLM score). As the recurrent step increases, the performance gap generally increases.

Model Size	Average Performance Gap	
	Step 2	Step 4
170M	0.021	0.039
340M	0.023	0.037
680M	0.015	0.026
1.3B	0.017	0.025

D.2 RQ2: How do recurrent step impact the total loss and step-wise loss in the context of LoopLM?

In this subsection, we investigate the predictability and generalizability of LoopLM from the perspective of training loss, examining the impact of recurrent step on the trends in total loss and step-wise loss. The experiment is set up in complete consistency with Section D.1, but we focus more on the total loss and step-wise loss during the training process. Here, step-wise loss refers to the loss of the same LoopLM at different recurrent step.

Here, we have following variables: model size N , training data size D , maximum recurrent step T_m , recurrent step T , total loss L_t , and step-wise loss L_s . Following Chinchilla [94], we first attempt to fit the relationship between L_t and N, D, T_m in the form of a power law:

$$L_t = E + \frac{A}{(N + t_1)^\alpha} + \frac{B}{(D + t_2)^\beta} + \frac{C}{(T_m + t_3)^\gamma}$$

The purpose of t_1 , t_2 , and t_3 is to prevent the variables from exploding in value near zero, allowing the fitting curve to be smoother. We refer to above formula as the **Total Loss Scaling Law**. First, to validate the

predictability of LoopLM, we fit all the data points, and the resulting curve is shown in Figure 15. We find that the actual loss curve and the predicted loss curve are highly consistent, demonstrating the predictability of LoopLM in terms of model size, training data size, and max recurrent step. We quantify the consistency of the scaling law using the coefficient of determination R^2 . An absolute value of the R^2 closer to 1 indicates a better fit, with positive values representing a positive correlation and negative values representing a negative correlation. Fitting the Total Loss Scaling Law using all data points and calculating R^2 with all data points, we obtain an R^2 value of 0.9596. This confirms the strong correlation between total loss and model size, training data size, and max recurrent step, demonstrating the predictability of the Total Loss Scaling Law.

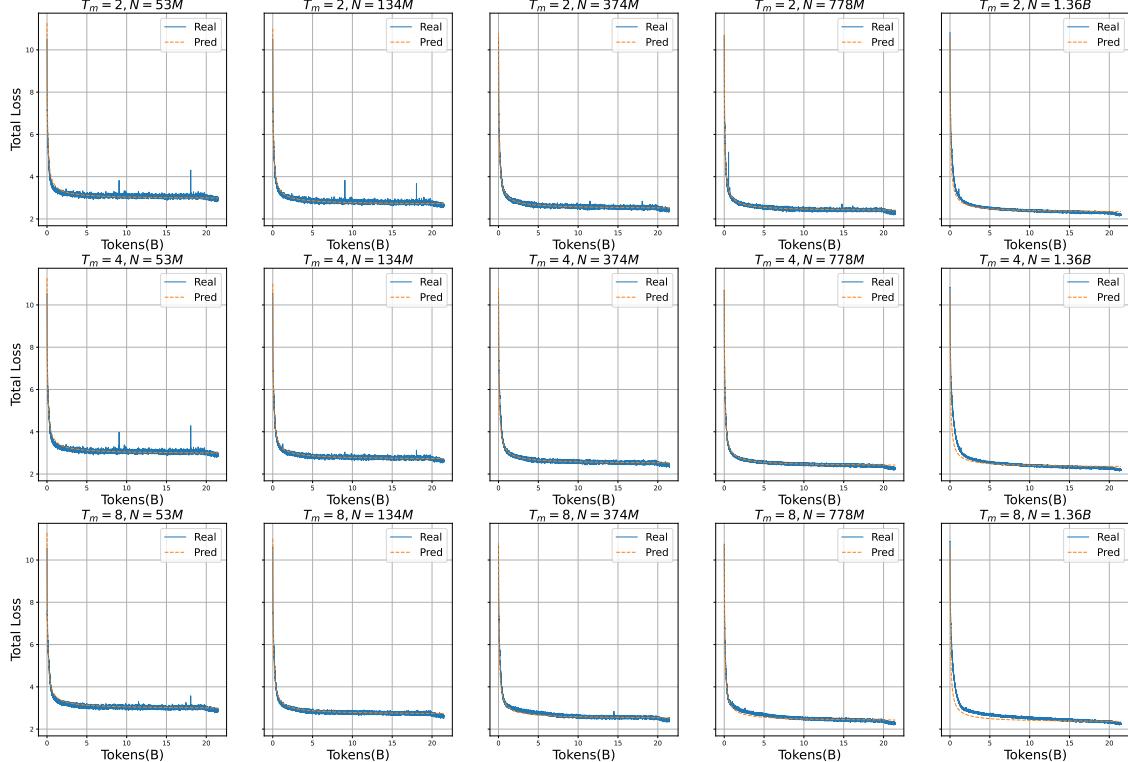


Figure 15 Illustration of the actual loss curve and the loss curve predicted by the scaling law. To demonstrate the predictability of LoopLM, we have used all data points for fitting, proving its predictability in terms of model size, training data size, and max recurrent step. The orange dashed line represents the prediction, while the blue solid line represents the actual loss.

