PARCAE: A DYNAMICAL SYSTEMS PERSPECTIVE TO STABLE LOOPED LLMS

Hayden Prairie[1,2], **Zachary Novack**[1], **Taylor Berg-Kirkpatrick**[1], **& Daniel Y. Fu**[1,2]
[1] University of California, San Diego
[2] Together AI
{hprairie,znovack,tberg,danfu}@ucsd.edu

ABSTRACT

Traditional fixed-depth architectures scale performance by increasing training FLOPs, typically through expanded parameterization at the expense of higher memory footprints. Recent work has suggested a promising alternative–*looped* architectures–which increases FLOPs while keeping parameters constant by looping layers in the model. While initial results are promising, existing recipes for training looped architectures can be unstable, suffering from residual explosion and loss spikes. To address these challenges, we recast layer looping as a nonlinear time-variant dynamical system. Linearizing this system, we observe strict theoretical conditions for stability on the parameters of the system. With these conditions in mind, we propose ***Parcae***, a novel looped Transformer that enforces stability on the looped layers. When trained on the same data, Parcae converges up to 1.25x faster than prior looped models, while reducing downstream perplexity by up to 27% and 64% in comparison to similarly-sized prior looped models and transformers, respectively. Finally, we investigate recurrence as a medium to increase FLOPs, observing a new scaling axis in training and providing a means of scaling test-time compute and improving quality with constant memory overhead.

1 INTRODUCTION

Scaling laws have established that model performance improves predictably with increased FLOPs Hoffmann et al. (2022), typically by increasing parameterization or training data. As a result, parameter counts of state-of-the-art models have grown in an effort to increase scale along with training data. This shift has inflated the memory footprint required to deploy these models, limiting their accessibility in resource-constrained environments Frantar et al. (2023).

Recently, a promising alternative has emerged: layer-looped models such as recurrent depth models or looped transformers (Geiping et al., 2025; Zhu et al., 2025; Dehghani et al., 2019).[1] These looped models scale FLOPs *without* increasing parameterization by iteratively looping layers within a model (e.g., a looped model could achieve an effective depth of ten by looping one layer ten times). Initial results are promising, with looped models matching the quality of multi-billion-parameter systems while maintaining a sub-billion-parameter memory footprint (Geiping et al., 2025; Wu et al., 2025). Moreover, they show potential for latent reasoning (Schwarzschild et al., 2021; Yang et al., 2024; Xu & Sato, 2025) and improved mechanistic interpretability of depth (Csordás et al., 2025; Lu et al., 2025).

Unfortunately, prior research Geiping et al. (2025); McLeish et al. (2025); Anonymous (2026) and our work observe these models' training to be unstable, finding residual state explosion during training. Since these models loop the layers of complex non-linear architectures (e.g., transformer blocks), the source of instability in looped models can be difficult to understand analytically. As a result, training requires sensitive hyperparameter selection and residual normalization (e.g., Post-Norm) to correct. Furthermore, even in marginally stable training runs, we still observe loss spikes from looped models' doubly stochastic training objective.

To elucidate instability, we propose viewing layer looping as a nonlinear time-variant dynamical system:

$$h_{t+1} = \overline{\boldsymbol{A}} h_t + \overline{\boldsymbol{B}} e + \overline{\mathcal{R}}(h_t, e), \tag{1}$$

[1] We use the term recurrent depth models synonymously with looped transformers or universal transformers.

where for an input e, the evolution of a hidden state h across the depth of an architecture is modeled using linear operators $\overline{\boldsymbol{A}}$ and $\overline{\boldsymbol{B}}$ and a non-linear operator $\overline{\mathcal{R}}$. By linearizing this framework, we can use classic control theory results to infer stability conditions on the eigenvalues of $\overline{\boldsymbol{A}}$. We observe that prior work does not satisfy these stability conditions, which we hypothesize to contribute to their instability.

To address instability architecturally, we propose ***Parcae***, a novel looped transformer that enforces the stability conditions of our dynamical systems. ***Parcae*** explicitly discretizes a continuous representation of Equation (1) in conjunction with constraining the eigenvalues of $\overline{\boldsymbol{A}}$, to automatically perform state normalization in looped layers. Furthermore, to correct loss spikes induced by looped models' doubly stochastic training objective, we propose a per-sequence depth algorithm to reduce gradient variance and loss spikes.

We evaluate the quality, convergence speed, and stability of Parcae. To assess quality and convergence speed, we train at 100M and 350M scales, finding that Parcae reduces downstream perplexity by up to 27% and 64% for comparably-sized RDMs and standard transformers, respectively, while additionally converging up to 1.25x faster than prior RDMs. For stability, we analyze the hyperparameter sensitivity and convergence of Parcae with prior RDMs, further observing that per-sequence depth effectively mitigates loss spikes.

Beyond evaluation comparison, we investigate the generalization of recurrence as a medium to scale training and inference FLOPs. When scaling recurrences in training, we observe monotonic improvements in validation loss, implying recurrence as an orthogonal scaling axis to data and parameters. At test time, recurrence can scale model performance *without increasing the KV-cache size*. This can be achieved with a negligible perplexity drop by only utilizing the final recurrence's KV-cache, enabling a constant memory footprint. Our core contributions are as follows:

- **Dynamical Reinterpretation of Depth:** We introduce a novel formulation of looped models as a discretization of a nonlinear time-variant dynamical system. Linearizing this framework provides explicit conditions for stability, which we use to derive Parcae.
- **Fast, Quality, and Stable Training at Scale:** At 100M and 350M scales, Parcae achieves significant relative improvements in downstream quality, reducing perplexity by up to 27% and 64% over comparatively-sized prior RDMs and transformers, while converging up to 1.25x faster than prior RDMs. We further demonstrate that Parcae reduces instability and hyperparameter sensitivity.
- **Scaling FLOPs both in Training and Inference:** We show that recurrence exhibits a new axis of scaling FLOPs in training, orthogonal to parameter and data. At inference, we observe that Parcae can scale test-time compute through extended recurrence, while utilizing a ***single*** KV cache for all recurrences (with only a 0.01 perplexity drop), enabling memory-efficient test-time scaling.

2 Background

We provide a brief background on recurrent depth models (Section 2.1) and LTI systems (Section 2.2), which are essential to derive stable looped transformers. We include an extended literature review in Section B.

2.1 Looped Models

To understand how looped models can increase capacity without increasing the parameter count within language modeling (Dehghani et al., 2019; Zhu et al., 2025; Geiping et al., 2025; McLeish et al., 2025; Bae et al., 2025) and small-scale algorithmic problems (Schwarzschild et al., 2021; Yang et al., 2024; Bansal et al., 2022; Wang et al., 2025; Jolicoeur-Martineau, 2025), we will focus on middle-looping Saunshi et al. (2025) recurrent-depth Geiping et al. (2025); McLeish et al. (2025) and looped transformers Yang et al. (2024). Specifically, middle-looped recurrent depth architecture contains three units: an initial *prelude* unit $\mathcal{P}$, a middle *recurrent* unit $\mathcal{R}$, and a final *coda* unit $\mathcal{C}$. Formally, given an input $s \in V^n$, where V is vocabulary and n is sequence dimension, the outputs $p \in \mathbb{R}^{n \times |V|}$ can be computed by the following update rule:

$$e = \mathcal{P}(s) \quad h_{t+1} = \mathcal{R}(h_t, e) \quad p = \mathcal{C}(h_T) \tag{2}$$

where $h_0 \sim \mathcal{N}(0, \sigma^2 I_{d \times d})$ and d the embedding dimension. Intuitively, $\mathcal{P}$ embeds inputs into the latent space, conditioning $\mathcal{R}$ as it recursively updates the hidden state $h_t \in \mathbb{R}^{n \times d}$ for T iterations, which $\mathcal{C}$ uses to generate p. Within $\mathcal{R}$, prior work injected e using addition $h_{t+1} = \mathcal{R}(h_t + e)$ (Yang et al., 2024) or concatenation with projection $h_{t+1} = \mathcal{R}(W[h_t; e])$ (Geiping et al., 2025), where $W \in \mathbb{R}^{d \times 2d}$.

Although looped models can be viewed as a traditional model with weight-shared layers, the modern looped models, which we focus on, allow for variable depth. During training, a depth of T is sampled

for each batch using the distribution Λ (e.g., Poisson) with a mean of μ_{rec}, variable depth which imbues the looped model with unique properties (e.g., adaptive computation, over-extrapolation) Schwarzschild et al. (2021); Geiping et al. (2025), which are not empirically present in simple weight-shared models Yang et al. (2024). The training objective thus minimizes the expectation over the dataset $\mathcal{D}$ and Λ:

$$\theta^\star = \underset{\theta}{\operatorname{argmin}} \mathbb{E}_{(x,y)\sim\mathcal{D},T\sim\Lambda}\left[\ell\big(f_\theta(x;T),y\big)\right] \tag{3}$$

Finally, rather than computing the backward pass through all recurrences, truncated backpropagation in depth can be used, similar to backpropagation through time in RNNs Hinton & Sutskever (2013), and is often kept as a constant μ_{bwd} Geiping et al. (2025). To perform this, prior work truncates the sampling distribution Λ to a mean $\mu_{\text{rec}} - \mu_{\text{bwd}}$ (Geiping et al., 2025) . Thus, per step, T recurrences are used in the forward pass, where $T \sim \Lambda(\mu_{\text{rec}} - \mu_{\text{bwd}})$, and gradients are propagated μ_{bwd} recurrences in the backward pass.

While looped models are not limited to transformers Schwarzschild et al. (2021), we focus on transformer implementations by prior work Geiping et al. (2025); Yang et al. (2024). Specifically, Geiping et al. (2025) found looped models to be unstable to train at scale and thus adopted a transformer block pattern that explicitly normalizes the residual, combining both Pre-Norm and Post-Norm with the following update rule:

$$\bar{x}^{(\ell)} = \text{LN}(\text{MHA}(\text{LN}(x^{(\ell-1)})) + x^{(\ell-1)}) \tag{4}$$

$$x^{(\ell)} = \text{LN}(\text{FFN}(\text{LN}(\bar{x}^{(\ell)})) + \bar{x}^{(\ell)}) \tag{5}$$

where $\text{LN}(\cdot)$ denotes layer normalization, $\text{MHA}(\cdot)$ multi-head attention, and $\text{FFN}(\cdot)$ feed-forward networks. We later show that residual normalization is unnecessary if stability is properly controlled.

2.2 Linear Time-Invariant Dynamical Systems

To study the instability of looped models, we will use a linear-time-invariant (LTI) dynamical system as a tractable linear surrogate for complex non-linear looped models. In classical control theory, LTI systems are formalized through first-order differential equations that describe the evolution of a hidden state $h(t) \in \mathbb{R}^{d_h}$ given an input signal $e(t) \in \mathbb{R}^{d_e}$:

$$\dot{h}(t) = \boldsymbol{A}h(t) + \boldsymbol{B}e(t), \qquad y(t) = \boldsymbol{C}h(t) \tag{6a}$$

$$h_t = \overline{\boldsymbol{A}}h_{t-1} + \overline{\boldsymbol{B}}e_t, \qquad y_t = \boldsymbol{C}h_t \tag{6b}$$

where $\boldsymbol{A} \in \mathbb{R}^{d_h \times d_h}$ governs the dynamics, $\boldsymbol{B} \in \mathbb{R}^{d_h \times d_e}$ controls how external inputs influence the state, and $\boldsymbol{C} \in \mathbb{R}^{d_e \times d_h}$ projects the hidden state to the output $y(t) \in \mathbb{R}^{d_e}$. The continuous-time system Equation (6a) can be discretized to obtain Equation (6b) using a step size Δ; for instance, zero-order hold (ZOH) can be used, yielding the following parameterization $\overline{\boldsymbol{A}} = \exp(\Delta\boldsymbol{A})$ and $\overline{\boldsymbol{B}} = (\Delta\boldsymbol{A})^{-1}(\exp(\Delta\boldsymbol{A}) - I) \cdot \Delta\boldsymbol{B}$. LTI systems can be classified into three states: *stable*, *marginally stable*, and *unstable*. Stable LTI systems remain bounded as $t \to \infty$, and will converge to a steady state even with a disturbance such as noise. Marginally-stable LTI systems neither converge nor explode as $t \to \infty$, behaving with oscillatory dynamics. Finally, unstable LTI systems grow unboundedly as $t \to \infty$, and thus diverge from equilibrium points.

