Solve the Loop: Attractor Models for Language and Reasoning

Anonymous Authors[1]

Abstract

Looped Transformers offer a promising alternative to purely feed-forward computation by iteratively refining latent representations, improving language modeling and reasoning. Yet recurrent architectures remain unstable to train, costly to optimize and deploy, and constrained to small, fixed recurrence depths. We introduce *Attractor Models*, in which a backbone module first propose output embeddings, then an attractor module refines them by solving for the fixed point, with gradients obtained through implicit differentiation. Thus, training memory remains constant in effective depth, and iterations are chosen adaptively by convergence. Empirically, this yields a single mechanism that outperforms existing models and scales across two regimes, large-scale language modeling and reasoning with tiny models. In language modeling, Attractor Models outperform standard Transformers and stable looped models across sizes, improving perplexity by up to 46.6% and downstream accuracy by up to 19.7% while reducing training cost. On challenging reasoning tasks, we show that our model with only 27M parameters and around 1000 examples achieves 91.4% accuracy on Sudoku-Extreme and 93.1% on Maze-Hard, scaling favorably where frontier models fail completely and specialized recursive reasoners collapse at larger sizes. Lastly, we show Attractor Models enjoy a novel phenomenon, which we call *equilibrium internalization*: fixed-point training places the model's initial output embedding near equilibrium, allowing the solver to be removed at inference-time with little degradation.

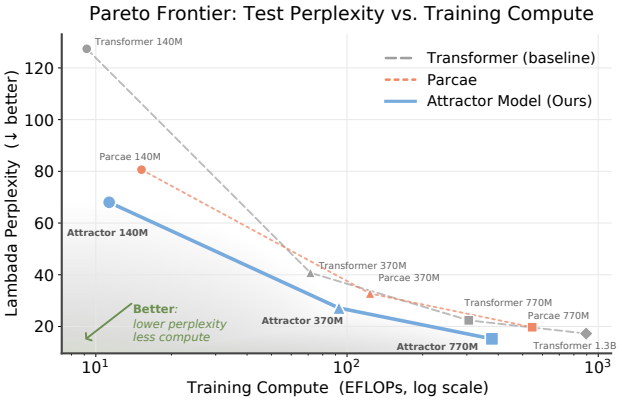

Figure 1. Pareto frontier of Lambada perplexity vs training FLOPs. Attractor Models achieve strong performance on large-scale language modeling with less compute.

1. Introduction

The modern language-modeling era has been dominated by Transformers (Vaswani et al., 2017), which produce each token through a fixed feed-forward computation. Recent industry-scale LLMs are increasingly pretrained on token budgets far beyond those prescribed by the compute-optimal Chinchilla scaling laws (Hoffmann et al., 2022), with models such as Qwen3 and SmolLM3 continuing this trend (Yang et al., 2025; Bakouch et al., 2025). In parallel, a growing line of work has explored reusing the same computational blocks to iteratively refine a representation for a fixed input (Sapunov, 2026; Yang et al., 2024a; Labovich, 2026; Saunshi et al., 2025a; Xu & Sato, 2025). Chain-of-thought reasoning (Wei et al., 2023) can be viewed as one form of such refinement, where a model writes intermediate tokens, feeds them back into its context, and uses them to shape later predictions. While effective at test time, this approach is difficult to integrate into pretraining because it requires reasoning through discrete token generation, which is expensive and can make optimization unstable.

This limitation has inspired several lines of work on latent (or implicit) thinking and a re-emergence of architectural recurrence, which move thinking away from purely token-level generation. These include Universal Transformers (Dehghani et al., 2019), looped Transformers (Fan et al., 2025), looped language models (Zhu et al., 2025), latent

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the FoGen Workshop at ICML 2026. Do not distribute.

reasoning methods (Wang et al., 2025; Jolicoeur-Martineau, 2025), and continuous chain-of-thought approaches (Hao et al., 2025). Looped architectures can, in principle, express iterative or algorithmic procedures (Yang et al., 2024b), emulate additional depth through weight sharing (Dehghani et al., 2019), reduce the context-length costs of token-level reasoning, and improve downstream generalization (Zhu et al., 2025). Empirically, recent looped language models offer gains in language modeling and reasoning (Prairie et al., 2026; Zhu et al., 2025), and tiny recursive models (Jolicoeur-Martineau, 2025) have shown that recurrence can be useful on hard reasoning tasks in small-data regimes.

The obstacle is that recurrence has remained difficult to stabilize. It is often accompanied by unstable training, growing memory requirements, and significant, sequential compute (Prairie et al., 2026). Training recurrent networks typically requires difficult backpropagation through time and carefully designed stabilization techniques; even then, latent-thinking models remain unstable and difficult to optimize (Wei et al., 2025). For training, looped language models tend to require substantially more compute than comparable feed-forward models and can become memory-limited at larger recurrence depths. For instance, (Schwethelm et al., 2026) reports that training a recurrent model can consume raw FLOPs comparable to those of a feed-forward model ten times larger. At the opposite end of the spectrum, specialized tiny recursive reasoners exhibit a troubling "less is more" behavior and respond negatively to scaling: increasing model size can degrade or even collapse performance (Jolicoeur-Martineau, 2025).

Contributions

In this work, we ask whether a single, general-purpose architecture for iterative refinement exists that can satisfy these criteria at once: (i) stable to train, (ii) constant-memory in the number of refinement steps, (iii) substantially cheaper to train than explicit unrolling, (iv) efficient during inference, and (v) achieves a strong performance across both large-scale language modeling and hard reasoning with tiny models.