In addition to its predictability, we further explore the generalizability of the Total Loss Scaling Law. Predictability refers to the ability of the Scaling Law to fit all data points into a unified curve when all data points are available. Generalizability, on the other hand, indicates whether the Scaling Law can predict unseen data points when fitting is done with a subset of data points. For example, generalizability tests whether the performance of a 14B model can be predicted using the known performances of 1B and 7B models [95]. To verify its generalizability across model size N , training data size D , and maximum recurrent step T_m , we have conducted related experiments, details can be found in Appendix E.1.

During the LoopLM training process, we compute the cross-entropy loss at each recurrent step, which we refer to as step-wise loss L_s . We aim to explore the relationship between step-wise loss L_s and the current recurrent step T , model size N , and training data size D . Similarly, we can fit the scaling law between L_s and N, D, T , with the formula as follows:

$$L_s = E + \frac{A}{(N + t_1)^\alpha} + \frac{B}{(D + t_2)^\beta} + \frac{C}{(T + t_3)^\gamma}$$

We refer to the above formula as the **Step-wise Loss Scaling Law**. We also present the fitting effectiveness from the perspectives of predictability and generalizability. Regarding predictability, we fit all data points. Even with the same recurrent step, the loss curve can vary significantly across different maximum recurrent steps. To ensure the independence of the variable recurrent step T , we do not consider the maximum recurrent step in the Step-wise Loss Scaling Law formula and focus solely on the relationship between L_s and N, D, T . Therefore, we have a total of three major experiments, each representing the fitting of the Step-wise Loss Scaling Law for maximum recurrent steps of 2, 4, and 8. The fitting results of the Step-wise Loss Scaling Law are shown in Figure 16, Figure 17, and Figure 18, which illustrate the trends of the actual and fitted curves for maximum recurrent steps of 2, 4, and 8, respectively. We find that in some cases in Figure 17 and Figure 18, L_s increases with the increase in D . We consider this a special case and will discuss it in detail in Section D.3; we will ignore these outlier data points during fitting. The R^2 for the three max recurrent steps are 0.8898, 0.8146, and 0.795, respectively. As the maximum recurrent step increases, the increase in the number of data points leads to lower R^2 values. The step-wise loss itself is less stable than the total loss, resulting in greater variability. Thus, the obtained R^2 values are not as high as those of the Total Loss Scaling Law. However, it is still evident that the scaling law is able to capture the overall trend of the curves, demonstrating the predictability of the Step-wise Loss Scaling Law. The fitting parameter γ of the Step-wise Loss Scaling Law is positive, indicating that L_s decreases as the recurrent step increases. This aligns with our original intent in the design of the recurrence. Besides, we present the generalizability of the Step-wise Loss Scaling Law in Appendix E.2.

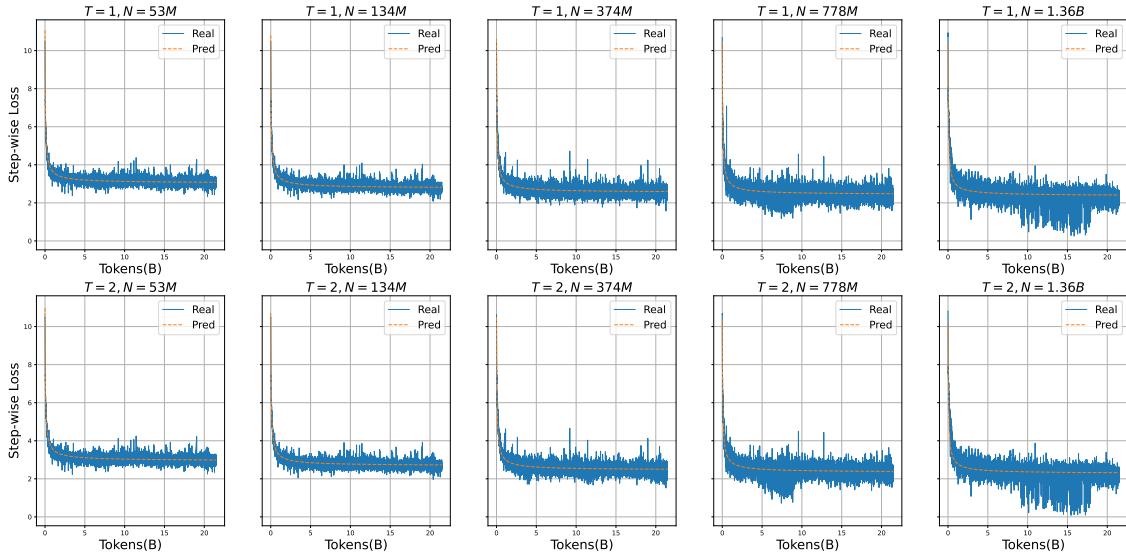


Figure 16 Illustration of the actual loss curve and the loss curve predicted by the Step-wise Loss Scaling Law when the maximum recurrent step is equal to 2.

In summary, both total loss and step-wise loss exhibit a strong correlation with $N, D, T/T_m$. The fitting results demonstrate the predictability and generalizability of the Scaling Law for LoopLM. In next section, we will explore the relationship between total loss and step-wise loss in greater depth.

D.3 RQ3: What is the inherent connection between total loss and step-wise loss?