A fundamental property of LTI systems is that their *stability* is determined by the eigenvalues of $\boldsymbol{A}$. For continuous LTI systems, stability requires eigenvalues of $\boldsymbol{A}$ to be negative; for discrete LTI systems, stability requires that eigenvalues of $\overline{\boldsymbol{A}}$ lie within the unit circle Desoer & Wu (1968) (i.e., $\rho(\overline{\boldsymbol{A}}) < 1$ where ρ computes the spectral norm of A), with marginally-stable systems having the eigenvalues of $\overline{\boldsymbol{A}}$ lie on the unit circle (i.e., $\rho(\overline{\boldsymbol{A}}) = 1$), and unstable systems having eigenvalues of $\overline{\boldsymbol{A}}$ lie outside the unit circle (i.e., $\rho(\overline{\boldsymbol{A}}) > 1$). These conditions for stable LTI systems ensure that the hidden state remains bounded over time in both the continuous and discrete representations. LTI systems have been extensively used in sequence modeling as sub-quadratic alternatives (Gu et al., 2022; Gu & Dao, 2024; Dao & Gu, 2024), leveraging structured parameterizations of $\boldsymbol{A}$ to ensure stability across the sequence dimension.

3 Understanding Instability in Looped Architectures

While prior work has observed general difficulties in training looped architecture stably Geiping et al. (2025); McLeish et al. (2025), through empirical analysis, we more formally characterize that instability stems from unconstrained state explosion (see Figure 1; Section F). To remedy this, prior work often utilized residual normalization (e.g., PostNorm) Geiping et al. (2025); McLeish et al. (2025); however,

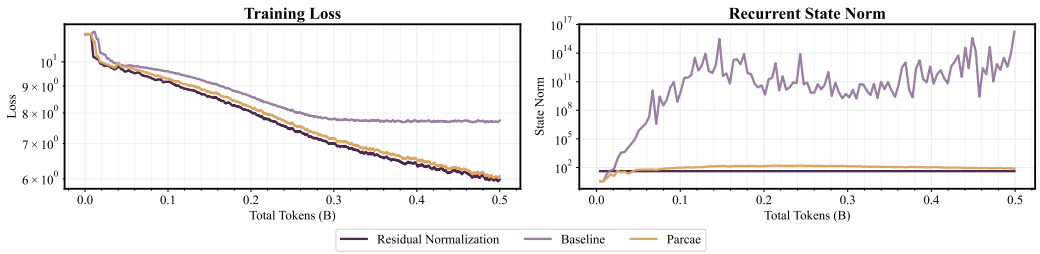

Figure 1: **Training Instability of Looped Architectures.** In the left plot, we observe divergent training loss of baseline Pre-Norm recurrent depth models, while training with residual normalization and Parcae are convergent. From the right plot, it can be observed that instability stems from an exploding recurrent state norm, the norm of the hidden embedding after T recurrences (i.e., $||h_T||_2$).

residual normalization is known to induce divergent training in fixed-depth transformers Xiong et al. (2020), a property we empirically observe to hold in looped models (see Table 1). While reasoning about the complex, non-linear structure of looped models may be difficult, we show that there is a simple LTI framework to easily reason about their instability and state-explosion (Section 3.1). We further utilize this framework to elucidate the source of instability in prior looped models (Section 3.2), observing that state explosion can be directly linked to unstable parameterizations learned in training (see Figure 2).

3.1 Viewing Depth as a Dynamical System

Our key insight comes from recasting the forward pass of a neural network as a dynamical system itself. Consider a simplified transformer-based looped model defined in Section 2.1 for language modeling, in which the prelude block $\mathcal{P}$ is only an embedding layer, which maps tokens a sequence of tokens $s \in V^n$ into embedding space $e \in \mathbb{R}^{n \times d_h}$, the coda block $\mathcal{C}$ is only a projection head, which maps the final latents into probability space though a linear projection head $g: d_h \rightarrow |V|$ with normalization (i.e., $p = \mathcal{C}(h_T) = g(\text{LN}(h_T))$), and the recurrent block $\mathcal{R}$ is parameterized with L transformer blocks. While one could come up with several methods of input injection that condition $\mathcal{R}$ on e, building on prior work , we focus on linear methods of injection (e.g., $\mathcal{R}(h_t, e) = \mathcal{R}(W_1 h_t + W_2 e)$, where $W_1 \in \mathbb{R}^{d_h \times d_h}$ and $W_2 \in \mathbb{R}^{d_h \times d_e}$ are linear projectors).[2]

Recall in Section 2.1 that $\mathcal{R}$ denotes the exact output of our recurrent system, that is, $h_{t+1} = \mathcal{R}(h_t, e)$, folding in all operations including residual connections in standard transformers. A common interpretation of a fixed-depth Pre-Norm transformer views the residual stream as a communication channel where the final outputs h_T are the sum of the relative outputs of all previous layers and the original embedding . To recast looped models this way, let $\overline{\mathcal{R}}$ be the *relative contribution* of nonlinear operations in the recurrent unit, ignoring the residual connection with respect to the input (e.g., $\overline{\mathcal{R}}(W_1 h_t + W_2 e) = \mathcal{R}(W_1 h_t + W_2 e) - (W_1 h_t + W_2 e)$). In doing so, it can then be observed that our recurrent update rule is thus

$$h_{t+1} = W_1 h_t + W_2 e + \overline{\mathcal{R}}(h_t, e) \tag{7}$$

where we use $\overline{\mathcal{R}}(h_t, e) = \overline{\mathcal{R}}(W_1 h_t + W_2 e)$ for brevity. While this system is non-trivial to reason about, as $\overline{\mathcal{R}}$ is composed of non-linear attention and feed-forward blocks, we can first start by noticing that the recurrent update can be exactly formulated as a *non-linear time-variant dynamical system* of the form:

$$\dot{h}(t) = \boldsymbol{A} h(t) + \boldsymbol{B} e + \overline{\mathcal{R}}(h(t), e), \qquad y(t) = \boldsymbol{C} h(t) \tag{8a}$$

$$h_t = \overline{\boldsymbol{A}} h_{t-1} + \overline{\boldsymbol{B}} e + \overline{\mathcal{R}}(h_{t-1}, e), \qquad y_t = \boldsymbol{C} h_t \tag{8b}$$

where $\overline{\boldsymbol{A}} = W_1$, $\overline{\boldsymbol{B}} = W_2$, and $\boldsymbol{C} \in R^{d_c \times d_h}$ is an output projection decoupling the embedding dimension of $\mathcal{C}$ and $\mathcal{R}$ (i.e. $p = \mathcal{C}(\boldsymbol{C}(h_T))$). Though this formulation does not immediately elucidate instability, a linearization of this system (i.e., dropping the nonlinear term $\overline{\mathcal{R}}$) resembles a discrete LTI system of the form

$$h_{t+1} = \overline{\boldsymbol{A}} h_t + \overline{\boldsymbol{B}} e \tag{9}$$

To maintain the stability of this system, the parameterization of $\overline{\boldsymbol{A}}$ and $\overline{\boldsymbol{B}}$ should be constrained to *prevent state explosion and enforce stability* following Section 2.2. While an oversimplification of a nonlinear looped transformer, we posit that the theoretical conditions for stability in the case must also be satisfied

[2]Note that both prior forms of input injection, addition Yang et al. (2024) and concatenation Geiping et al. (2025), fall under this framework.

LR	Baseline	Res. Norm	Parcae
2e-4	✓	✓	✓
4e-4	✗	✓	✓
6e-4	✗	✗	✓
8e-4	✗	✗	✓
1e-3	✗	✗	✓

Table 1: **Hyperparameter Instability.** Convergence across learning rates for Pre-Norm RDMs, Residual-Normalized RDMs, and Parcae. We observe that Parcae is less susceptible to hyperparameter selection, aligning with our dynamical systems framework.

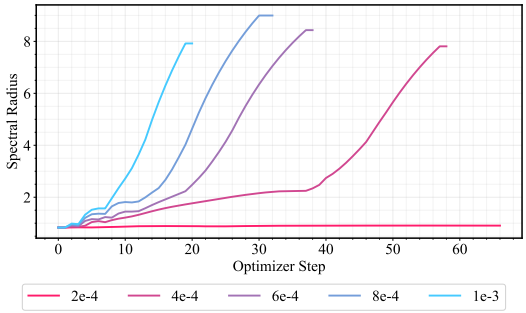

Figure 2: **Spectral Radius of Unconstrained $\overline{\boldsymbol{A}}$.** For a baseline Pre-Norm RDM, we plot the spectral radius of training across different learning rates, finding that convergent runs never learn a spectral radius greater than 1 for $\overline{\boldsymbol{A}}$. The state explosion, in Figure 1 can thus directly be linked to the growing spectral radius of $\overline{\boldsymbol{A}}$.

Method	Update Equation	Stability
Addition Injection Yang et al. (2024)	$h_{t+1} = Ih_t + Ie + \overline{\mathcal{R}}(h_t, e)$	*marginally-stable*
Concatenation Injection Geiping et al. (2025)	$h_{t+1} = W_1 h_t + W_2 e + \overline{\mathcal{R}}(h_t, e)$	*unstable*
Parcae (ours)	$h_{t+1} = \overline{\boldsymbol{A}} h_t + \overline{\boldsymbol{B}} e + \overline{\mathcal{R}}(h_t, e)$	*stable*

Table 2: **Comparison of Prior Update Rules.** Overview of current methods of input injection and looped update rules. We categorize the stability of prior systems based on the analysis performed in Section 3.2, along with our derivation of a stable recurrent update rule for Parcae in Section 4.1.

in the nonlinear case.[3] We will later show in Section 3.2 that, empirically, a significant source of state explosion in Pre-Norm based looped models stems from an unconstrained $\overline{\boldsymbol{A}}$ and not the non-linear unit $\overline{\mathcal{R}}$.[4]

3.2 Instability of Prior Recurrent Depth Models

Using our dynamical systems framework, the instability of prior looped architectures becomes obvious. In particular, we will take the model's *linearized forms* (e.g., ignoring the nonlinear block $\overline{\mathcal{R}}$) as a lens to analyze their stability. First, recall that in Section 2.2, if $\rho(\overline{\boldsymbol{A}}) \geq 1$, our linear system is unstable. Using the *relative contribution* representation of looped models in Section 3.1, we recast prior mediums of input injection in Section 2.1 into our framework (see Table 2). Specifically, for Pre-Norm looped models using addition as injection Yang et al. (2024), the dynamical systems update rule can be written as $h_{t+1} = Ih_t + Ie + \overline{\mathcal{R}}(Ih_t + Ie)$. When linearized, $\overline{\boldsymbol{A}} = I$, meaning the model would be *marginally-stable* system. Alternatively, the update rule for Pre-Norm looped models using concatenation as injection Geiping et al. (2025) can be rewritten as $h_{t+1} = W[h_t; e] + \overline{\mathcal{R}}(W[h_t; e]) = W_1 h_t + W_2 e + \overline{\mathcal{R}}(W_1 h_t + W_2 e)$. Here $\overline{\boldsymbol{A}} = W_1$ is unbounded and can create state explosion if not carefully maintained during training.

This instability does not only exist theoretically: empirically, we plot the spectral radius of $\overline{\boldsymbol{A}}$ over the training of baseline Pre-Norm looped model with concatenation (i.e., W_1) Geiping et al. (2025) in Figure 2. During training, we observe that divergent runs (i.e., the runs with state explosion in Figure 1 and Section F) learn $\rho(\overline{\boldsymbol{A}}) \geq 1$, while convergent runs always maintain $\rho(\overline{\boldsymbol{A}}) < 1$, confirming that our intuition in the LTI case exactly describes instability in the nonlinear time-variant case. The phenomenon is perhaps not all that surprising. In a simple case using $T = 32$ recurrences, if the $\rho(\overline{\boldsymbol{A}}) = k > 1$, then the output state will include the initial noise magnified by k^{32}, dwarfing the relative contribution of the nonlinear blocks and leading to numerical overflow.[5] This theoretical and empirical analysis emphasizes the importance of carefully parameterizing $\overline{\boldsymbol{A}}$ for stable model training, which we will use to build our stable looped architecture in the next section.

[3] While not guaranteed for time-variant systems, prior work on linear time-variant dynamical systems for sequence modeling still found stable parameterizations of $\boldsymbol{A}$ to be important Gu & Dao (2024); Dao & Gu (2024).

[4] State explosion induced by $\overline{\mathcal{R}}$ is still an issue for concern; however, methods such as weight decay can prevent it.

[5] We note that this analysis only applies to Pre-Norm looped models, as looped models using residual normalization Geiping et al. (2025) (e.g., Post-Norm) do not fall under our dynamical systems framework and instead likely suffer from gradient vanishing issues Xiong et al. (2020).

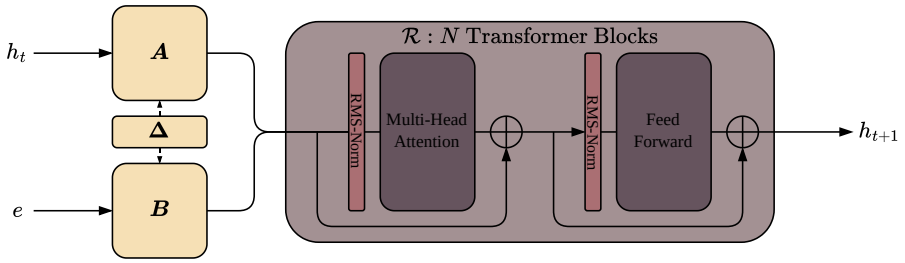

Figure 3: **Parcae Recurrent Block Diagram.** The diagram above describes the computation graph for a single recurrence. Given parameters $\boldsymbol{A}$,$\boldsymbol{B}$,Δ, discretization is used to generate $\overline{\boldsymbol{A}}$ and $\overline{\boldsymbol{B}}$. The final output embedding h_{t+1} given a hidden state h_t and input e, is computed by first passing through $\overline{\boldsymbol{A}}$ and $\overline{\boldsymbol{B}}$ and followed by $\mathcal{R}$, which is N layers of Pre-Norm transformer blocks.