Refine outputs by solving the loop with Attractor Models. We introduce *Attractor Models*, a new family of architectures that treat latent refinement as a fixed-point problem in the output embedding space. The model first proposes an initial guess embedding using a non-recurrent backbone module. A separate, typically smaller, recurrent module then refines this guess. Recent mechanistic analyses of looped language models demonstrate that, for the vast majority of tokens, the recurrent trajectory converges to a fixed point (Blayney et al., 2026). We build directly on this observation and instead of unrolling the loop for a predefined number of steps, we solve for the state to which the loop converges, inspired by Deep Equilibrium Models (DEQ; (Bai et al., 2019)). The name *Attractor Model* comes from dynamical systems, where an attractor is a set of states toward which a system evolves. In a sense, Attractor Models can be viewed as *thinking* before producing each token: the backbone proposes an initial latent prediction, the attractor module refines it to equilibrium, which is decoded into the next-token distribution.

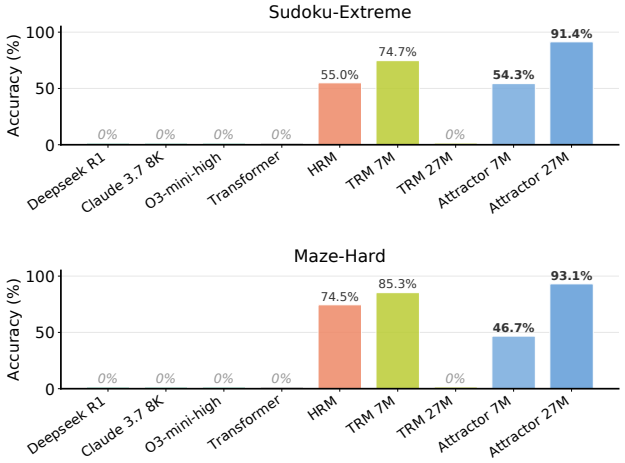

Figure 2. Attractor Models outperform both frontier models and specialized recursive models in hard reasoning tasks.

Attractor Models offer stable, constant-memory, efficient training, and adaptive refinement. Unlike looped LMs, which finitely unroll the recurrent block, Attractor Models solve an equilibrium by treating the prediction target as a fixed-point computation. The number of refinement steps is therefore chosen adaptively according to convergence.

Novel phenomenon: Equilibrium internalization. We observe that despite the fact that Attractor models are trained only with the next-token prediction loss, they learn to make the solver unnecessary. During training, the backbone's initial prediction moves progressively closer to the fixed point, so fewer refinement steps are needed to reach approximate equilibrium (c.f. Figures 7 and 6). We call this phenomenon *equilibrium internalization*: the model appears to self-distill the iterative refinement process into its own initial output embedding, through a form of automatic curriculum. In this sense, recurrence acts as a moving training target, teaching the backbone where its computation should converge.

Strong performance in large-scale language modeling and hard reasoning with tiny models. Our experiments show that this single mechanism scales across two regimes. In large-scale language modeling, Attractor Models consistently outperform standard Transformers and stable looped language models across small (140M), medium (370M), and large (770M) sizes. They improve validation perplexity,

out-of-distribution perplexity on Lambada (Paperno et al., 2016), and downstream benchmark accuracy while using substantially less training compute than comparable looped baselines. Notably, a 770M-parameter Attractor Model outperforms a 1.3B-parameter Transformer trained on twice as many tokens. Compared to looped LM Parcae (Prairie et al., 2026), our models use roughly half the training compute across model sizes, while avoiding the memory growth associated with explicit unrolling. In hard reasoning tasks with tiny models, with only 27M parameters and approximately 1000 training examples, Attractor Models achieve 91.4% accuracy on Sudoku-Extreme and 93.1% on Maze-Hard. In this regime, standard Transformers as well as proprietary frontier models such as DeepSeek R1, Claude, and o3-mini fail completely at 0%, while specialized recursive architectures underperform our model and collapse when scaled. Attractor Models, in contrast, improve with scale.

2. Background: Looped Architectures

We start by a providing a background on looped architectures. Let $x = (x_1, \ldots, x_n) \in \mathcal{V}^n$ be an input sequence over vocabulary $\mathcal{V}$, and let d denote the model width. Looped models can be written as the composition of three units. Looped models can be written as the composition of three units: a prelude unit $\tilde{x} = \mathcal{P}(x) \in \mathbb{R}^{n\times d}$, which produces an input representation $\tilde{x} \in \mathbb{R}^{n\times d}$; a weight-tied recurrent unit $h_{t+1} = \mathcal{R}(h_t, \tilde{x})$, which is applied repeatedly to a latent state $h_t \in \mathbb{R}^{n\times d}$; and a coda unit, which maps the final latent state to output probabilities $p = \mathcal{C}(h_T) \in \Delta(\mathcal{V})^n$. Importantly, looped architectures commonly initiate the latent state at an *uninformative value*, such as $h_0 = 0$ or Gaussian noise $h_0 \sim \mathcal{N}(0, \sigma^2 I)$ (Geiping et al., 2025a). Furthermore, the recurrent step may use the input representation only at the first step (Zhu et al., 2025). or into every recurrent step (Prairie et al., 2026). Such injection may be through addition to or concatenation with the recurrent state.

Models such as Parcae, Huggin, and Ouro differ mainly in how they train, stop, or scale this looped architecture. In particular, the recurrence depth T is a central choice in these models. It may be fixed, sampled during training, or determined by an auxiliary halting mechanism. Training then minimizes an objective averaged over both the data distribution and the chosen recurrence-depth mechanism, typically by backpropagation through depth. Consequently, both training cost and gradient memory are tied to the number of recurrent steps, and changing T at inference changes the computation on which the model was trained.