We first review the training objectives of LoopLM:

$$L_t = \sum_{T=1}^{T_m} q_\phi(z = t | x) L_s^{(T)} - \beta \cdot H(q_\phi(z | x))$$

$L_s^{(T)}$ represents the step-wise loss at the recurrent step T . By analyzing the above form, we can see that the

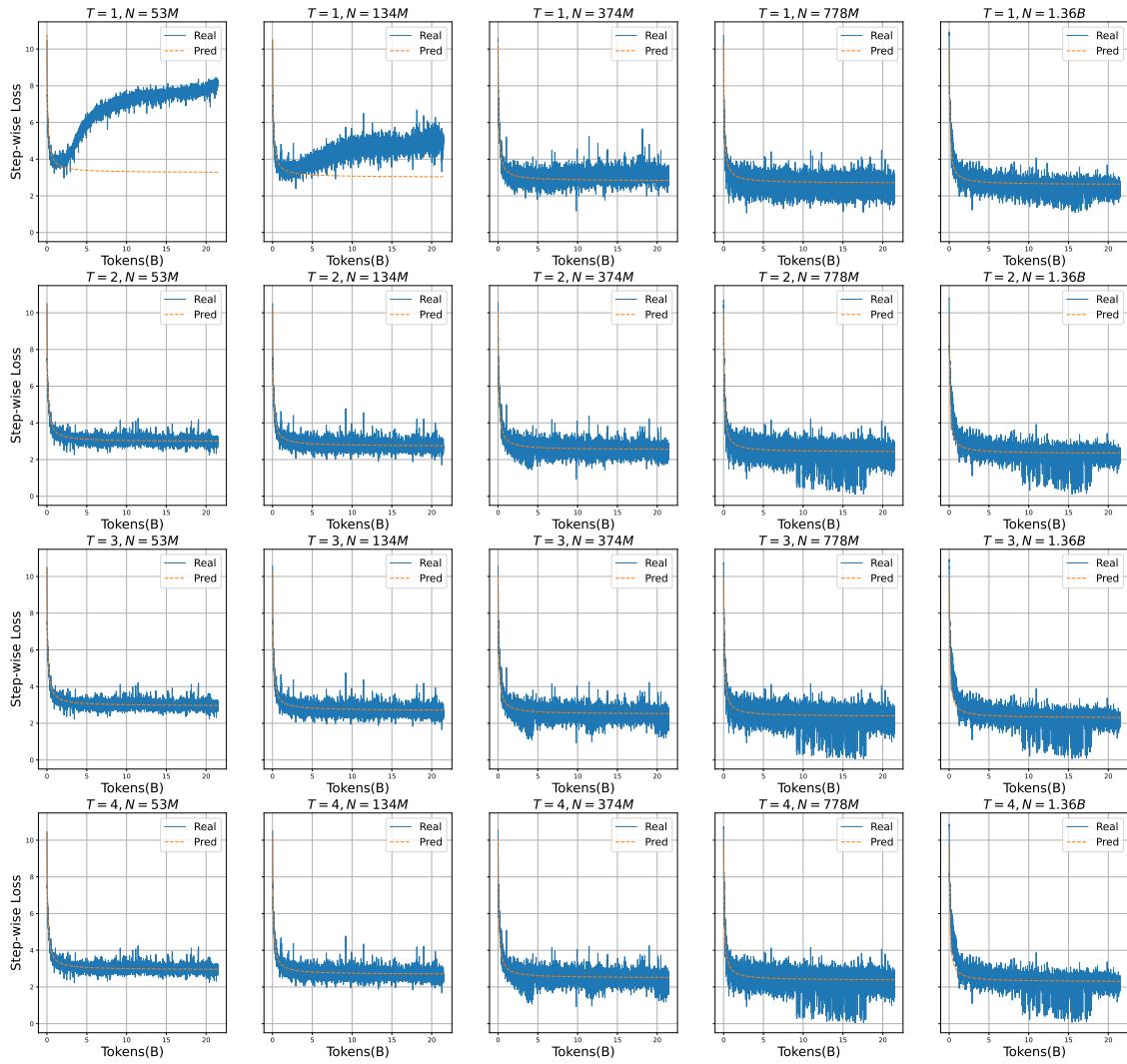


Figure 17 Illustration of the actual loss curve and the loss curve predicted by the Step-wise Loss Scaling Law when the maximum recurrent step is equal to 4.