4 PARCAE: A STABLE LOOPED ARCHITECTURE

Using our framework in Section 3.1, we create ***Parcae***, a new looped architecture that explicitly satisfies the stability constraints in Section 4.1. Furthermore, we propose a per-sequence depth sampler to reduce gradient variance and further stabilize Parcae's doubly stochastic training objective (Section 4.2).

4.1 BLOCK DESIGN AND STABLE PARAMETERIZATION

Inspired by our empirical observations in Section 3.2, Parcae is parameterized to explicitly uphold the LTI constraints of our dynamical systems framework in Section 3.1. Specifically, Parcae is parameterized with the continuous formulation of $\boldsymbol{A}$ and $\boldsymbol{B}$, using a learned vector $\Delta \in \mathbb{R}^{d_h}$ to discretize our system [6]. To discretize this system, we use a combination of ZOH and Euler discretizations (i.e., $\overline{\boldsymbol{A}} = \exp(\Delta \boldsymbol{A})$ and $\overline{\boldsymbol{B}} = \Delta \boldsymbol{B}$), which is inspired by work in sequence modeling Gu & Dao (2024); Dao & Gu (2024). While we expect many parameterizations of $\boldsymbol{A}$ to work, given they satisfy the LTI constraints discussed in Section 3.1, we choose to parameterize $\boldsymbol{A}$ as a negative diagonal matrix, which makes matrix exponentiation cheap. To enforce negative eigenvalues of $\boldsymbol{A}$ throughout training, we learn $\boldsymbol{A} = \texttt{Diag}(-\exp(\texttt{log_A}))$, where `log_A` is a learnable d_h-dimensional vector. While many formulations of $\boldsymbol{A}$ would work, ensuring negative eigenvalues in the diagonal case is simple and much cheaper than higher-rank approaches, and leave the exploration of more expressive parameterizations of $\boldsymbol{A}$ to future work. Overall, our update rule becomes:

$$h_{t+1} = \overline{\boldsymbol{A}} h_t + \overline{\boldsymbol{B}} e + \overline{\mathcal{R}}(h_t, e) \tag{10}$$

Finally, we parameterize $\mathcal{R}$ using N standard transformer blocks, each using standard multi-head attention and feed-forward units. However, we note that Parcae is agnostic to how $\mathcal{R}$ is parameterized (e.g., $\mathcal{R}$ could use linear-attention Gu & Dao (2024); Yang et al. (2025) or mixture-of-experts Shazeer et al. (2017); Fedus et al. (2022)), allowing for different design choices. A full forward pass of Parcae can be found in Section E.

4.2 STABLE TRAINING ALGORITHMS FOR PARCAE

Beyond architecture, we observe that we can further reduce instability with better training algorithms. Notably, observe that the training objective in Equation (3) is doubly stochastic, and thus to faithfully model the expectation over Λ, more depths should be sampled per optimizer step. While some work used a per-batch depth Bansal et al. (2022), a per-micro-batch depth is more commonly used with lock-stepping to prevent device stalling (i.e., by synchronizing sampled T across devices) Geiping et al. (2025). Although micro-batch sampling improves our estimation of the expectation of our training objective, lock-stepping harms it, as noted by Geiping et al. (2025). To rectify these challenges, we propose a simple, novel alternative, which samples a per-sequence depth within a micro-batch, while still synchronizing per-sequence depth across devices. Interestingly, these techniques are somewhat similar to layer dropout for fixed-depth models Huang et al. (2016); Hayou & Ayed (2021). A detailed training algorithm is included in Section D.

Furthermore, rather than truncating Λ based on μ_{bwd}, we sample forward recurrence from Λ using only μ_{rec}. Specifically, using a Poisson distribution for Λ we find that extrapolation to lower and higher recurrences can be heavily affected when truncating Λ based on μ_{bwd}, an ablation for which can be found in Section I. Finally, we choose $\mu_{\text{bwd}} = \lceil \frac{\mu_{\text{rec}}}{2} \rceil$ for all of our experiments, an ablation for which can be found in Section J.

[6] With abuse of notation, we let $\Delta \boldsymbol{A} = \Delta \odot \boldsymbol{A}$ (i.e., elementwise multiplication).

Table 3: **PPL Performance.** Perplexity metrics for a validation set Geiping et al. (2025), Wikitext Merity et al. (2016), Pile-10k Nanda (2022), and LAMBADA Paperno et al. (2016) on 100M and 350M scales, using eval recurrence of μ_{rec}. **Bolded** indicates **best** for each parameter scale.

	Model	Val	Wiki.	Pile	Lamb.
100M	Transformer	18.28	91.54	332.01	737.67
	RDM	14.23	63.27	197.73	356.35
	Parcae	**13.59**	**60.33**	**184.73**	**260.27**
350M	Transformer	11.82	48.04	145.23	148.95
	RDM	10.76	41.31	117.24	116.77
	Parcae	**10.09**	**37.53**	**103.78**	**92.31**

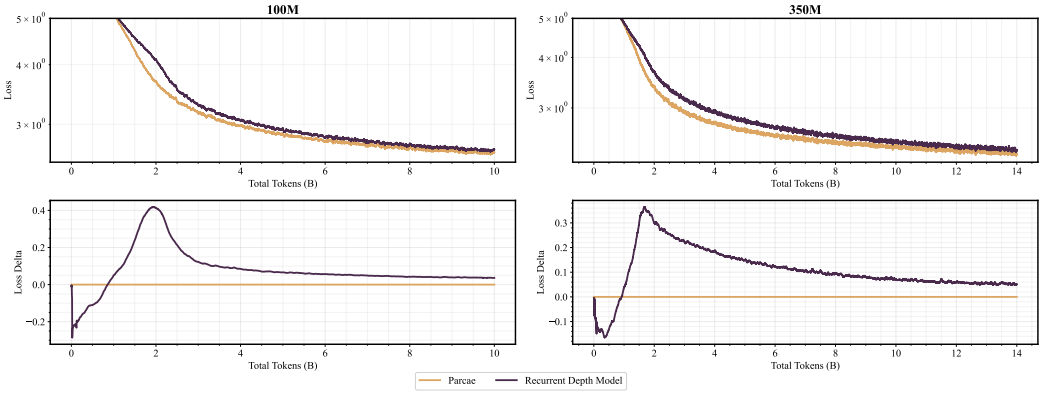

Figure 4: **Convergence Comparison of Parcae versus Baseline RDMs.** Smoothed training loss delta for 100M (left) and 350M (right) models over total tokens. Parcae consistently maintains a lower training loss gap over baseline RDMs Geiping et al. (2025).

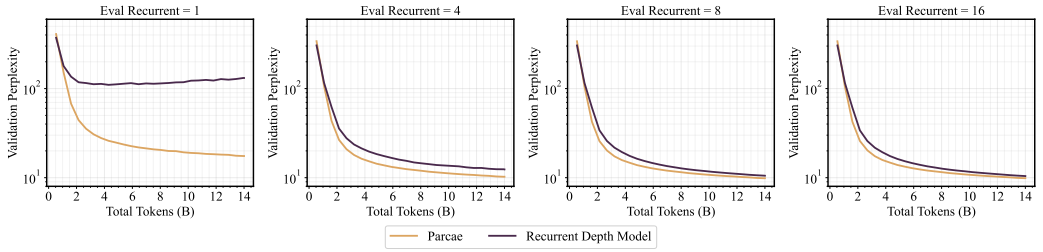

Figure 5: **Validation Perplexity Across Different Depths during Training.** For the 350M scale, we show the perplexity on a held-out validation set Geiping et al. (2025) for Parcae and baseline RDMs at $T = 1,4,8,16$, demonstrating stronger convergence across all recurrences.

5 Benchmarks and Results

We evaluate Parcae on convergence speed, quality, and stability. To assess the convergence speed and quality, we train 100M and 350M models, using prior RDMs Geiping et al. (2025) and standard Transformers as baselines. We use relative loss gaps and validation perplexity to show accelerated convergence over prior RDMs (see Section 5.2), while using downstream perplexity and zero-shot accuracy benchmarks to show the quality of Parcae (see Section 5.3). Finally, to evaluate stability, we compare the hyperparameter sensitivity of Parcae with other looped architectures and further justify our per-sequence sampling strategy (see Section 5.4).

5.1 Setup

We train using the dataset and tokenizer from Geiping et al. (2025). Specifically, we use a Poission distribution for Λ and choose $\mu_{\text{bwd}} = \lfloor \frac{\mu_{\text{rec}}}{2} \rfloor$, an ablation that can be found in Section J. For 100M and 350M models, we use $\mu_{\text{rec}} = 16$ and $\mu_{\text{rec}} = 8$, respectively. Additionally, we tweak the sampling method of Geiping et al. (2025) to decouple μ_{rec} and μ_{bwd} from Λ, an ablation which can be found in Section I. Extended details on both model definitions and hyperparameters can be found in Section K and Section L, respectively.

Table 4: **Zero-Shot Evaluation Results.** A comparison of Parcae and baseline RDMs Geiping et al. (2025) across 100M and 350M parameter scales on Hellaswag Zellers et al. (2019), ARC Clark et al. (2018), PIQA Bisk et al. (2020), WinoGrande Sakaguchi et al. (2021), BoolQ Wang et al. (2019), and SciQ Welbl et al. (2017). We additionally compare against fixed-depth Transformers with exact parameterizations of Parcae excluding $\boldsymbol{A}$,$\boldsymbol{B}$,$\boldsymbol{C}$,Δ. **Bolded**/underlined results indicate **best**/second-best for each parameter scale.

	Model	T	Hellaswag	ARC-c	ARC-e	PIQA	WinoGrande	BoolQ	SciQ	Avg
100M	Transformer	–	26.56	18.00	40.57	58.00	50.75	41.38	62.30	42.51
	RDM ($\mu_{\text{rec}}=16$)	1	26.57	17.45	36.41	56.80	50.67	37.77	51.90	39.65
		4	26.96	17.32	40.40	57.45	50.04	60.21	62.90	45.04
		16	27.16	17.66	42.38	59.14	49.25	51.35	**72.50**	45.63
	Parcae ($\mu_{\text{rec}}=16$)	1	26.94	18.43	32.79	56.31	48.15	48.56	57.80	41.28
		4	27.04	**18.86**	39.98	58.11	49.01	54.80	70.80	45.51
		16	**27.18**	18.09	**43.10**	**59.30**	**50.75**	**61.83**	71.50	**47.39**
350M	Transformer	–	28.15	19.88	44.91	60.83	51.62	57.55	76.00	48.42
	RDM ($\mu_{\text{rec}}=8$)	1	26.70	18.52	34.43	56.58	49.41	37.95	42.70	38.04
		4	28.37	18.86	46.04	61.48	49.64	**61.93**	74.40	48.67
		8	28.55	20.90	47.26	61.75	49.88	61.53	76.70	49.51
	Parcae ($\mu_{\text{rec}}=8$)	1	28.06	18.09	44.02	60.45	48.70	61.90	73.40	47.80
		4	28.78	20.82	**49.07**	61.43	**51.22**	59.27	**79.00**	49.94
		8	**29.23**	**21.08**	48.78	**62.08**	50.75	60.73	78.80	**50.21**

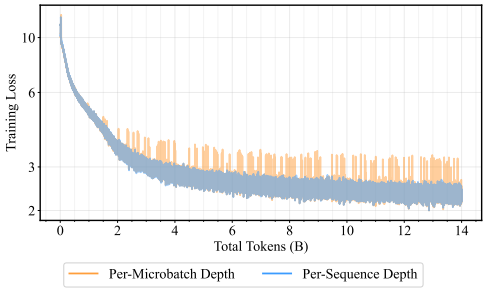

Figure 6: **Per-Sequence Sampling Mitigates Loss Spikes**. Observe no loss spikes with per-sequence over per-batch sampling, indicating stable training dynamics.

Table 5: **Per-Microbatch vs. Per-Sequence Depth Comparison**. We compare perplexity of Parcae models trained with per-batch depth sampling Bansal et al. (2022) and per-sequence depth sampling, using different recurrences (T) on a held-out validation set Geiping et al. (2025). **Bolded** results indicate **best** for each scale.