3. Solve the Loop with Attractor Models

As discussed in the previous section, standard looped language models (Zhu et al., 2025; Prairie et al., 2026) use weight sharing to recurrently refine a hidden state initialized from an uninformative value and based on input embeddings. Predictions are then read out after a finite number of loops (Prairie et al., 2026), or once an auxiliary halting head becomes confident (Graves, 2017; Zhu et al., 2025). This design carries three drawbacks: the loop count must be chosen at training time, training memory grows linearly in the number of loops, and accuracy degrades when more loops are run at inference than were seen during training (Zhu et al., 2025). As a result, recurrence often comes with unstable training, growing memory requirements, and large sequential compute, in some cases approaching training non-recurrent models ten times large (Zhu et al., 2025).

Recent mechanistic analyses of looped language models (Blayney et al., 2026) reveals that for the vast majority of tokens, the recurrent trajectory eventually converges to a fixed point. This suggests that the weight-tied recurrent modules are often approximating an underlying fixed-point computation, and do so by recursive implementation truncated after T steps. This observation motivates the design of Attractor Models, which we subsequently describe.

3.1. Attractor Model: Backbone and Attractor Modules

Motivated by the fixed-point behavior observed in looped models, we model recurrent refinement as an attractor. Rather than training a model to produce good predictions after a prescribed number of recurrent steps, we define the output as the equilibrium of the recurrent refinement. Attractor Models consist of two modules: the *backbone module* first proposes a meaningful initial output embedding, and the *attractor module* then refines this proposal until convergence. This makes the number of refinement steps a solver choice rather than a fixed architectural choice.

We first start by mapping the inputs x into input embeddings $\tilde{x} = E(x) \in \mathbb{R}^{n\times d}$, where E denotes the tied embedding/unembedding. Then, the input embedding is processed by the backbone and attractor modules as described below.

The backbone module proposes an initial output embedding. The backbone module $\mathcal{T}_{\theta_b}$ maps the input embeddings to an initial proposal: $\tilde{y}_0 = \mathcal{T}_{\theta_b}(\tilde{x})$, where $\tilde{x} = E(x)$. We use $\tilde{y}_0$ as an initialization for the attractor module. Instead of initializing the loop from zero, noise, or an input-side representation, Attractor Models initialize the recurrent computation from a state that is already a coherent prediction embedding. In practice, $\mathcal{T}_{\theta_b}$ is a relatively high-capacity causal Transformer, so the refinement begins near a meaningful initialization rather than 0.

The attractor module: refine the output embedding. The attractor module is a separate weight-tied refinement map $\mathcal{T}_{\theta_a}$. Starting from the backbone proposal $\tilde{y}_0$, it repeatedly refines the output embedding according to $\tilde{y}_{t+1} =$

$\mathcal{T}_{\theta_a}(\tilde{y}_t, \tilde{y}_0)$, where $\tilde{y}_0 = \mathcal{T}_{\theta_b}(\tilde{x})$. Here, we persistently inject the initial proposal $\tilde{y}_0$ at every refinement step. Injecting $\tilde{y}_0$ at every step keeps the attractor proposal-dependent and prevents collapse to a proposal-independent fixed point. We ablate this conditioning mechanism in Section 4.

Rather than rolling out recurrent steps to reach a fixed point, we directly solve for convergence:

$$\mathcal{A}_{\theta_a}(\tilde{y}^\star, \tilde{y}_0) \coloneqq \mathcal{T}_{\theta_a}(\tilde{y}^\star, \tilde{y}_0) - \tilde{y}^\star = 0$$
$$\Rightarrow \tilde{y}^\star = \texttt{RootFind}(\mathcal{A}_{\theta_a}(\cdot, \tilde{y}_0); y_0)\,.$$

In the forward pass, we compute this equilibrium with a root finder initialized at the backbone proposal. In our implementation, `RootFind` uses Anderson acceleration, which combines a small window of past iterates and residuals to reach the fixed point faster than plain recursion. The solver exits when $\|\mathcal{A}_{\theta_a}(\tilde{y}_t, \tilde{y}_0)\|_2 / \|\tilde{y}_t\|_2 < \varepsilon$ or after $T_{\max}$ steps. Thus, computation is controlled by convergence of the residual rather than by a learned halting head or a preset loop count. In contrast to fixed unrolling, the number of refinement steps can therefore vary at inference time without changing the model. Finally, the equilibrium embedding is decoded with the tied unembedding.

Parameters of the Attractor Models will be a combination of tied embedding/unembedding matrix, parameters of the backbone module, and parameters of the attractor module: $\theta \coloneqq (\theta_a, \theta_b, E)$. Compared to finite-loop models, Attractor Models change both the starting point and the endpoint of recurrence: the loop is initialized from an output-side proposal $\tilde{y}_0$, and the decoded state is the attractor $\tilde{y}^\star$ rather than a finite unroll.

3.2. Training and Inference of Attractor Models

We now describe how Attractor Models are trained. We first explain how to differentiate through the fixed-point solve using implicit differentiation, and then present the loss used to train the model.

Backward pass and implicit differentiation. Because Attractor Models define the output embedding as the solution to a fixed-point equation, we differentiate through the equilibrium using the implicit function theorem. Let $\mathcal{L}$ denote the training loss and let $v = \partial\mathcal{L}/\partial\tilde{y}^\star$. Applying the implicit function theorem to $\mathcal{A}_{\theta_a}(\tilde{y}^\star, \tilde{y}_0) = 0$ gives

$$\frac{\partial\mathcal{L}}{\partial\theta} = u^\top \frac{\partial\mathcal{T}_{\theta_a}(\tilde{y}^\star, \tilde{y}_0)}{\partial\theta}, \quad u = \left(I - J_{\tilde{y}}^\top\right)^{-1} v,$$
$$\text{where} \quad J_{\tilde{y}} = \left.\frac{\partial\mathcal{T}_{\theta_a}(\tilde{y}, \tilde{y}_0)}{\partial\tilde{y}}\right|_{\tilde{y}=\tilde{y}^\star}.$$

The derivative with respect to $\theta = (\theta_a, \theta_b, E)$ includes the direct dependence on the attractor parameters θ_a, as well as the dependence on the initialization $\tilde{y}_0 = \mathcal{T}_{\theta_b}(E(x))$ through the backbone parameters θ_b and the tied embedding parameters E.