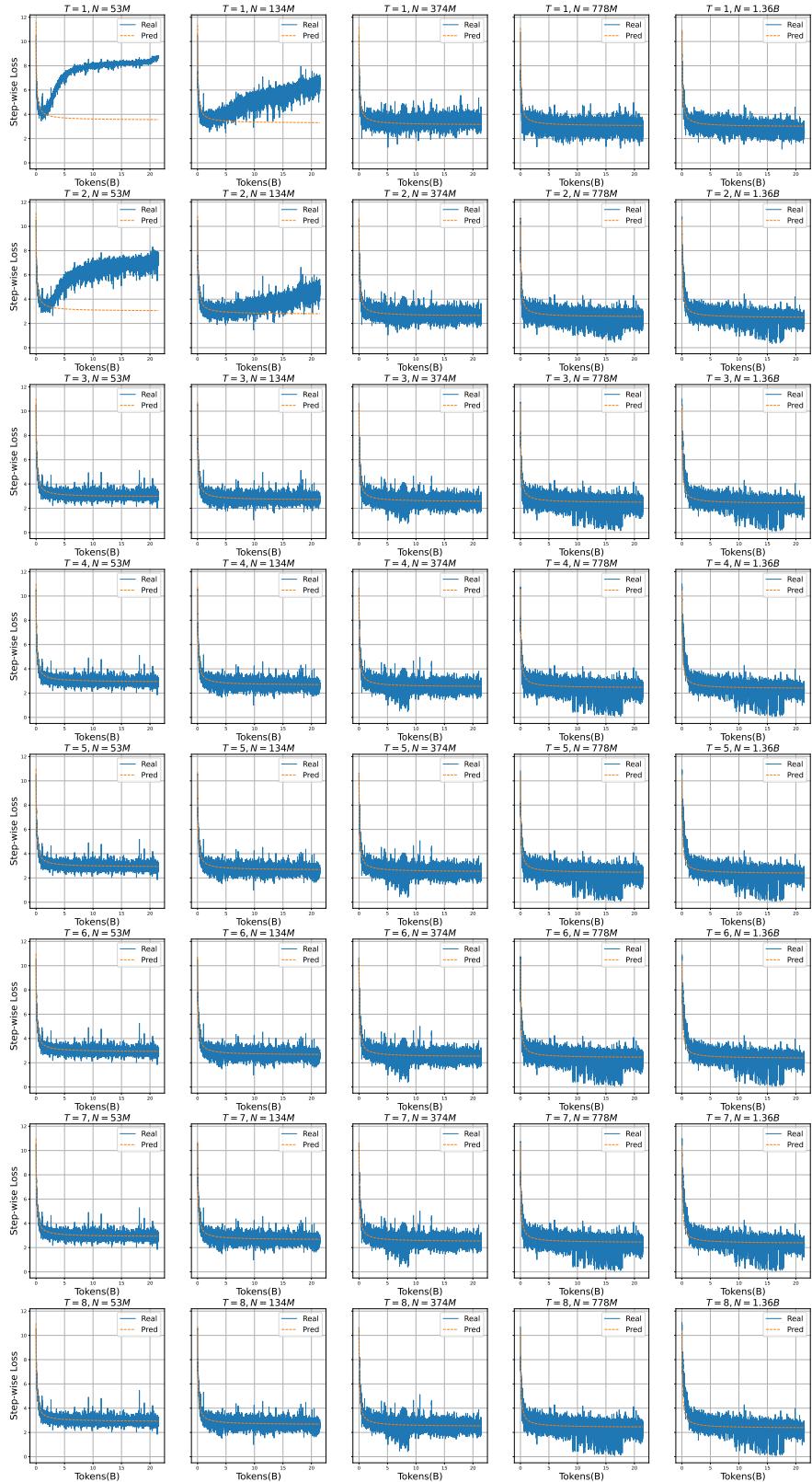


Figure 18 Illustration of the actual loss curve and the loss curve predicted by the Step-wise Loss Scaling Law when the maximum recurrent step is equal to 8.

total loss consists of two components. The first part is the expected task loss, which is a weighted sum of the step-wise loss. The second part is entropy regularization, whose primary purpose is to ensure that the learned gating mechanism q_ϕ does not converge to a specific recurrent step.

In our extensive small-scale experiments, we have observed an interesting phenomenon: the exploitation of total loss for shallow step-wise loss. To be specific, as shown in Figure 17 and Figure 18, when the model size is insufficient, the shallow step-wise loss increases with the growing amount of training data. This is an unusual phenomenon, typically, all step-wise losses should decrease as the amount of training data increases. We attempt to explain this phenomenon. In Section D.2, it is mentioned that the step-wise loss decreases with an increasing recurrent step, indicating that deeper recurrent steps result in lower L_s . To minimize the expected task loss, the learned gating mechanism assigns more weight to deeper recurrent steps. However, entropy regularization ensures that the learned gating mechanism does not deviate too much from the prior distribution. When the model size is insufficient, the amount of information it can encode is limited. To further reduce the total loss, this results in an increase in shallow step-wise loss, which in turn allows the weights to favor higher recurrent steps to lower the total loss. Thus, to ensure that the trend of step-wise loss remains normal, a larger model size may be more effective for LoopLM.

As mentioned in Section D.2, the scaling law for LoopLM is predictable and generalizable for both total loss and step-wise loss. We have:

$$L_t = E_t + \frac{A_t}{(N + t_{1t})^\alpha} + \frac{B_t}{(D + t_{2t})^\beta} + \frac{C_t}{(T_m + t_{3t})^\gamma}$$

$$L_s^{(T)} = E_s + \frac{A_s}{(N + t_{1s})^\alpha} + \frac{B_s}{(D + t_{2s})^\beta} + \frac{C_s}{(T + t_{3s})^\gamma}$$

The subscripts s and t represent the fitting parameters for the Step-wise Loss Scaling Law and the Total Loss Scaling Law, respectively. By substituting the Step-wise Loss Scaling Law into the training objectives, we have:

$$L_t = \sum_{T=1}^{T_m} q_\phi(z = t | x) \left(E_s + \frac{A_s}{(N + t_{1s})^\alpha} + \frac{B_s}{(D + t_{2s})^\beta} + \frac{C_s}{(T + t_{3s})^\gamma} \right) - \beta \cdot H(q_\phi(z | x))$$