	Method	T=1	T=4	T=8	T=16
100M	Batch	300.32	36.75	16.65	13.81
	Sequence	**70.47**	**17.15**	**14.08**	**13.59**
350M	Batch	167.61	12.80	10.40	10.24
	Sequence	**17.92**	**10.49**	**10.09**	**10.11**

5.2 Accelerated Training Convergence

To evaluate the accelerated training convergence, we compare training loss gaps and validation perplexity between Parcae and prior RDMs Geiping et al. (2025) throughout training. As seen in Figure 4, Parcae exhibits accelerated learning compared to baseline RDMs, converging 1.25x faster without requiring residual normalization. For both 100M and 350M models, Parcae maintains a consistently lower loss-delta throughout the training process. Figure 5 further mirrors this improvement, demonstrating a stronger validation perplexity in all sampled recurrences throughout training. The consistent gap suggests that the stable parameterization of Parcae facilitates stronger learning.

5.3 Improved Downstream Quality

Beyond accelerated training convergence, Parcae outperforms both Transformers and prior RDMs on standard open-source benchmarks. As shown in Table 4, Parcae achieves the highest average scores across a suite of publicly available datasets. At the 350M scale with $T=8$, Parcae reaches an average score of 50.21, surpassing the RDM baseline and the fixed-depth Transformer. Additionally, from Table 4 we observe that the average accuracy of the 100M Parcae model is only 1.02% under the 350M transformer model, while being 4.88% over the 100M transformer model, highlighting Parcae's parameter efficiency. Furthermore, for in-domain and out-of-distribution perplexity metrics in Table 3, Parcae reduces LAMBADA perplexity by roughly 26.9% and 64.7% compared to RDMs and Transformers, respectively, at the 100M scale.[7] Similar trends can be observed at the 350M scale, indicating that Parcae's stability translates directly into more robust linguistic representations.

[7]We stipulate that high perplexity on Pile and Lambada likely stems from Geiping et al. (2025) data mixture.

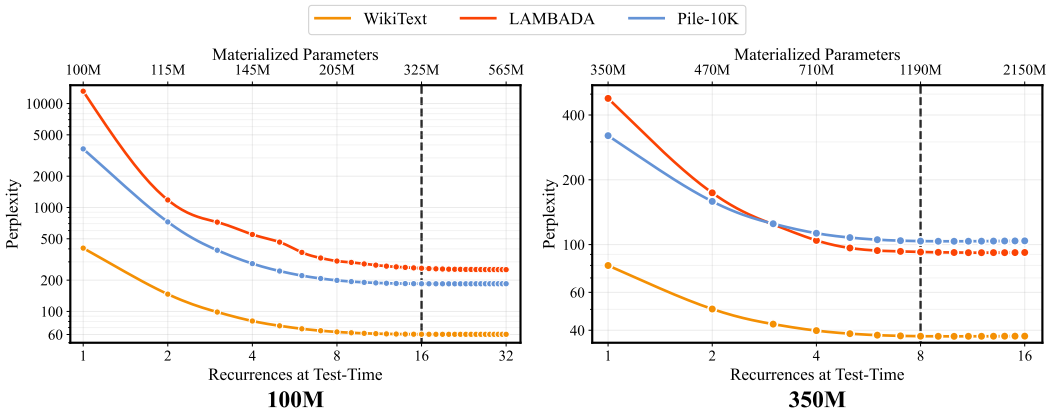

Figure 7: **Scaling of Test-Time Recurrence.** Perplexity results on WikiText Merity et al. (2016), LAMBADA Paperno et al. (2016), and Pile-10k Nanda (2022) datasets as a function of test-time recurrences, showing performance improvements as compute (materialized parameters) scales for both 100 and 350 million parameter Parcae models. Dashed line indicates μ_{rec} used in training.

5.4 Stable Recurrent Training

We evaluate stability by comparing the hyperparameter sensitivity of looped architectures. We also evaluate our per-sequence algorithm's ability to mitigate loss spikes and further improve quality.

Architectural Stability. A primary motivator for our dynamical system framework was the observation that both Pre-Norm and residual normalized RDMs suffered from training instability (see Figure 1). To evaluate the robustness of Parcae, we test for convergence across a range of learning rates. As seen in Table 1, baseline Pre-Norm RDMs and RDMs with residual normalization suffer from training divergence at higher learning rates. In contrast, the constrained parameterization of Parcae enables stable convergence across all tested configurations, suggesting that Parcae is significantly less susceptible to precise hyperparameter selection. Extended details in Section F.

Training Algorithm Stability. To understand the impact of per-sequence depth sampling compared to per-batch sampling Bansal et al. (2022), we train Parcae models with per-batch sampling, where we observe that per-sequence depth sampling further helps stability by preventing loss spikes during training (see Figure 6; extended results in Section G). Additionally, per-sequence sampling lowers perplexity across all evaluation depths (see Table 5), with the most noticeable improvements at lower recurrences, showing that our method additionally improves model quality.

6 Scaling Compute via Recurrence

Unlike fixed-depth models, looped architectures can decouple FLOPs from parameters in both training and inference by increasing recurrences. In section Section 6.1, we show a new scaling axis of FLOPs by showing monotonic improvements in validation loss when scaling up recurrence in training. Additionally, we observe that recurrence can scale computation at inference time (Section 6.2), but fascinatingly *without* increasing memory overhead by using a single KV cache for all recurrences (Section 6.3), allowing memory-efficient test-time computation.

6.1 Recurrence as a New Scaling Axis in Training

While looped architectures decouple FLOPs from parameters and data in training, it is not apparent if scaling up the depth of recurrence (i.e., μ_{rec}) in training improves model quality. To understand the impact of recurrence as a scaling axis, we train four 100M Parcae models with different μ_{rec}, keeping the architecture and hyperparameters the same. The validation loss throughout each training run can be observed Figure 8, where we note that increasing μ_{rec} consistently improves the model quality, effectively increasing FLOPs in training, without increasing parameterization or data. This implies the existence of a new axis for scaling FLOPs during training, orthogonal to parameters and data.

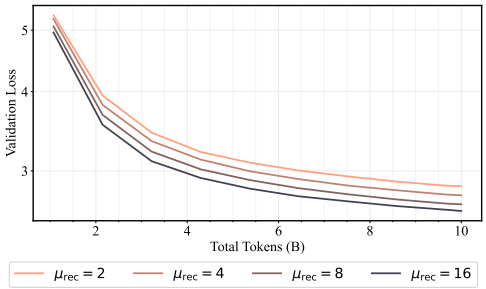

Figure 8: **Scaling of μ_{rec}.** Training with larger μ_{rec} monotonically improves the validation loss of 100M Parcae throughout training.

Table 6: **Comparing KV-Cache Sharing**. We record perplexity on a subset of Geiping et al. (2025) validation data, using full and last-recurrence KV-caching, showing last-recurrence KV-caching as a viable method at larger inference time recurrences T.

	Method	T=2	T=4	T=8	T=16
100M	Full	29.54	17.84	14.65	14.15
	Last	49.26	18.69	14.38	14.15
350M	Full	12.61	10.64	10.24	10.25
	Last	18.76	10.62	10.25	10.26

6.2 Test-Time Compute Scaling via Recurrence

Looped architectures can scale recurrences at test time, allowing the model to increase its computational capacity without expanding its weight storage. Specifically, we find that increasing the number of recurrences of Parcae at inference time consistently leads to monotonic improvements in model performance across WikiText, LAMBADA, and Pile-10k datasets (see Figure 7). Of note, we observe that the performance of scaling test-time compute with recurrences saturates approximately at the μ_{rec} used in training, indicating a fundamental limitation of training looped models.

6.3 Memory Efficient Single KV-Cache Sharing

At inference, one concern with scaling recurrences is a linear increase in KV-cache size. While Geiping et al. (2025) used circular KV-caching, we observe that Parcae can use the final recurrences' KV-cache with negligible degradation of performance at large recurrences (see Table 6). To measure the degradation of compression, we calculate the perplexity of each token using only the last recurrences' KV-cache for all prior-tokens on a subset of validation data Geiping et al. (2025).[8] To understand the impacts of this strategy, we visualize the attention map for a single token across multiple recurrences for full and last-recurrence KV-caches, noting that both extract features similarly in middle and final recurrences. We include visualization of KV-Cache strategies in Section H.

7 Conclusion

In this work, we propose a novel framework for understanding and controlling the residual dynamics of looped architectures by recasting depth as a nonlinear time-variant dynamical system. Using this framework, we derive exact constraints on parameterization, which we use to create a novel looped architecture, Parcae. We empirically show that Parcae's architecture and training algorithms improve downstream quality, accelerate convergence, and stabilize training. Unlike fixed-depth models, Parcae decouples FLOPs from parameters, enabling a new scaling axis in training and memory-efficient scaling at test-time with last-recurrence KV-caching. Our dynamical systems framework for recurrence enables and warrants deeper exploration of: parameterizations of $\boldsymbol{A}$; usage of discretization; and recurrent algorithms. Furthermore, the stability of Parcae encourages deeper exploration of FLOP and memory-efficient algorithms and architectures for both training and inference.

References

Cem Anil, Ashwini Pokle, Kaiqu Liang, Johannes Treutlein, Yuhuai Wu, Shaojie Bai, Zico Kolter, and Roger Grosse. Path independent equilibrium models can better exploit test-time computation, 2022. URL https://arxiv.org/abs/2211.09961.

[8]This task can be fairly computationally expensive due to the serial token-by-token perplexity calculation. Thus, we use a subset of 64 sequences from the validation set.

Anonymous. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RzYXb5YWBs.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation, 2025. URL https://arxiv.org/abs/2507.10524.

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=PPjSKy40XUB.

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, 2020.

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.

Róbert Csordás, Christopher D Manning, and Christopher Potts. Do language models use their depth efficiently? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=Kz6eUL86XP.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality, 2024. URL https://arxiv.org/abs/2405.21060.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

C. Desoer and Min-Yen Wu. Stability of linear time-invariant systems. *IEEE Transactions on Circuit Theory*, 15(3):245–250, 1968. doi: 10.1109/TCT.1968.1082819.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJg7KhVKPH.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.681. URL http://dx.doi.org/10.18653/v1/2024.acl-long.681.

Katie Everett, Lechao Xiao, Mitchell Wortsman, Alexander A. Alemi, Roman Novak, Peter J. Liu, Izzeddin Gur, Jascha Sohl-Dickstein, Leslie Pack Kaelbling, Jaehoon Lee, and Jeffrey Pennington. Scaling exponents across parameterizations and optimizers, 2024. URL https://arxiv.org/abs/2407.05872.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022. URL https://arxiv.org/abs/2101.03961.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL https://arxiv.org/abs/2210.17323.

Jonas Geiping and Tom Goldstein. Cramming: Training a language model on a single GPU in one day. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11117–11143. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/geiping23a.html.

Jonas Geiping, Sean Michael 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. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=S3GhJooWIC.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL https://arxiv.org/abs/2312.00752.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces, 2022. URL https://arxiv.org/abs/2111.00396.

Soufiane Hayou and Fadhel Ayed. Regularization in resnet with stochastic depth, 2021. URL https://arxiv.org/abs/2106.03091.

Geoffrey E. Hinton and Ilya Sutskever. Training recurrent neural networks, 2013. URL https://api.semanticscholar.org/CorpusID:61713861.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep networks with stochastic depth, 2016. URL https://arxiv.org/abs/1603.09382.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks, 2025. URL https://arxiv.org/abs/2510.04871.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

Wenquan Lu, Yuechuan Yang, Kyle Lee, Yanshu Li, and Enqi Liu. Latent chain-of-thought? decoding the depth-recurrent transformer, 2025. URL https://arxiv.org/abs/2507.02199.

Sean McLeish, Ang Li, John Kirchenbauer, Dayal Singh Kalra, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Jonas Geiping, Tom Goldstein, and Micah Goldblum. Teaching pretrained language models to think deeper with retrofitted recurrence. *arXiv preprint arXiv:2511.07384*, 2025.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Neel Nanda. NeelNanda/pile-10k – datasets at hugging face. https://huggingface.co/datasets/NeelNanda/pile-10k, 2022.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 1525–1534, 2016.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models, 2024. URL https://arxiv.org/abs/2404.02258.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial Winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers, 2025. URL https://arxiv.org/abs/2502.17416.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=Tsp2PL7-GQ.

Noam Shazeer. Glu variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017. URL https://arxiv.org/abs/1701.06538.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

Sho Takase, Shun Kiyono, Sosuke Kobayashi, and Jun Suzuki. Spike no more: Stabilizing the pre-training of large language models, 2025. URL https://arxiv.org/abs/2312.16903.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/4496bf24afe7fab6f046bf4923da8de6-Paper.pdf.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model, 2025. URL https://arxiv.org/abs/2506.21734.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *NUT@EMNLP*, 2017.

Mitchell Wortsman, Tim Dettmers, Luke Zettlemoyer, Ari S. Morcos, Ali Farhadi, and Ludwig Schmidt. Stable and low-precision training for large-scale vision-language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=sqqASmpA2R.