Following prior work on implicit models (Geng et al., 2022; Fung et al., 2021), we use the one-step approximation $u \approx v$. This avoids the extra linear solve for u and reduces the backward pass to one vector–Jacobian product through $\mathcal{A}_{\theta_a}$. Since we do not backpropagate through every solver step, memory in the attractor block does not grow with the number of forward iterations.

Remark. Attractor Models are implicit equilibrium models in the spirit of DEQs (Bai et al., 2019), but the equilibrium plays a different role. Classical DEQs replace the prediction network with a hidden-state equilibrium $z^\star$ (single-layer), often initialized from an uninformative state and decoded with a separate output head. We instead keep a standard causal Transformer backbone and add an equilibrium refinement block on top of its prediction state. The fixed point lives directly in the tied embedding space, so every iterate $\{y_0, y_1, \ldots, y^\star\}$ is already an output-side representation that can be decoded. This gives two practical differences: the solver is initialized from a semantically meaningful proposal $\tilde{y}_0$ rather than from an uninformative state such as zero (as in DEQ), and inference can stop according to a residual tolerance ε rather than a fixed depth or learned halting head.

Training and Inference. We train Attractor Models using the standard next-token prediction cross-entropy objective using $y^\star$. Inference reuses the same equilibrium computation. Given an input sequence x, the backbone first produces the proposal $\tilde{y}_0 = \mathcal{T}_{\theta_b}(E(x))$, the attractor solver computes $\tilde{y}^\star$, and the tied unembedding decodes $\tilde{y}^\star$ into next-token probabilities. Peak memory is bounded by a single forward through the attractor module and standard KV-caching applies in the backbone. In principle, the solver tolerance ε and maximum iteration budget $T_{\max}$ are inference-time hyperparameters: they can be adjusted without retraining the model, turning test-time computation into a budget for approaching the learned attractor. Interestingly, however, we find that trained Attractor Models often require very little test-time refinement, as we describe in the next section.

3.3. Equilibrium Internalization and Stability Bias in Attractor Models

Although Attractor Models define predictions through the equilibrium $\tilde{y}^\star$, we observe a surprising phenomenon: after training, the backbone proposal $\tilde{y}_0$ often lies close to the equilibrium (c.f. Figures 7 and 6). We refer to this phenomenon as *equilibrium internalization*. Intuitively, the attractor module appears to act as a moving teacher for the backbone, resulting in a form of automatic curriculum. Early in training, the proposal $\tilde{y}_0$ may be far from a good prediction, and the solver must perform nontrivial refinement to reach $\tilde{y}^\star$. Since $\tilde{y}_0$ and $\tilde{y}^\star$ live in the same tied

Table 1. Parameter scaling results.

Size	Model	Val. PPL ↓	Lambada PPL ↓	Core ↑	Core-Ext. ↑
140M	Transformer	21.48	127.39	13.00 ± 0.15	8.80 ± 0.21
	PARCAE	19.06	80.64	14.04 ± 0.20	9.67 ± 0.28
	Attractor Model	**18.30**	**68.02**	$\mathbf{14.59 \pm 0.11}$	$\mathbf{10.03 \pm 0.05}$
	rel. Δ	↓14.8%	↓46.6%	↑12.2%	↑14.0%
370M	Transformer	15.79	40.77	17.46 ± 0.03	11.71 ± 0.22
	PARCAE	14.49	32.74	20.00 ± 0.06	$\mathbf{12.75 \pm 0.31}$
	Attractor Model	**14.03**	**27.14**	$\mathbf{20.24 \pm 0.09}$	12.64 ± 0.33
	rel. Δ	↓11.1%	↓33.4%	↑15.9%	↑7.9%
770M	Transformer	13.08	22.37	22.42 ± 0.20	14.20 ± 0.63
	PARCAE	12.49	19.71	25.07 ± 0.33	15.19 ± 0.43
	Attractor Model	**12.09**	**15.21**	$\mathbf{26.83 \pm 0.29}$	$\mathbf{15.42 \pm 0.51}$
	rel. Δ	↓7.6%	↓32.0%	↑19.7%	↑8.6%
1.3B	Transformer	11.95	17.26	25.45 ± 0.08	15.90 ± 0.23

output-embedding space, gradients through the equilibrium also train the backbone proposal to move toward the state that the solver would have found. Thus, when the backbone is sufficiently expressive, much of the prediction work can be internalized into $\tilde{y}_0$, leaving the attractor to perform a stable refinement.

Stability bias. Equilibrium training also biases the recurrent map toward convergent dynamics. The implicit gradient contains the inverse factor $(I - J_{\tilde{y}}^{\top})^{-1}$, which becomes ill-conditioned near non-contractive regimes. This creates a barrier against unstable fixed-point dynamics, unlike fixed-loop training, which can learn trajectories that are accurate only at a prescribed step count and fail under extra inference-time loops. We discuss this contrast in detail in Appendix B.4. We refer to Appendix B for theoretical analysis of Attractor Models and comparison with finite-loop models.

4. Experiments

In this section, we evaluate Attractor Models in two regimes. We first study large-scale language modeling across model sizes, comparing against parameter-matched Transformers and looped-LM baselines. We evaluate scaling behavior, downstream accuracy, and training efficiency. We also present results on hard reasoning tasks, where we test whether the same fixed-point refinement mechanism improves small models on problems that require iterative computation.