For the first three terms in $L_s^{(T)}$, the sum of q_ϕ equals 1, allowing us to factor it out, which gives us:

$$L_t = E_s + \frac{A_s}{(N + t_{1s})^\alpha} + \frac{B_s}{(D + t_{2s})^\beta} + \sum_{T=1}^{T_m} q_\phi(z = t | x) \frac{C_s}{(T + t_{3s})^\gamma} - \beta \cdot H(q_\phi(z | x))$$

As the amount of training data increases, the learned gating mechanism q_ϕ stabilizes, and we observe that the value of the entropy regularization term becomes relatively low, accounting for approximately 1% to 5% of the total loss. If the form of q_ϕ gradually stabilizes, we treat it as a constant term, and the formula becomes:

$$L_t = E_s + \frac{A_s}{(N + t_{1s})^\alpha} + \frac{B_s}{(D + t_{2s})^\beta} + E_{other}$$

In Section D.2, when considering the Step-wise Loss Scaling Law, we will fix the maximum recurrent step. Once we determine the model's maximum recurrent step T_m , the forms of the above formula and the Total Loss Scaling Law are completely consistent, indicating that there is a trend consistency in the scaling law between total loss and step-wise loss.

We further demonstrate this through practical experiments. We take the situation where the max recurrent step is equal to 4. First, we perform a standard fitting of the Step-wise Loss Scaling Law to obtain the fitting parameters E_s, A_s, B_s, C_s , and so on. Next, we observe and record the distribution of q_ϕ for each

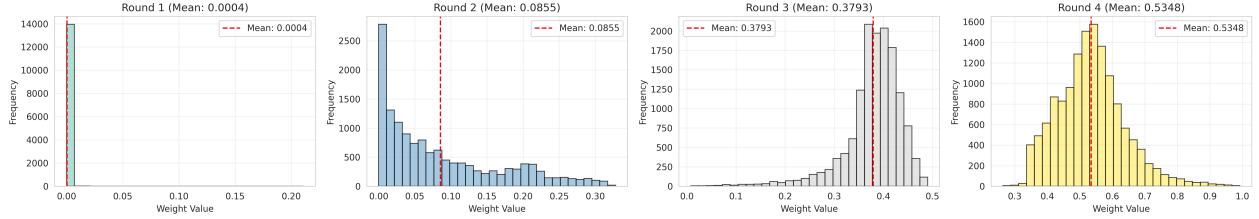


Figure 19 Distribution of the learned ponder weights ($q_\phi(z = t | x)$) for each recurrent step t when the maximum recurrent step $T_m = 4$. These weights were collected during inference on the MMLU benchmark.

recurrent step t when the maximum recurrent step $T_m = 4$, as shown in Figure 19. For convenience, we take the average value of q_ϕ at different recurrent steps and treat it as a normal discrete distribution, resulting in the distribution $\{0.0004, 0.0855, 0.3793, 0.5348\}$. We then substitute this distribution and the T values into the training objective, ignoring the entropy regularization term (after the training stabilizes, it becomes relatively low, for simplicity, we will just ignore it). This leads to a fitting formula, and upon substituting the actual fitting data points N and D , the computed R^2 value is 0.961, with the fitting results illustrated in Figure 20. We can see that the fitting accuracy is high, and the predicted curve closely matches the actual curve, indicating that, under a relatively rough estimate, step-wise loss can be transformed into total loss, thus indirectly suggesting an intrinsic connection between the two.

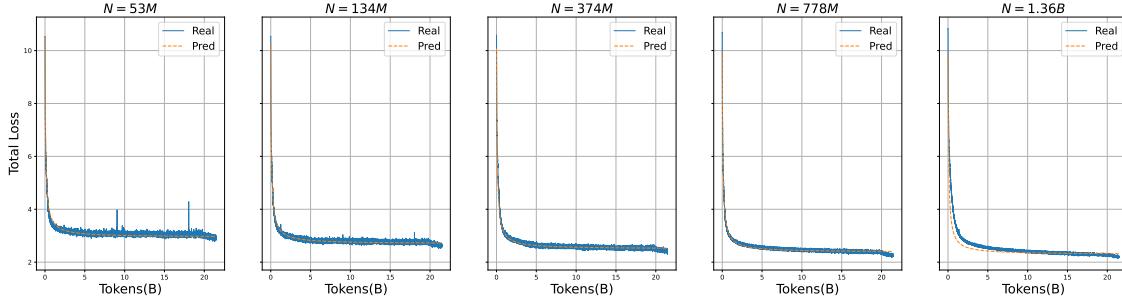


Figure 20 Illustration of the actual loss curve and the loss curve predicted by the estimated Scaling Law when the maximum recurrent step is equal to 4.

E Details of the Scaling Law for LoopLM

E.1 Generalizability for the Total Loss Scaling Law

To demonstrate the generalizability of the Total Loss Scaling Law across model size, training data, and maximum recurrent step, we have conducted relevant experiments. Our evaluation metric is the coefficient of determination R^2 . To evaluate the fitting effectiveness of the Scaling Law, we calculate the coefficient of determination of all data points.

Model Size Generalizability For model size generalizability, our total data points include five different model sizes: 53M, 134M, 374M, 778M, and 1.364B. We select three model sizes as fitting data points, resulting in $\binom{5}{3} = 10$ possible combinations. After fitting, the average R^2 across the 10 combinations is 0.9542, which is similar to the result obtained with the full data points, demonstrating the model size generalizability of the Total Loss Scaling Law. Figure 21 illustrates an example.