Bohong Wu, Mengzhao Chen, Xiang Luo, Shen Yan, Qifan Yu, Fan Xia, Tianqi Zhang, Hongrui Zhan, Zheng Zhong, Xun Zhou, Siyuan Qiao, and Xingyan Bin. Parallel loop transformer for efficient test-time computation scaling, 2025. URL https://arxiv.org/abs/2510.24824.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture, 2020. URL https://openreview.net/forum?id=B1x8anVFPr.

Kevin Xu and Issei Sato. On Expressive Power of Looped Transformers: Theoretical Analysis and Enhancement via Timestep Encoding, June 2025.

Liu Yang, Kangwook Lee, Robert D Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=HHbRxoDTxE.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length, 2025. URL https://arxiv.org/abs/2406.06484.

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*, 2019.

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12104–12113, June 2022.

Biao Zhang and Rico Sennrich. Root mean square layer normalization, 2019. URL https://arxiv.org/abs/1910.07467.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling latent reasoning via looped language models, 2025. URL https://arxiv.org/abs/2510.25741.

A GLOSSARY

A.1 NOTATION

Notation	Description
d	Embedding **dimension** of the model
t	Discrete temporal **state** axis of $\mathcal{R}$ on $\mathbb{N}$
b	Global batch size used during pretraining
$\bar{b}$	Micro batch size used with gradient accumulation in pretraining
$\bar{d}$	Number of devices used during training
$\mathcal{P}$	Initial **prelude** block of a recurrent architecture
$\mathcal{R}$	Middle **recurrent** block of a recurrent architecture
$\mathcal{C}$	Final **coda** block of a recurrent architecture
$\boldsymbol{A}$	The linear continuous state transition matrix
$\boldsymbol{B}$	The linear continuous state injection matrix
$\boldsymbol{C}$	The linear state output matrix
Δ	Learnable discrete parameter for decay, discretizing our model
s	Input sequence to a model
e	Output embedding of the prelude block $\mathcal{P}$
h	Hidden embedding of the recurrent block $\mathcal{R}$
μ_{rec}	Mean recurrent forward propagation steps during pre-training
μ_{bwd}	Mean recurrent backward propagation steps during pre-training
n	Sampled number of recurrent steps with no gradient updates
k	Sampled number of recurrent steps with gradient updates
T	Sampled or fixed number of recurrent steps
Λ	Distribution that recurrences are sampled from during training

Table 7: Glossary of notation and terminology. (*Top*) Frequently used dimensions for tensors. (*Middle*) Definition of Parcae blocks. (*Bottom*) Tensors and distributions are used to express recurrent depth models.

A.2 COMMON METRICS

- Recurrent Residual Metric: $||h_T - h_{T-1}||_2$, where $r \sim \Lambda$. This metric tells us how much we jump around at the final recurrence. Overly small jumps indicate that $\mathcal{R}$ isn't learning anything meaningful, while overly large jumps indicate $\mathcal{R}$ is suffering from state explosion or is unable to learn fixed-point dynamics.
- Recurrent State Norm: $||h_T||$, where $T \sim \Lambda$. In general, we don't want an overly large recurrent state norm as it creates numerical instabilities and leads to overly large gradients.

B EXTENDED LITERATURE REVIEW

Recurrence in model depth is not novel; with a large body of work studying recurrence within general language modeling (Dehghani et al., 2019; Zhu et al., 2025; Geiping et al., 2025; McLeish et al., 2025; Bae et al., 2025) or small-scale algorithmic problems (Schwarzschild et al., 2021; Yang et al., 2024; Bansal et al., 2022; Wang et al., 2025; Jolicoeur-Martineau, 2025). Within recurrent depth architectures, the design of training paradigms can be relatively split between architectures with explicit halting mechanisms (Dehghani et al., 2019; Zhu et al., 2025; Bae et al., 2025; Jolicoeur-Martineau, 2025; Wang et al., 2025) and those with implicit halting mechanisms (Geiping et al., 2025; McLeish et al., 2025; Anonymous, 2026; Xu & Sato, 2025). Beyond training paradigms, there are several differing architectural design choices of recurrent depth models Geiping et al. (2025); Bansal et al. (2022); Saunshi et al. (2025).

Training Paradigms. Recurrent depth architectures trained with an explicit halting mechanism use specialized architectures to predict when to early exit tokens, preventing additional computation updates on their recurrent stream (Wang et al., 2025; Jolicoeur-Martineau, 2025; Bae et al., 2025; Dehghani et al., 2019;

Elbayad et al., 2020). Specifically, Wang et al. (2025); Jolicoeur-Martineau (2025) formalize *adaptive-computation-time*, a method that utilizes Q-learning as a means to determine convergence. Similarly, works such as Bae et al. (2025) define an architecture that has light-weight routers to assign dynamic recursion depths, while Zhu et al. (2025) uses a prediction head to dynamically define a probability of exiting after recurrent passes. A majority of these approaches draw on methods of layer skipping Elhoushi et al. (2024); Raposo et al. (2024); however, these methods differ from using a shared parameterization for a recurrent block. Alternatively, recurrent depth architectures with an implicit halting mechanism, such as Geiping et al. (2025); McLeish et al. (2025), train models with stochastically sampled recurrent steps during pretraining, and then use the KL-divergence between two successive steps to decide when to exit from the recurrent unit early. Finally, Anonymous (2026) ignores adaptive early exiting altogether, instead pretraining a recurrent unit on a static number of recurrences and enforcing a consistency loss on intermediate recurrences. For our purposes, we focus on implicit recurrent depth models Geiping et al. (2025); McLeish et al. (2025), which are derived from initial work by Schwarzschild et al. (2021); Bansal et al. (2022).

Recurrent Depth Architectures. Where recurrent units are introduced into an architecture has additionally been previously explored by Saunshi et al. (2025), where they find the middle looping in recurrent units is the most effective. However, there are several other formulations, such as pre-looping and post-looping, which loop the beginning and end of the model. These findings are consistent with initial work in synthetic problems by Bansal et al. (2022); Schwarzschild et al. (2021) and with the architecture choices of Geiping et al. (2025); McLeish et al. (2025) in large-scale language model pertaining. The formulation of the architecture by Geiping et al. (2025); McLeish et al. (2025) follows the prior work showing the importance of injection (Bansal et al., 2022; Yang et al., 2024) and the importance of state-independent initialization to encourage *path-independence* (Anil et al., 2022).

C Discretization of Non-Linear Dynamical Systems

Consider the non-linear ODE from Equation (8a):

$$\dot{h}(t) = \boldsymbol{A}h(t) + \boldsymbol{B}e(t) + f(h(t)) \tag{11}$$

ZOH Discretization. Assume $e(t) = e_k$ and $f(h(t)) \approx f(h_k)$ constant over $[k\Delta, (k+1)\Delta]$.

$$\begin{aligned}
h_{k+1} &= \exp(\boldsymbol{A}\Delta)h_k + \int_0^{\Delta} \exp(\boldsymbol{A}(\Delta - s))(\boldsymbol{B}e_k + f(h_k))ds \qquad \text{(variation of constants)} && (12)\\
&= \exp(\boldsymbol{A}\Delta)h_k + \int_0^{\Delta} \exp(\boldsymbol{A}s)(\boldsymbol{B}e_k + f(h_k))ds && (13)\\
&= \exp(\boldsymbol{A}\Delta)h_k + \left(\int_0^{\Delta} \exp(\boldsymbol{A}s)ds\right)(\boldsymbol{B}e_k + f(h_k)) && (14)\\
&= \exp(\boldsymbol{A}\Delta)h_k + \boldsymbol{A}^{-1}(\exp(\boldsymbol{A}\Delta) - I)(\boldsymbol{B}e_k + f(h_k)) && (15)\\
&= \overline{\boldsymbol{A}}h_k + \overline{\boldsymbol{B}}e_k + \overline{\boldsymbol{\Delta}}f(h_k) && (16)
\end{aligned}$$

where (assuming $\boldsymbol{A}$ is invertible) then

$$\overline{\boldsymbol{A}} = \exp(\Delta\boldsymbol{A}) \qquad \overline{\boldsymbol{B}} = (\Delta\boldsymbol{A})^{-1}(\exp(\Delta\boldsymbol{A}) - I)\Delta\boldsymbol{B} \qquad \overline{\boldsymbol{\Delta}} = (\Delta\boldsymbol{A})^{-1}(\exp(\Delta\boldsymbol{A}) - I)\Delta$$

Euler Discretization. Using forward Euler, approximate $\dot{h}(t) \approx \frac{h_{k+1} - h_k}{\Delta}$ and evaluate the RHS at (h_k, e_k):

$$\begin{aligned}
h_{k+1} &= h_k + \Delta(\boldsymbol{A}h_k + \boldsymbol{B}e_k + f(h_k)) && (17)\\
&= (I + \Delta\boldsymbol{A})h_k + \Delta\boldsymbol{B}e_k + \Delta f(h_k) && (18)\\
&= \overline{\boldsymbol{A}}h_k + \overline{\boldsymbol{B}}e_k + \overline{\boldsymbol{\Delta}}f(h_k). && (19)
\end{aligned}$$

where

$$\overline{\boldsymbol{A}} = (I + \Delta\boldsymbol{A}) \qquad \overline{\boldsymbol{B}} = \Delta\boldsymbol{B} \qquad \overline{\boldsymbol{\Delta}} = \Delta$$

D Per-Sample Training Algorithm

We display our algorithm to sample per-sequence depths during Parcae training while maintaining compute efficiency in Algorithm 1. We do per-sequence depth sampling, but taking the max depth within a batch and performing no state updates at the *beginning* of the recurrent computation. This allows for batched processing of different depths while maintaining efficient gradient flow.

Algorithm 1 Efficient Per-Sequence Stochastic Depth Training

Require: Batch of sequences $\{s_i\}_{i=1}^{B}$, means $\mu_{\text{rec}}, \mu_{\text{bwd}}$, and sampling distribution Λ

$\boldsymbol{e}^{(i)} \leftarrow \mathcal{P}(s_i)$ for all i ▷ embed sequences
Sample $T^{(i)} \sim \Lambda(\mu_{\text{rec}})$ for each $i \in [B]$
$T_{\max} \leftarrow \max_i T^{(i)}, \quad \tau^{(i)} \leftarrow T_{\max} - T^{(i)}$
$\boldsymbol{h}_0^{(i)} \sim \mathcal{N}(0, \sigma \boldsymbol{I})$ for all i
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} \leftarrow \text{DISCRETIZE}(\boldsymbol{A}, \boldsymbol{B}, \Delta t)$
for $t = 0, \ldots, T_{\max} - 1$ **do**
  for all i **where** $t < \tau^{(i)}$: $\quad \boldsymbol{h}_{t+1}^{(i)} \leftarrow \boldsymbol{h}_t^{(i)}$ ▷ no state update
  for all i **where** $\tau^{(i)} \leq t < T_{\max} - \mu_{\text{bwd}}$: ▷ without gradients
    $\boldsymbol{h}_{t+1}^{(i)} \leftarrow \overline{\boldsymbol{A}} \boldsymbol{h}_t^{(i)} + \overline{\boldsymbol{B}} \boldsymbol{e}^{(i)} + \mathcal{R}(\boldsymbol{h}_t^{(i)}, \boldsymbol{e}^{(i)})$
  for all i **where** $t \geq T_{\max} - \mu_{\text{bwd}}$: ▷ with gradients
    $\boldsymbol{h}_{t+1}^{(i)} \leftarrow \overline{\boldsymbol{A}} \boldsymbol{h}_t^{(i)} + \overline{\boldsymbol{B}} \boldsymbol{e}^{(i)} + \mathcal{R}(\boldsymbol{h}_t^{(i)}, \boldsymbol{e}^{(i)})$
end for
return $\{\mathcal{C}(\boldsymbol{C} \boldsymbol{h}_{T_{\max}}^{(i)})\}_{i=1}^{B}$

E Model Forward Pass Algorithm

Algorithm 2 Parcae Forward Pass

Require: Input sequence $s \in V^n$ and recurrent steps T.

$e \leftarrow \mathcal{P}(s)$
$h_0 \sim \mathcal{N}(0, \sigma^2 I_{n \times d})$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} \leftarrow \boldsymbol{A}, \boldsymbol{B}, \Delta$
for $t = 1$ to T **do**
  $h_t \leftarrow \overline{\boldsymbol{A}} h_{t-1} + \overline{\boldsymbol{B}} e + \overline{\mathcal{R}}(h_t, e)$
end for
return $\mathcal{C}(\boldsymbol{C} h_T)$

F Additional Stability Ablations

We include all training curves for our hyperparameter sweep experiments in Section 5.4. We conduct a learning rate sweep over $\{2e-4, 4e-4, 6e-4, 8e-4, 1e-3\}$ observing that Parcae exhibits stable training over both baseline Pre-Norm RDMs and residual normalized RDMs. The training curves and the accompanying recurrent state norm can be observed in Figure 9.