4.1. Attractor Models Improve Large-Scale Language Modeling

Setup. We follow the `nanochat` (Karpathy, 2025) pretraining recipe used by Parcae (Prairie et al., 2026) for its main Transformer comparison, training on FineWeb-Edu (Penedo et al., 2024). To ensure a fair comparison, all models are matched in parameter count and trained with the same data budget, optimizer, and learning-rate schedule as the Parcae baselines; the only change is the recurrent block. We compare at three scales: 140M, 370M, and 770M parameters. Parcae is a middle-looped language model with prelude, coda, and recurrent blocks. The stability in this model comes from a linear injection that bounds the spectral radius of the recurrence below one. In contrast, our architecture uses standard Transformer blocks followed by a fixed-point iteration block, and lets the solver itself control the effective depth. Detailed hyperparameter settings are in Appendix C.

Parameter Scaling. We demonstrate how large-scale pretraining scales with our model. We evaluate our method on 140M, 370M, and 770M parameters against a parameter-matched Transformer and PARCAE, a looped-language model (Prairie et al., 2026). Our model improves monotonically with scale. Across all three sizes, our method achieves the best validation PPL, Lambada PPL, and CORE accuracy. These results show that our fixed-point refinement scales cleanly with model size, with especially large gains on Lambada, where iterative refinement substantially improves long-context prediction.

Figure 3. **Training-time efficiency.** Despite being a looped architecture, our method uses 25–31% fewer FLOPs than Parcae because the fixed-point solver often converges before $T_{\max}$ and the backward pass uses the one-step implicit-gradient approximation.

Training efficiency: Lower FLOPs. The IFT backward pass keeps training memory constant in the number of solver iterations. The memory of standard looped language models like (Prairie et al., 2026) scales linearly with the number of loops. The total training FLOPs (Figure 3) follow the

same trend: although every step of our recurrent block costs roughly the same as Parcae's, our solver typically converges below ε in well under $T_{\max}$ steps, so the realized depth during training is lower despite identical $T_{\max}$, yielding 25–31% lower training FLOPs across scales.

Training efficiency: O(1) memory An additional advantage of our method is that the training memory required stays constant with the number of iterations (Figure 4). This is because our implicit backward pass does not need to store the intermediate activations from every recurrent step. In contrast, standard looped language models must backpropagate through each unrolled iteration, causing memory usage to grow linearly with the number of loops.

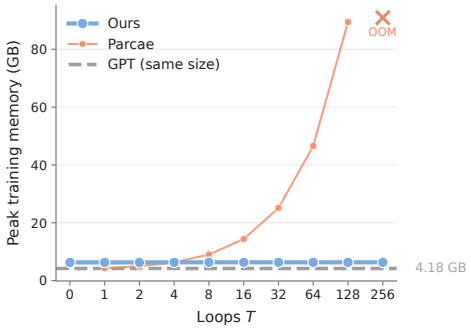

Figure 4. Training memory requirement varying the number of loops/iterations.

4.2. Attractor Models Lead to Significant Gains in Hard Reasoning Tasks with Tiny Models

Beyond language modeling, we evaluate on Sudoku-Extreme and Maze-Hard (Wang et al., 2025; Jolicoeur-Martineau, 2025): two challenging reasoning benchmarks, where standard Transformers and most frontier LLMs fail.

Setup. We train small models with 1000 training examples for each task and require predicting the full output grid in a single direct forward pass (no autoregressive decoding). We follow the TRM training protocol (Jolicoeur-Martineau, 2025), using deep-supervision steps.

Method. TRM carries two latents across deep-supervision steps, a current answer y and a reasoning state z, and applies a tiny two-layer network $T(n+1)$ times per step to update them. We keep this protocol except for the inner update: instead of unrolling $T(n+1)$ applications, we solve directly for the fixed point of the (y, z) update with our solver. The initialization is handled by deep supervision itself: the previous step's (y, z) initializes the solver at the next step, with a learned embedding at step zero, so we do not use a separate backbone.

We present our results in Table 2. The fixed-depth Transformer fails on both tasks. HRM (27M) achieves 55.0% / 74.5%. TRM is the strongest tiny baseline at 7M (74.7% / 85.3%), but (counter to the goal of scaling) collapses to 0% on both tasks when scaled to 27M parameters. Our model scales naturally with parameter count. We attribute the difference to the explicit fixed-point objective, which appears to provide regularization that bare iterative refinement lacks at higher capacity.

Table 2. Small-sample training results on Sudoku-Extreme and Maze-Hard.

Method	# Params	Sudoku ↑	Maze ↑
Deepseek R1	671B	0.0%	0.0%
Claude 3.7	?	0.0%	0.0%
O3-mini-high	?	0.0%	0.0%
Transformer	27M	0.0%	0.0%
HRM	27M	55.0%	74.5%
TRM	7M	74.7%	85.3%
TRM	27M	0.0%	0.0%
Ours	7M	54.3%	46.7%
Ours	27M	**91.4%**	**93.1%**

For the backward pass we use the phantom-gradient scheme (Geng et al., 2022) instead of the one-step approximation used in the language-modeling experiments. We find this necessary at the parameter counts considered here. The very small networks and training sets (∼1,000 examples) are too sensitive to the one-step approximation, consistent with TRM's report that switching to one-step drops Sudoku-Extreme accuracy from 87.4% to 56.5% (Jolicoeur-Martineau, 2025).

This setup is still an instance of our Attractor Model framework, where an output-space representation is iteratively refined to an equilibrium, decoded, and differentiated through implicitly. The only difference is the initialization mechanism. Rather than a Transformer backbone producing $\tilde{y}_0$, deep supervision supplies the initialization. Specifically, the previous supervision step's (y, z) initializes the solver at the next step, with a learned embedding at step zero.