Training Data Generalizability Regarding the training data size, we are primarily interested in whether the Scaling Law can predict model performance as training data increases. Therefore, we typically use the preceding data points to predict future data points. To align with this starting point, we have conducted three sets of experiments, using the current 25%, 50%, and 75% of the data points as fitting data to predict the overall fitting performance. The R^2 values for using the first 25%, 50%, and 75% of the data as fitting

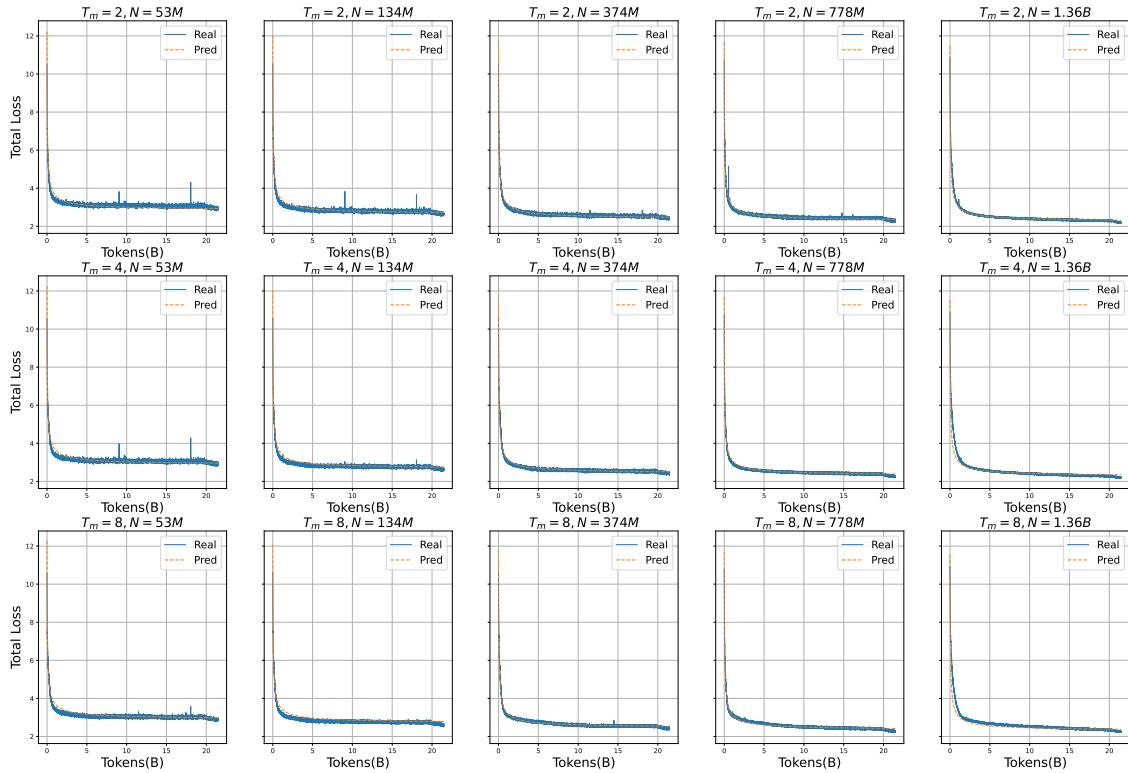


Figure 21 Illustration of model size generalizability for the Total Loss Scaling Law. The fitting data includes model sizes of 374M, 778M, and 1.364B. The predicted curves for the unseen model sizes of 53M and 134M closely align with the actual curves, demonstrating the generalizability of the Total Loss Scaling Law with respect to model size.

points are 0.9385, 0.9609, and 0.962, respectively. It is evident that as the number of data points increases, the consistency between the fitted curves and the actual curves improves. In other words, if you want to predict model performance at larger training sizes, collecting data points closer to those of larger model sizes will yield better prediction results.

Max Recurrent Step Generalizability We have conducted a total of three different maximum recurrent steps: 2, 4, and 8. To verify the generalizability with respect to maximum recurrent step, we select two of these as fitting data points and perform the fitting, followed by validation on the full data points and calculation of R^2 . The average R^2 for the three sets of experiments is 0.9581, demonstrating the generalizability of the Total Loss Scaling Law with respect to maximum recurrent step.

E.2 Generalizability for the Step-wise Loss Scaling Law

Following the same approach as in Section E.1, we seek to explore the performance of the Scaling Law on unseen data points, specifically regarding the generalizability of the Scaling Law. In this subsection, we explore the generalizability of the Step-wise Loss Scaling Law from three aspects: model size generalizability, training data generalizability, and recurrent step generalizability. The evaluation metric remains the coefficient of determination R^2 . To evaluate performance on unseen data points, we will calculate the coefficient of determination using all data points, while fitting will only use a subset of the data points.