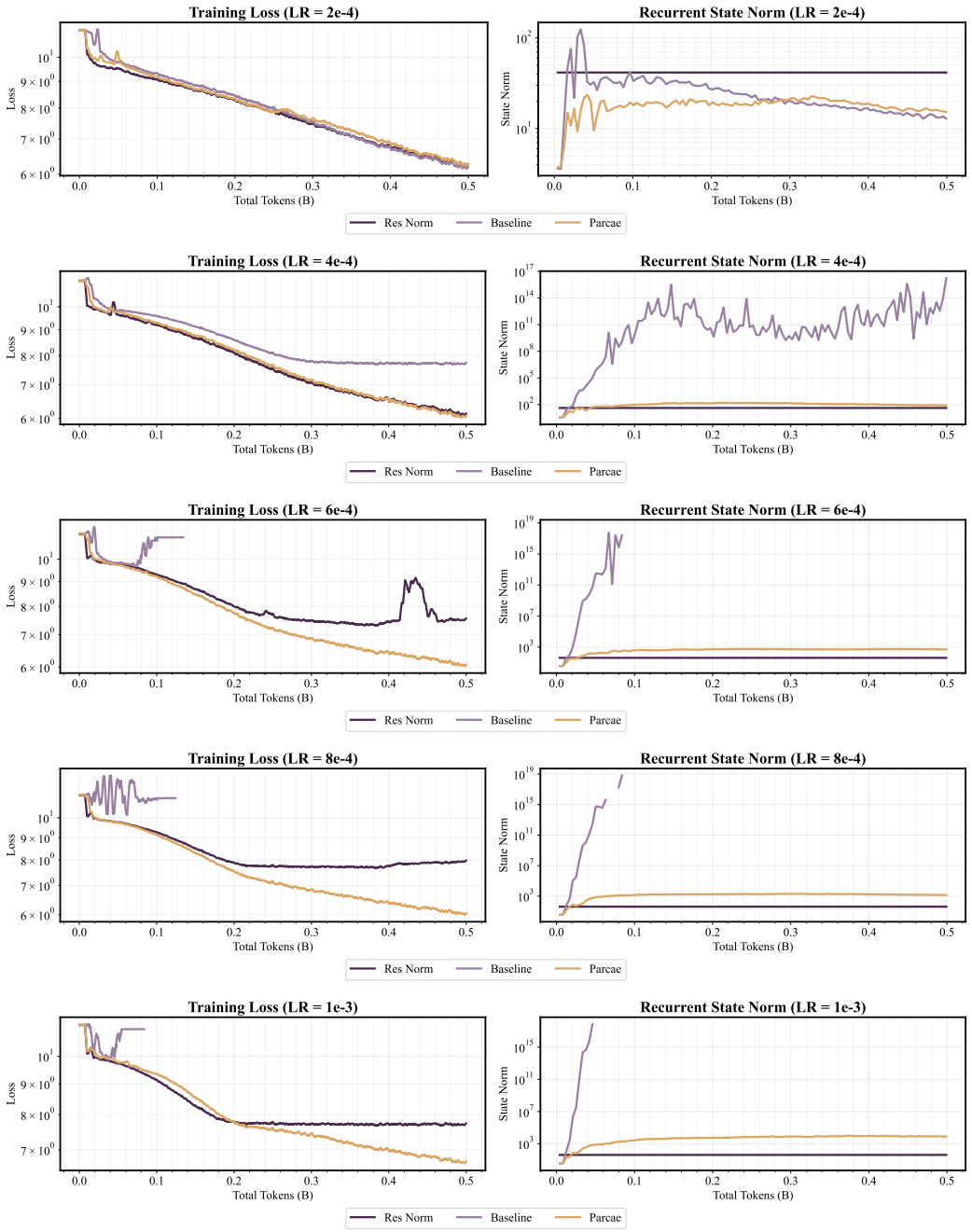

Figure 9: Training instability of recurrent depth models across different learning rates. We show both training losses and recurrent state norm to understand divergence and state explosion.

G Per-sequence Sampling Reduces Loss Spikes

When running our per-sequence sampling experiments, we observed that the training curves of per-sequence sampling helped eliminate loss spikes during training. Specifically, in Figure 10, we can observe that for our 350M parameter Parcae models, per-micro-batch has several loss spikes through training while per-sequence sampling does not. We can observe from Figure 11, that these training spikes stem directly from overly large recurrent residual jumps at the final recurrence, implying the model is not learning to converge to a steady-state fixed point solution. Since the recurrent state norm does not exhibit these spikes,

it appears that the per-microbatch depth can effectively stay on the manifold of language learned by the model, but jumps around too much and does not learn to converge to a fixed point. It can then be observed that per-sequence depth helps provide a better estimate for our training objective, enabling convergent fixed-point behavior and preventing loss spikes during training.

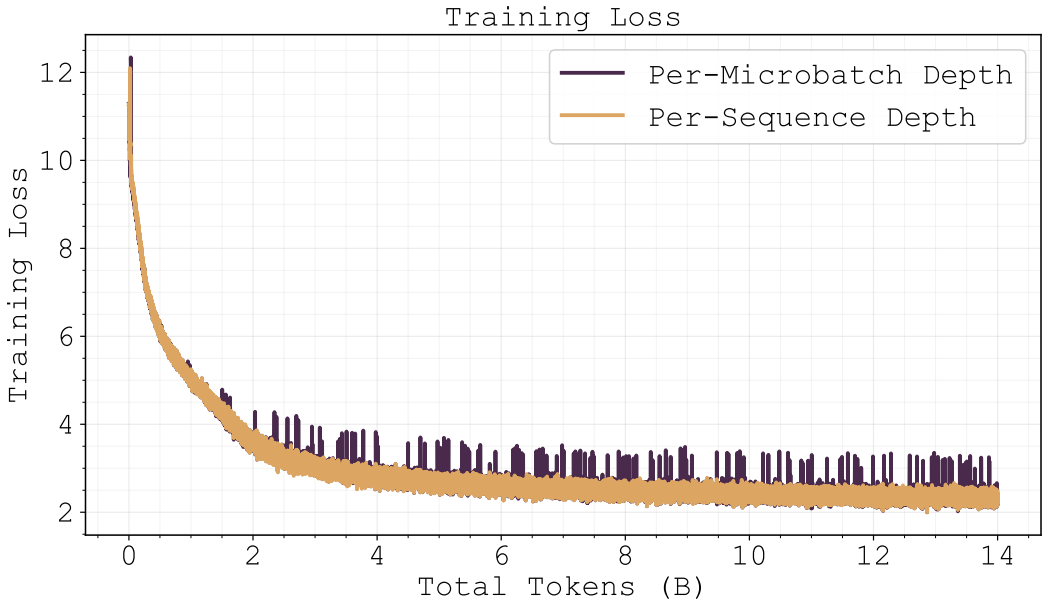

Figure 10: Training curves showing per-sequence sampling effectively eliminates loss spikes in training over per-micro-batch sampling.

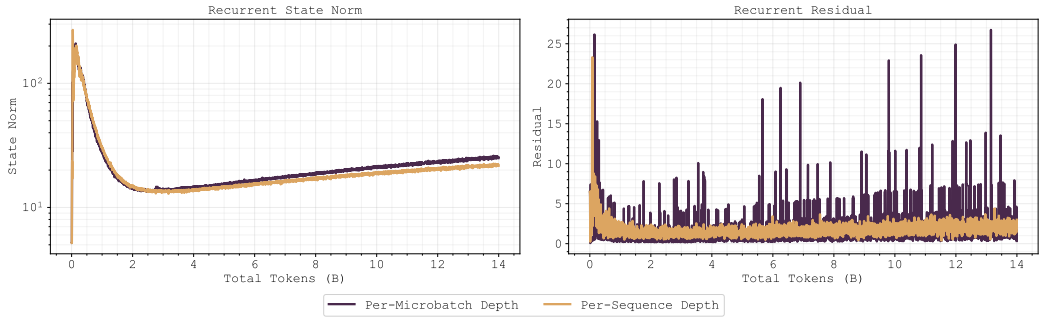

Figure 11: Comparison of recurrent residual and state norm metrics (defined in Section A.1), which show that per-sequence sampling enables stronger fixed point behavior in training.

H Additional KV-Cache and Attention Map Analysis

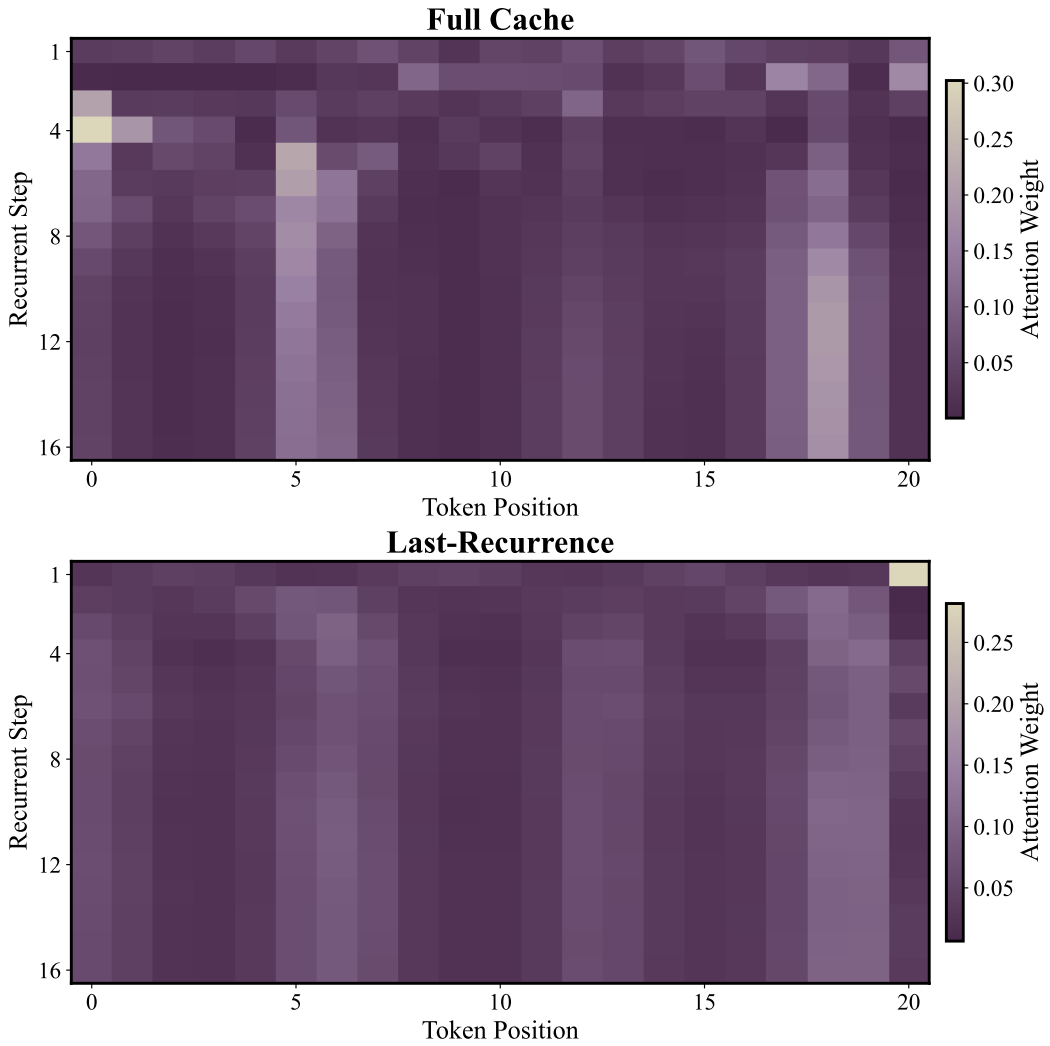

Figure 12: **Visualization of KV-cache Strategies Attention Maps.** We visualize attention maps for a single token across multiple recurrences for both full (top) and last-recurrence (bottom) KV-cache strategies to understand their effect on state computation. Both methods converge to similar feature maps as recurrence increases.