4.3. Equilibrium Internalization and Test-Time Behavior

Fixed-point convergence. Figures 5 and 7 visualize convergence in two complementary ways. First, we project the trajectory of the final-position representation onto its first two principal components over 16 iterations. Our method rapidly contracts to a fixed point: iterations 8–16 collapse onto a single attractor. Parcae also moves toward a stable state, but does so more slowly; its trajectory remains noisier and continues to drift through later recurrences. This suggests that while finite-loop language models can exhibit fixed-point-like behavior, explicitly training the loop as a fixed-point problem produces faster and cleaner convergence. Second, we track the number of solver iterations required during

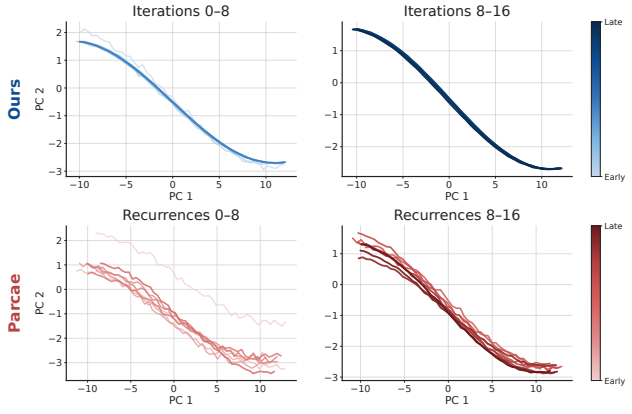

Figure 5. PCA projection of the residual stream at the final sequence position over 16 iterations.

training. The DEQ baseline requires increasingly many iterations as optimization progresses, consistent with the observations in the original work (Bai et al., 2019). In contrast, Attractor Models rapidly converge to the minimum iteration count and remain stable. This behavior provides evidence for equilibrium internalization, where optimization shifts work from the iterative solver into the backbone proposal.

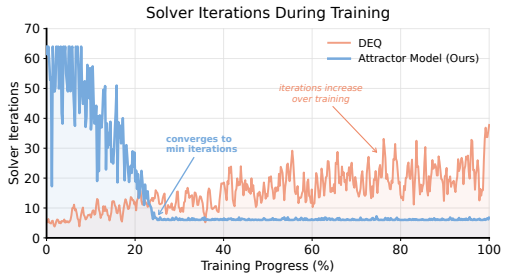

Figure 7. The DEQ baseline requires increasingly many solver iterations during training, whereas ours rapidly settles to the minimum iteration count and remains stable.

Test-time iterations vs. quality. In Figure 6, we report validation perplexity, CORE accuracy, and CORE-Extended accuracy as we vary the number of inference iterations T while holding training fixed. For Parcae, quality improves monotonically from $T = 1$ until it plateaus near $T = 8$, indicating that the model relies on repeated test-time refinement. In contrast, our method reaches peak performance at $T = 1$ at every scale, and even $T = 0$ (decoding the backbone proposal $\tilde{y}_0$ directly through the tied unembedding) is already at or near the converged value.

We attribute this behavior to *equilibrium internalization*: during training, the backbone learns to produce a proposal $\tilde{y}_0$ that already lies close to the fixed point that the solver would otherwise compute through iteration. Thus, the iterative block still shapes the learned representation during training, but at inference time the model has largely internalized the result of that refinement into its initialization. Strikingly, the backbone-only readout at $T = 0$ is already stronger than larger standard Transformers trained without attractor assistance, and matches or exceeds finite-loop baselines that require many test-time recurrences. Concretely, our 140M model at $T = 0$ matches Parcae 140M at $T = 8$; the same pattern holds at 370M and 770M.

Takeaway: Equilibrium Internalization

Attractor Models learn to initialize a prediction near the fixed point, so inference is effectively converged before any explicit refinement step. As a result, they preserve the benefits of iterative computation during training while avoiding the need for many sequential iterations at test time, unlike standard looped models whose quality depends on repeated inference-time updates.

4.4. Ablations

For the ablations, we use a 60.3M-parameter version of our model trained on 1B tokens of FineWeb-Edu under the same `nanochat` configuration. All ablation runs share the same data stream, tokenizer, and optimizer; only the ablated component varies.

We isolate the contribution of (i) the location of the equilibrium and its initialization (Table 3), (ii) the additive injection

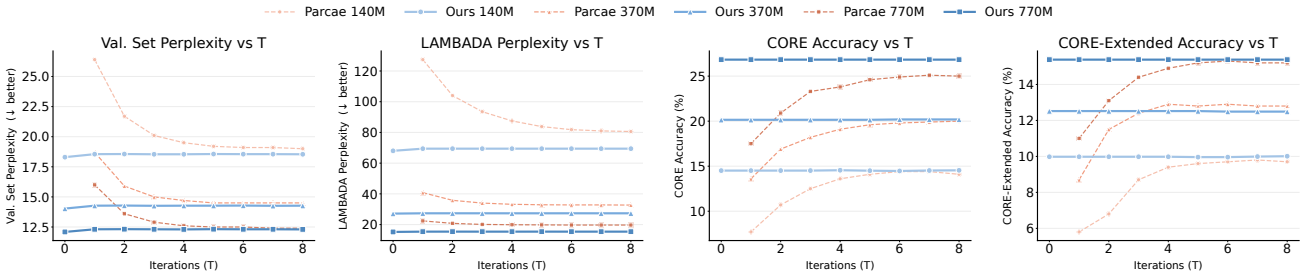

Figure 6. Validation perplexity on FineWeb-Edu, Lambada, CORE accuracy, and CORE-Extended accuracy as a function of test-time iterations T, at three scales. Parcae's quality improves monotonically up to $T \approx \mu_{\text{rec}} = 8$. Our method is essentially converged at $T = 1$ (and at $T = 0$ for the 770M model), because the solver is initialized at $\tilde{y}_0$, which is itself a coherent prediction.

Table 3. Comparison to DEQ at matched parameter count.