Model Size Generalizability The Scaling Law experiments include five different model sizes: 53M, 134M, 374M, 778M, and 1.364B. To verify the generalizability of model size, we select three of these as fitting data points. In each fitting experiment, the Scaling Law does not have access to the remaining two model size data points during fitting, ensuring the reasonableness and validity of the results through repeated experiments. Specifically, to save on resources, we have conducted experiments only for max recurrent step of 2 and 4, resulting in a total of $\binom{5}{3} \times 2 = 20$ small experiments. The final experimental results show that for max recurrent step of 2, the average R^2 value is 0.8815, while for max recurrent step of 4, the average R^2 value is 0.797. This difference is not significant compared to the R^2 values obtained from the full data points (0.8898 and 0.8146), demonstrating the generalizability of the Step-wise Loss Scaling Law with respect to model size. To illustrate the results more clearly, we show example fitting curves in Figure 22. It is important to note that using only a subset of data points for fitting may lead to miscalculations of the Scaling Law on unseen data points. Due to the nature of the power law, if the values are too small, it may result in a very large computed value, causing inaccuracies. To ensure the validity of the fitting, we can attempt to adjust the initial fitting values or impose some constraints on the fitting algorithm. For convenience, we adjust the initial fitting values to make the fitting formula effective over a broader range of data points.

Training Data Generalizability Following Section E.1, to ensure the validity of the fitting, we have selected the first 25%, 50%, and 75% of the data points for fitting. In the case of max recurrent step of 2, the R^2 values are 0.8686, 0.8882, and 0.8896, respectively. For max recurrent step of 4, the R^2 values are 0.793, 0.813, and 0.8142. It can be observed that as the number of fitting data points increases, the fitting accuracy improves. This aligns with the intuition that fitting with more data points generally yields better results. Additionally, these results are similar to those obtained from fitting with the full data points (0.8898 and 0.8146), demonstrating the generalizability of the Step-wise Loss Scaling Law with respect to training data.

Recurrent Step Generalizability In the case of max recurrent step equal to 2, there are only two recurrent step values, making it unreasonable to conduct generalizability experiments. Therefore, we choose to perform experiments with max recurrent step equal to 4. In this situation, we have four different recurrent step values: 1, 2, 3, and 4. We randomly select three of these as fitting data points, resulting in a total of $\binom{4}{3} = 4$ experiments. The average R^2 value obtained from these four experiments is 0.8118, which is similar to the R^2 value of 0.8146 obtained from the full data points, demonstrating the generalizability of the Step-wise Loss Scaling Law with respect to recurrent step. Figure 23 presents a specific example, showing a high degree of consistency between the fitted curve and the actual curve.