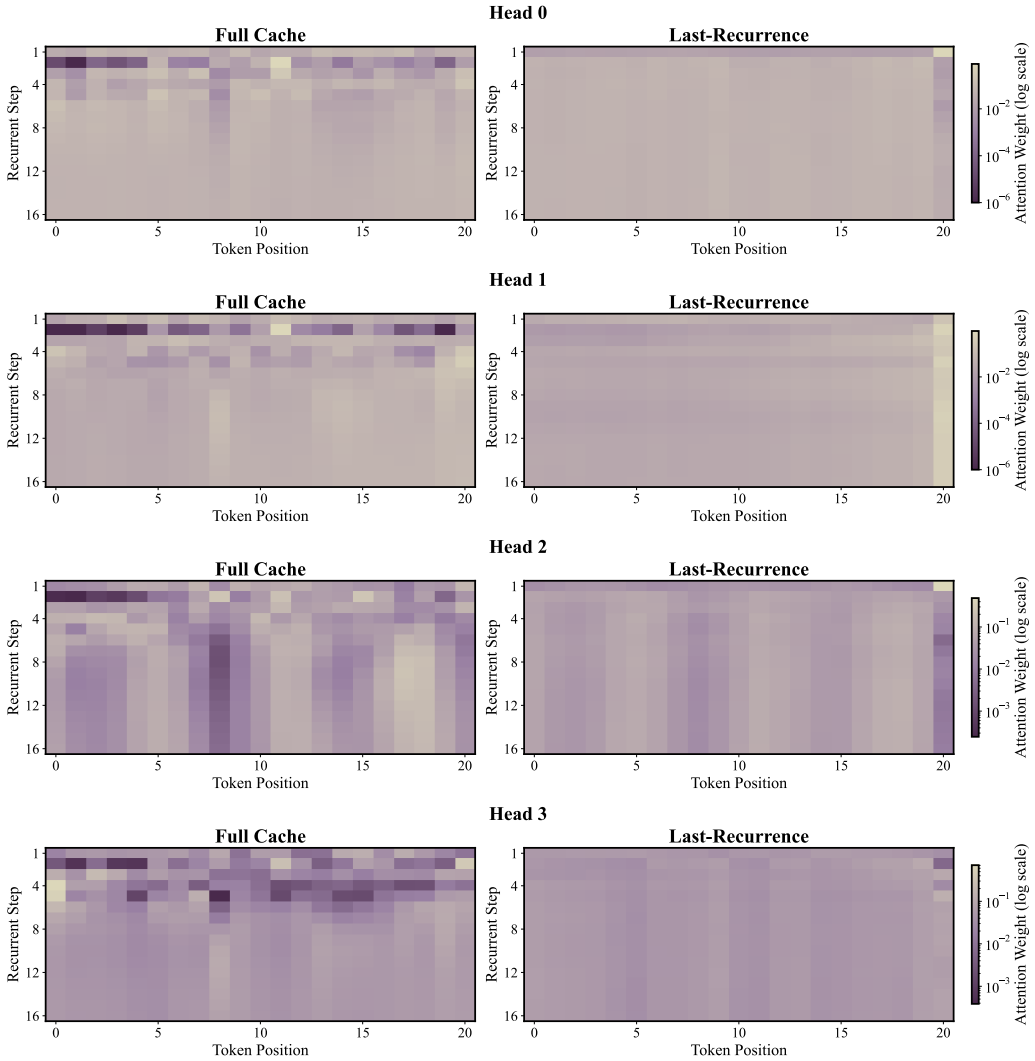

Figure 13: Attention map comparison for a single token in a sequence for the validation set of Geiping et al. (2025) throughout recursions. Observe that in later recurrences, Last-Recurrence and Full-Cache strategies extract features in a similar pattern.

I SAMPLING OF TRUNCATED RECURRENCE

In our very initial experiments, we observed that we could make a small change to the sampling algorithm of Geiping et al. (2025), which stems from Schwarzschild et al. (2021), to enhance the training of Parcae[9]. When given an arbitrary distribution to sample from Λ and two hyperparameters μ_{rec} (the desired mean steps of the recurrent blocks in pre-training) and μ_{bwd} (the desired mean back-propagation steps in pre-training), we observe that previous work by Geiping et al. (2025) had a distributional mismatch. Previously, the sampling method of Geiping et al. (2025) exactly followed Algorithm 3 with a poisson log-normal distribution with the following distribution

$$\tau \sim \mathcal{N}(\log(\mu_{\text{rec}} - \mu_{\text{bwd}}) - \frac{1}{2}\sigma^2, \sigma) \qquad n \sim \mathcal{P}(e^{\tau}) + 1 \qquad k \leftarrow \mu_{\text{bwd}} \tag{20}$$

where $\sigma = \frac{1}{2}$. To maintain a fixed computation memory budget, Geiping et al. (2025) sets k to μ_{bwd}; however, this minor change significantly impacts the underlying recurrent distribution, truncating and compressing the distribution of recurrence actually observed during pre-training. We propose making a minor algorithmic fix

[9]We make the same change to RDMs in the main body, observing that they perform better with it.

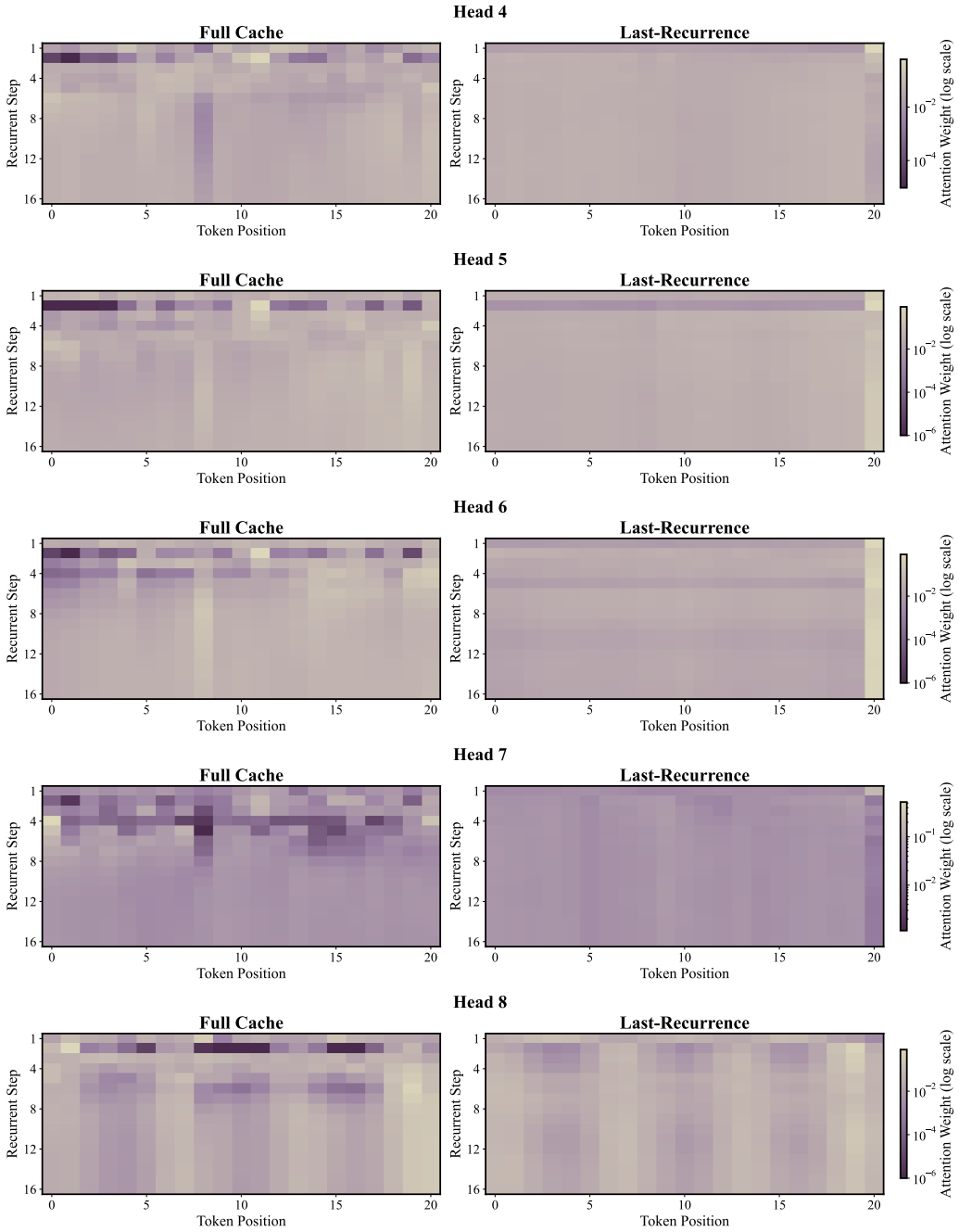

Figure 14: Attention map comparison for a single token in a sequence for the validation set of Geiping et al. (2025) throughout recursions. Observe that in later recurrences, Last-Recurrence and Full-Cache strategies extract features in a similar pattern.

to the sampling method, which can be observed in Algorithm 4. While a minor change, it can be observed in Figure 16 the implications the minor tweak in sampling has on the underlying recurrent distribution.

Beyond aligning the recurrent distribution with Λ, we pretrain several small Parcae models on 10 billion tokens to ablate on our design choice. Specifically, we set $\mu_{\text{rec}} = \mu_{\text{bwd}} = 8$ and use $\Lambda \sim$ Poisson, and use fixed architecture, hyperparameters, and data stream. We train three models: a baseline Parcae model that performs full backpropagation through recurrences, a Parcae model following Algorithm 3 by Geiping et al. (2025), and a Parcae model following Algorithm 4. The results of this ablation can be found in Figure 17.

Algorithm 3 GEIPING ET AL. (2025)

Input: μ_{rec}, μ_{bwd}, Λ, e
$n \sim \Lambda(\mu_{\text{rec}} - \mu_{\text{bwd}})$
$k \leftarrow \mu_{\text{bwd}}$
$T = n + k$
$h_0 \leftarrow \mathcal{N}(0, \sigma^2 I)$
for $t = 1$ **to** T **do**
  if $t \leq n$ **then**
    $h_t \leftarrow \mathcal{R}(h_{t-1}, e)$ **w/o grad**
  else
    $h_t \leftarrow \mathcal{R}(h_{t-1}, e)$ **w/ grad**
  end if
end for
return x_T

Algorithm 4 CORRECTION (OURS)

Input: μ_{rec}, μ_{bwd}, Λ, e
$T \sim \Lambda(\mu_{\text{rec}})$
$n \leftarrow \max(T - \mu_{\text{bwd}}, 0)$
$k \leftarrow \min(T, \mu_{\text{bwd}})$
$h_0 \leftarrow \mathcal{N}(0, \sigma^2 I)$
for $t = 1$ **to** T **do**
  if $t \leq n$ **then**
    $h_t \leftarrow \mathcal{R}(h_{t-1}, e)$ **w/o grad**
  else
    $h_t \leftarrow \mathcal{R}(h_{t-1}, e)$ **w/ grad**
  end if
end for
return x_T

Figure 15: Comparison of our sampling method with Geiping et al. (2025). It can be observed that the actual distribution of forward recurrence for Geiping et al. (2025) is a shifted Poisson distribution. The implications of this sampling strategy can be better visualized in Figure 16.

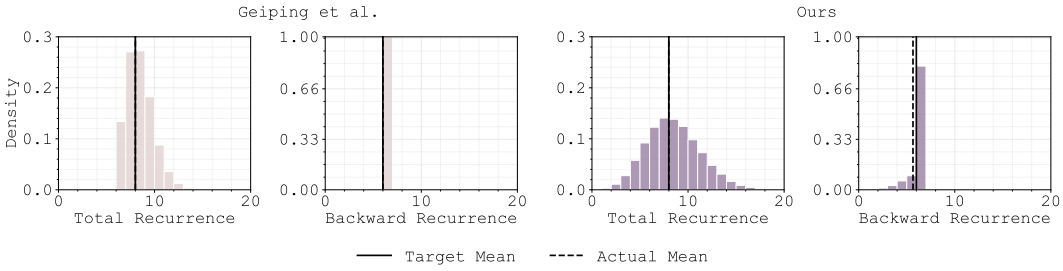

Figure 16: A distributional mismatch can be observed from the recurrent sampling method of Geiping et al. (2025). Specifically, if our desired pre-training distribution for μ_{rec} is a Poisson distribution, the distribution total recurrence T of Geiping et al. (2025) is truncated based on μ_{bwd}. However, our sampling method decouples the effects of μ_{bwd} on Λ, allowing the recurrent distribution to be faithfully sampled from.

From Figure 17, it can be observed that even though training trajectories and validation loss at $T = \mu_{\text{rec}} = 8$ are almost identical for each run, our method which decouples Λ from μ_{bwd}, significantly improves performance for the validation loss of, $T \in [4, 16, 16]$. Simply put, the constricting effect Geiping et al. (2025) observed in Figure 16 significantly reduces the effective range of recurrence seen in pretraining, hurting the validation loss of using more or fewer recurrence at test-time. The coupling of Λ with μ_{bwd} is especially undesirable when attempting to understand the best distribution to sample recurrences from.