Method	Size	Equilibrium	Val. PPL ↓	Avg. Iters ↓	Core ↑
DEQ (Bai et al., 2019)	60.3M	hidden ($z_0 = 0$, sep. head)	42.18	14.6	5.21 ± 0.14
DEQ + tied unemb.	60.3M	hidden ($z_0 = 0$)	38.74	13.9	5.83 ± 0.12
Attractor Model	60.3M	output emb. ($\tilde{y}_0 = \mathcal{T}_{\theta_b}(E(x))$)	**34.05**	**8.4**	$\mathbf{6.74 \pm 0.10}$

of c (Table 4), (iii) and the one-step backward approximation (Table 5). All ablations train a 60.3M-parameter model on 1B tokens of FineWeb-Edu under the same `nanochat` setup, with all other hyperparameters fixed.

Comparison to DEQ. Table 3 compares Attractor Models against a parameter-matched DEQ (Bai et al., 2019). The DEQ baseline solves for a hidden-state equilibrium initialized from an uninformative state and decodes it with a separate output head. Tying the unembedding closes part of the gap (Val. PPL $42.18 \rightarrow 38.74$), but the largest improvement comes from matching the equilibrium to the design of Attractor Models: the fixed point is placed directly in the tied output-embedding space and the root finder is initialized from the backbone proposal $\tilde{y}_0 = \mathcal{T}_{\theta_b}(E(x))$ ($\rightarrow 34.05$).

This proposal gives the solver a meaningful, input-dependent starting point rather than requiring the recurrent block to construct the representation from scratch. As a result, the Attractor Model reaches the same residual tolerance in $1.7\times$ fewer iterations while also achieving lower perplexity. We view this as evidence of *equilibrium internalization*: the backbone learns to produce an output-side proposal $\tilde{y}_0$ that already lies close to the eventual equilibrium $\tilde{y}^\star$, making the subsequent attractor refinement both easier and faster.

Proposal injection. Table 4 ablates how the backbone proposal $\tilde{y}_0$ is provided to the attractor. We use *proposal injection* to denote how $\tilde{y}_0$ enters the recurrent refinement map, and *persistent injection* to denote providing $\tilde{y}_0$ at every refinement step rather than only through the initial state.

When the proposal is used only for initialization, $\tilde{y}_{t=0} = \tilde{y}_0$ and $\tilde{y}_{t+1} = \mathcal{T}_{\theta_a}(\tilde{y}_t)$, the recurrent update no longer depends on the input after the first state. Consequently, the fixed point can become proposal-independent, and only 12.4% of validation tokens converge within $T_{\max}$. Persistent injection by concatenation, $\tilde{y}_{t+1} = \mathcal{T}_{\theta_a}([\tilde{y}_t; \tilde{y}_0])$, restores proposal-dependence and recovers much of the quality, but makes the refinement problem harder: convergence is slower (11.2 vs. 8.4 average iterations) and perplexity is worse (36.81 vs. 34.05). Additive proposal injection, $\tilde{y}_{t+1} = \mathcal{T}_{\theta_a}(\tilde{y}_t, \tilde{y}_0)$, provides the backbone proposal at every step while keeping the refinement map simple and well-conditioned, yielding the best results across all three metrics.

Backward pass. In table 5, we compare the one-step backward approximation against full implicit differentiation (Anderson on the linear system $(I - J_{\tilde{y}}^\top)u = v$) and the phantom gradient (Geng et al., 2022) unroll. Full IFT improves PPL by only 0.14 while increasing peak training memory by $4.8\times$ and step time by $2.7\times$; phantom gradient lies in between. The one-step approximation allows relatively cheap training cost while maintaining nearly all of the original performance of a full backward gradient computation.

Table 4. Effect of input injection on convergence and quality.

Injection of c	Val. PPL ↓	Avg. Iters ↓	Converge ↑
Initial only	51.92	$T_{\max}$	12.4%
Concatenation	36.81	11.2	88.6%
Additive (ours):	**34.05**	**8.4**	**99.7%**

Table 5. Ablation for running the full IFT gradients, phantom gradients, and the one-step.

Backward Pass	Val. PPL ↓	Train Mem. ↓	Step Time ↓	Δ vs. Full
Full IFT (Anderson on u)	33.91	$4.8\times$	$2.7\times$	—
Phantom gradient (k=3)	34.02	$1.8\times$	$1.4\times$	+0.11
Ours: one-step ($u \approx v$)	**34.05**	$\mathbf{1.0\times}$	$\mathbf{1.0\times}$	+0.14

5. Conclusion and Future Work

In this work, we propose Attractor Models, a new family of architectures that first produce meaningful prediction embeddings and then refines them through an attractor module by solving for a fixed point. This formulation makes recurrent refinement stable, memory-efficient, and adaptive, while avoiding the cost of explicit unrolling and achieving strong results across both large-scale language modeling and hard reasoning with tiny models. In stark contrast to prior looped language models, training Attractor Models gives rise to equilibrium internalization: the model learns to make the very refinement procedure that trained it largely unnecessary at inference time. An interesting direction for future work is to further study the equilibrium internalization phenomenon and the differences between Attractor Models and finite-loop recurrence.

References

Anderson, D. G. Iterative procedures for nonlinear integral equations. *Journal of the ACM*, 12(4):547–560, 1965. doi: 10.1145/321296.321305.

Bae, S., Kim, Y., Bayat, R., Kim, S., Ha, J., Schuster, T., Fisch, A., Harutyunyan, H., Ji, Z., Courville, A., and Yun, S.-Y. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation, 2025. URL https://arxiv.org/abs/2507.10524.

Bai, S., Kolter, J. Z., and Koltun, V. Deep equilibrium models. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 2019.

Bai, S., Koltun, V., and Kolter, J. Z. Multiscale deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/3812f9a59b634c2a9c574610eaba5bed-Abstract.html.