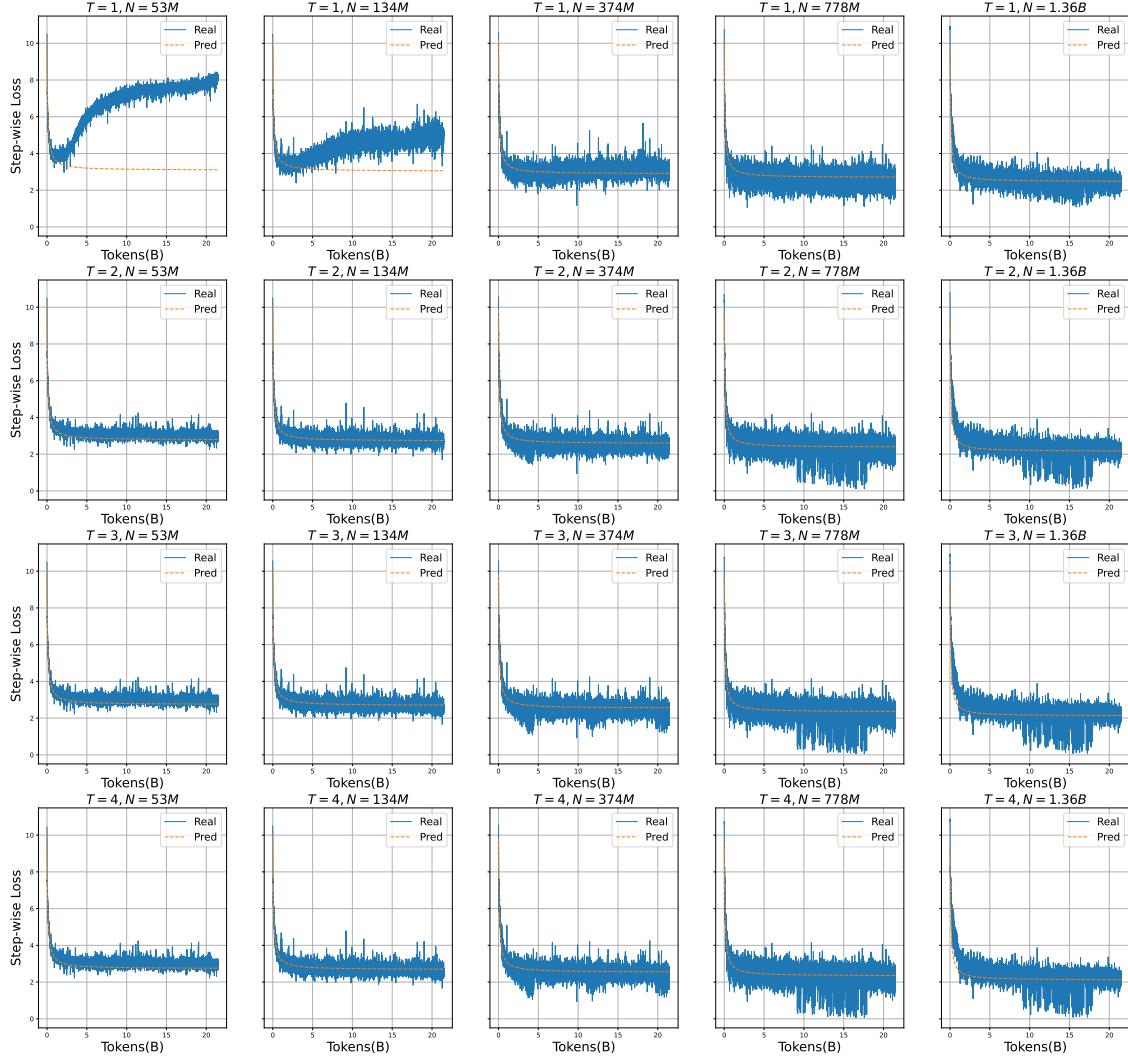


Figure 22 Illustration of model size generalizability for the Step-wise Loss Scaling Law. The fitting data comprises three medium model sizes: 134M, 374M, and 778M. To verify the fitting consistency of the model on unseen larger model size 1.364B and unseen smaller model size 53M, we can observe that the predicted curves reflect the trends of the actual data points, demonstrating the generalizability of the Step-wise Loss Scaling Law with respect to the model size.

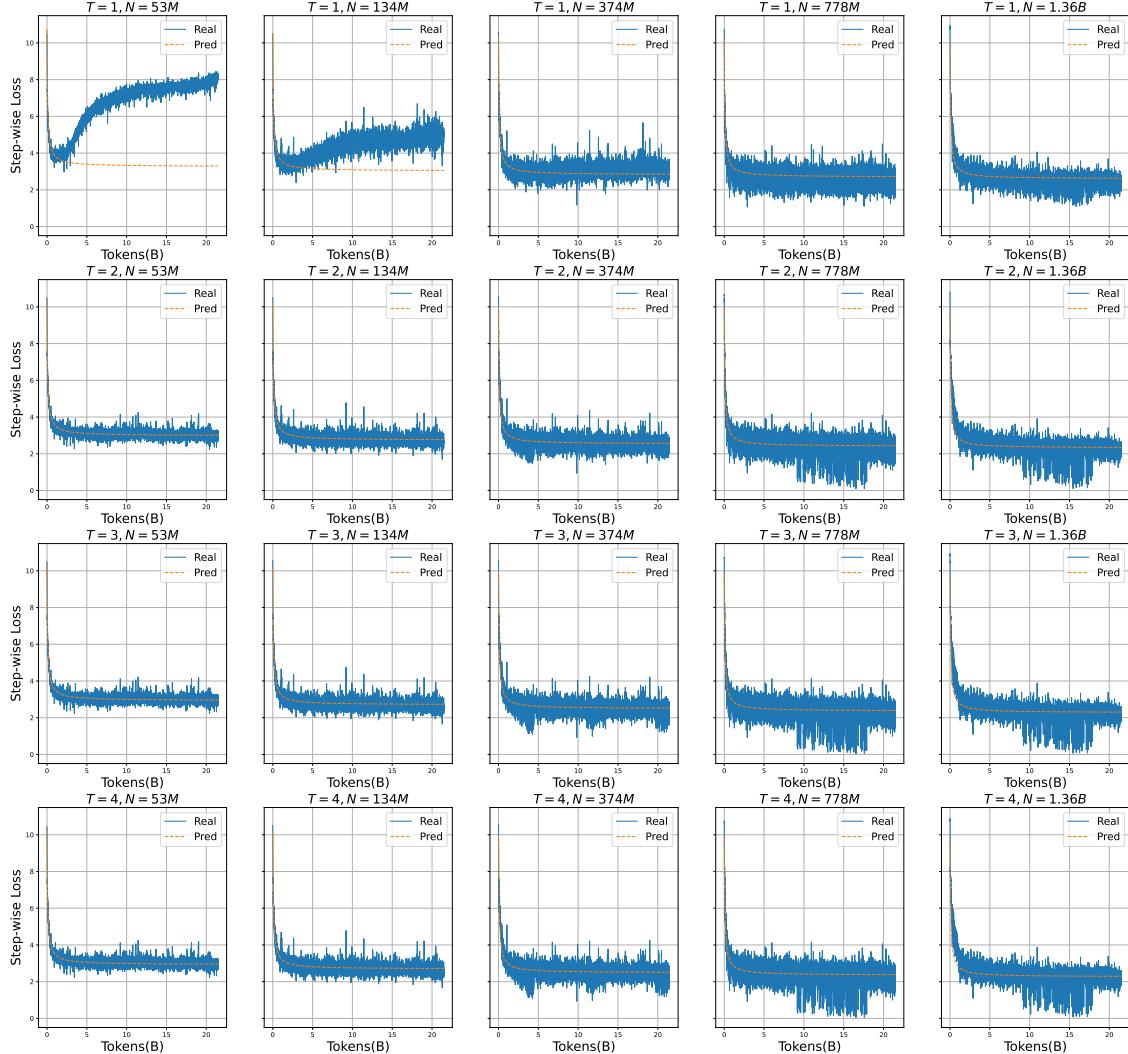


Figure 23 Illustration of recurrent step generalizability for the Step-wise Loss Scaling Law. The fitting data includes three different recurrent steps: recurrent step = 1, 2, and 3. At the unseen data points of recurrent step = 4, the predicted curve closely matches the actual curve, demonstrating the generalizability of the Step-wise Loss Scaling Law with respect to recurrent step.