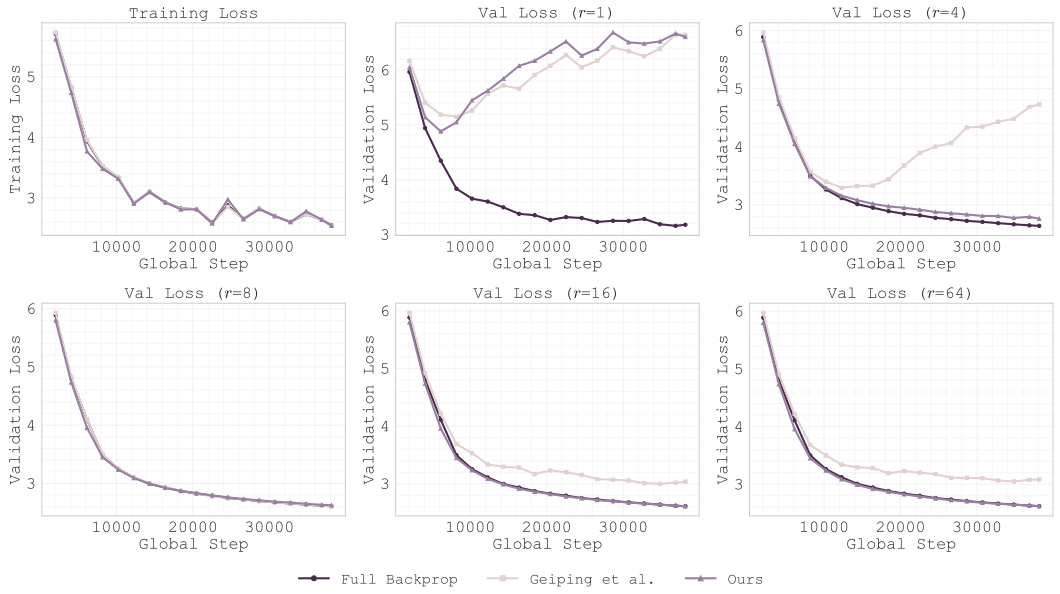

Figure 17: Training and validation curves of three 100 million parameter Parcae models pretrained on 10 billion tokens, comparing different truncated back-propagation methods (baseline is a model with no back-propagation truncation). Each model has identical architecture and hyperparameters, with μ_{rec} and μ_{bwd} both being set to eight, all using $\Lambda \sim$ Poisson. It can be observed that even though each model has similar training loss and validation loss when using $T=8$, our implementation more faithfully follows the validation loss of full back-propagation. Specifically for $T=4$, our implementation significantly improves validation loss compared to Geiping et al. (2025) sampling method.

J SELECTING μ_{REC} AND μ_{BWD}

	$\mu_{\text{rec}}=4$	$\mu_{\text{rec}}=8$	$\mu_{\text{rec}}=14$	$\mu_{\text{rec}}=20$	$\mu_{\text{rec}}=26$	$\mu_{\text{rec}}=32$
Val Loss	2.477	**2.453**	2.456	2.457	2.458	2.458
Val Perplexity	11.906	**11.624**	11.665	11.671	11.692	11.687

Table 8: Validation loss and perplexity for six different 150 million parameter models, each trained with different μ_{rec}, but a fixed $\mu_{\text{bwd}}=4$. We select $r=\mu_{\text{rec}}$. Surprisingly, the model where $\mu_{\text{rec}}=8$ performs the best out of each of the models.

A natural question is what choice of μ_{rec} and μ_{bwd} is appropriate for pretraining recurrent depth models. To answer this question, we conduct an experiment where we scale up μ_{rec}, while keeping μ_{bwd} fixed. In our very initial experiments, we pretrained several 150M parameter recurrent depth models on 10 billion tokens, with a fixed $\mu_{\text{bwd}}=4$ and with $\mu_{\text{rec}} \in [4,8,14,20,26,32]$[10]. The results for each of these models on a held-out set of validation data can be observed in Figure 18. We additionally include Table 8, which gives the validation loss of each model with $\mu_{\text{rec}} \in [4,8,14,20,26,32]$, where the recurrence that we use for each model at test-time is $T=\mu_{\text{rec}}$.

The fascinating observation of Figure 18 is that, contrary to our initial expectations, models trained with additional μ_{rec} beyond 8 perform worse at both lower and higher r used at test-time, even though more FLOPs were spent during pretraining these models. While it is a natural expectation that models trained with lower μ_{rec} perform better than models with larger μ_{rec} at low T, the fact that a μ_{rec} of eight performs the best at higher T (i.e., $T=16$ and $T=64$) is a surprising result. To determine if this is an inherent limitation of the capacity of the total parameters of the recurrent flow models or an artifact of the μ_{bwd}, we

[10]Note that these experiments were run before the distribution mismatch fix discussed in Section I. As the mismatch becomes more drastic as μ_{rec} gets closer to μ_{bwd}, we expect the model pretrained with $\mu_{\text{rec}}=4$ to be performing sub-optimally.

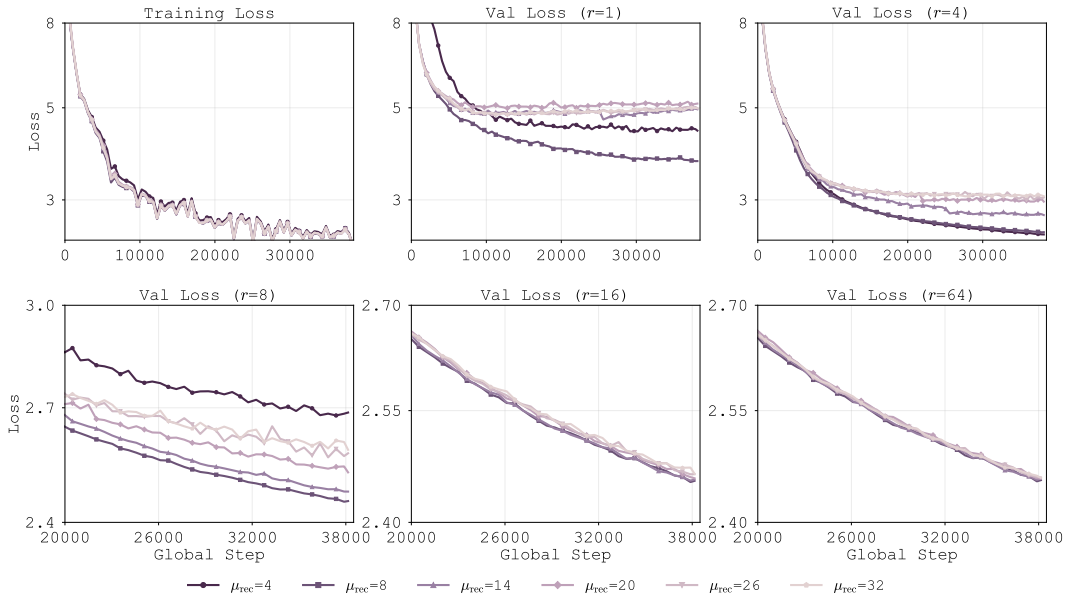

Figure 18: Validation curves of six different 150 million parameter recurrent depth models, pretrained on 10 billion tokens, with a fixed architecture and hyperparameters. Each model is pretrained with a fixed μ_{bwd} of 8 and varying μ_{rec} in $[4,8,14,20,26,32]$. The key observation is that scaling up μ_{rec} while keeping μ_{bwd} fixed, results in models which perform worse than if just pretrained on μ_{rec} of eight.

ran an additional experiment where we fixed $\mu_{rec}=20$ and instead varied $\mu_{\text{bwd}} \in [4,6,8,10,12]$, pretraining on 8.5 billion tokens for each model (reduced due to the increased computational load of increasing μ_{bwd}). Again, we keep hyperparameters fixed. The results for each of these models on a held-out set of validation data can be visualized in Figure 19. Again, we additionally provide Table 9, where we provide the final validation results of each pretrained model using $T=64$ at validation time.

	$\mu_{\text{bwd}}=4$	$\mu_{\text{bwd}}=6$	$\mu_{\text{bwd}}=8$	$\mu_{\text{bwd}}=10$	$\mu_{\text{bwd}}=12$
Val Loss	2.500	2.490	2.480	2.479	**2.474**
Val Perplexity	12.09	12.06	11.94	11.93	**11.86**

Table 9: Caption

While lower μ_{bwd} (i.e., $\mu_{\text{bwd}}=4,6,8$) appears to perform better with lower validation recurrences than higher μ_{bwd}, the validation loss using $T=16,64$ improves as μ_{bwd} increases. This implies that the capabilities of recurrent flow models utilizing deeper recurrences are heavily coupled with μ_{bwd}. However, it can be observed that increasing μ_{bwd} from ten to twelve has minimal impact on validation performance, at the cost of higher pretraining FLOPs. Using this insight, for our main training runs, we choose to use

$$\mu_{\text{bwd}}=\lfloor\frac{\mu_{\text{rec}}}{2}\rfloor \tag{21}$$

for our main training runs.

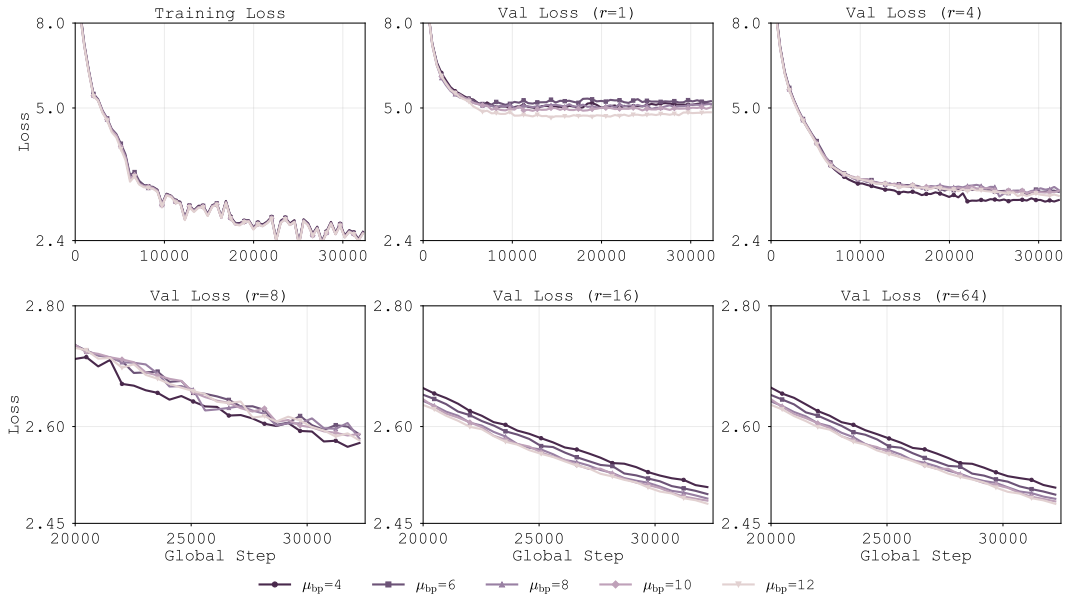

Figure 19: Validation and training curves of five different 150 million parameter recurrent flow models, pretrained on 8.5 billion tokens. Each model is pretrained with a fixed $\mu_{\text{rec}} = 20$ and $\mu_{bwd} \in [4,6,8,10,12]$. The key observation is that scaling up μ_{bwd} improves validation performance at higher and lower recurrences monotonically for $T = 1,16,64$. Interestingly, $\mu_{\text{rec}} = 4$ performs the best at $T = 4,8$, but consistently underperforms at higher and lower recurrence.

K Model Definitions

For all $\mathcal{P}$, $\mathcal{R}$, and $\mathcal{C}$ modules, we follow Geiping et al. (2025), and use standard, causal self-attention and gated SiLU MLP (Shazeer, 2020). For attention, we use RoPE (Su et al., 2023) with $\theta = 50000$ and for normalization we use RMSNorm Zhang & Sennrich (2019). We use Pre-Norm transformer blocks for all modules within Parcae, and follow Takase et al. (2025), initialize weights using $\mathcal{N}(0, \frac{2}{5d})$, where d is the model dimension.

	Parcae-100M	Parcae-350M	RDM-100M	RDM-350M
Parameters	114,238,464	378,558,464	114,242,560	382,765,056
Layers in $\mathcal{P}$	1	1	1	1
Layers in $\mathcal{C}$	1	1	1	1
Layers in $\mathcal{R}$	1	2	1	2
d_{model}	1,024	2,048	1,024	2,048
$d_{\text{intermediate}}$	3,520	7,040	3,520	7,040
Attention	Causal Self-Attention			
MLP	SiLU			
Pos. Embed.	RoPE			
Vocab Size	65,536			
Norm	RMS-Norm			

Table 10: Model definitions of both Parcae and baseline residual-norm RDMs Geiping et al. (2025).

L Hyperparameters and Training Details for Main Experiments and Ablations

We train with a warm-up and cool-down (4096 steps) and a constant learning rate ($\eta = 4 \times 10^{-3}$ for 100M models and $\eta = 2 \times 10^{-3}$ for 350M models) Geiping & Goldstein (2023); Zhai et al. (2022). As our

optimizer, we use Adam with decoupled weight regularization ($\beta_1 = 0.9, \beta_2 0.95$) Kingma & Ba (2017); Loshchilov & Hutter (2019), using update clipping Wortsman et al. (2023) and removing the ϵ constant Everett et al. (2024). Gradients above 1 are clipped.