Bai, S., Koltun, V., and Kolter, J. Z. Stabilizing equilibrium models by jacobian regularization, 2021. URL https://arxiv.org/abs/2106.14342.

Bakouch, E., Patiño, C. M., Lozhkov, A., Beeching, E., Roucher, A., Tazi, N., Reedi, A. J., Penedo, G., Kydlicek, H., Fourrier, C., Habib, N., Rasul, K., Gallouédec, Q., Larcher, H., Morlon, M., Joshua, Srivastav, V., Nguyen, X.-S., Raffel, C., Tunstall, L., Ben Allal, L., von Werra, L., and Wolf, T. Smollm3: smol, multilingual, long-context reasoner. https://huggingface.co/blog/smollm3, July 2025. Hugging Face blog post.

Blayney, H., Álvaro Arroyo, Obando-Ceron, J., Castro, P. S., Courville, A., Bronstein, M. M., and Dong, X. A mechanistic analysis of looped reasoning language models, 2026. URL https://arxiv.org/abs/2604.11791.

Chen, Y., Shang, J., Zhang, Z., Xie, Y., Sheng, J., Liu, T., Wang, S., Sun, Y., Wu, H., and Wang, H. Inner thinking transformer: Leveraging dynamic depth scaling to foster adaptive internal thinking, 2025. URL https://arxiv.org/abs/2502.13842.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Łukasz Kaiser. Universal transformers, 2019. URL https://arxiv.org/abs/1807.03819.

Fan, Y., Du, Y., Ramchandran, K., and Lee, K. Looped transformers for length generalization, 2025. URL https://arxiv.org/abs/2409.15647.

Fu, T., You, Y., Chen, Z., Dai, G., Yang, H., and Wang, Y. Think-at-hard: Selective latent iterations to improve reasoning language models, 2026. URL https://arxiv.org/abs/2511.08577.

Fung, S. W., Heaton, H., Li, Q., McKenzie, D., Osher, S., and Yin, W. Jfb: Jacobian-free backpropagation for implicit networks, 2021. URL https://arxiv.org/abs/2103.12803.

Geiping, J., McLeish, S., Jain, N., Kirchenbauer, J., Singh, S., Bartoldson, B. R., Kailkhura, B., Bhatele, A., and Goldstein, T. Scaling up test-time compute with latent reasoning: A recurrent depth approach, 2025a. URL https://arxiv.org/abs/2502.05171.

Geiping, J., Yang, X., and Su, G. Efficient parallel samplers for recurrent-depth models and their connection to diffusion language models, 2025b. URL https://arxiv.org/abs/2510.14961.

Geng, Z., Zhang, X.-Y., Bai, S., Wang, Y., and Lin, Z. On training implicit models, 2022. URL https://arxiv.org/abs/2111.05177.

Graves, A. Adaptive computation time for recurrent neural networks, 2017. URL https://arxiv.org/abs/1603.08983.

Hao, S., Sukhbaatar, S., Su, D., Li, X., Hu, Z., Weston, J., and Tian, Y. Training large language models to reason in a continuous latent space, 2025. URL https://arxiv.org/abs/2412.06769.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Inan, H., Khosravi, K., and Socher, R. Tying word vectors and word classifiers: A loss framework for language modeling, 2017. URL https://arxiv.org/abs/1611.01462.

Jeddi, A., Ciccone, M., and Taati, B. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation, 2026. URL https://arxiv.org/abs/2602.11451.

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

Karpathy, A. nanochat: The best chatgpt that $100 can buy, 2025. URL https://github.com/karpathy/nanochat.

Krantz, S. G. and Parks, H. R. *The Implicit Function Theorem: History, Theory, and Applications*. Birkhäuser, Boston, MA, 2002. ISBN 978-0-8176-4285-3. doi: 10.1007/978-1-4612-0059-8.

Labovich, A. Stability and generalization in looped transformers, 2026. URL https://arxiv.org/abs/2604.15259.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, N. Q., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

Penedo, G., Kydlíček, H., allal, L. B., Lozhkov, A., Mitchell, M., Raffel, C., Werra, L. V., and Wolf, T. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

Prairie, H., Novack, Z., Berg-Kirkpatrick, T., and Fu, D. Y. Parcae: Scaling laws for stable looped language models, 2026. URL https://arxiv.org/abs/2604.12946.

Press, O. and Wolf, L. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 157–163, 2017.

Sapunov, G. Universal transformers need memory: Depth-state trade-offs in adaptive recursive reasoning, 2026. URL https://arxiv.org/abs/2604.21999.

Saunshi, N., Dikkala, N., Li, Z., Kumar, S., and Reddi, S. J. Reasoning with latent thoughts: On the power of looped transformers, 2025a. URL https://arxiv.org/abs/2502.17416.

Saunshi, N., Dikkala, N., Li, Z., Kumar, S., and Reddi, S. J. Reasoning with latent thoughts: On the power of looped transformers, 2025b. URL https://arxiv.org/abs/2502.17416.

Schwethelm, K., Rueckert, D., and Kaissis, G. How much is one recurrence worth? iso-depth scaling laws for looped language models, 2026. URL https://arxiv.org/abs/2604.21106.

Teoh, J., Tomar, M., Ahn, K., Hu, E. S., Sharma, P., Islam, R., Lamb, A., and Langford, J. Next-latent prediction transformers learn compact world models, 2025. URL https://arxiv.org/abs/2511.05963.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pp. 5998–6008, 2017.

Wang, G., Li, J., Sun, Y., Chen, X., Liu, C., Wu, Y., Lu, M., Song, S., and Yadkori, Y. A. Hierarchical reasoning model, 2025. URL https://arxiv.org/abs/2506.21734.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Wei, X., Liu, X., Zang, Y., Dong, X., Cao, Y., Wang, J., Qiu, X., and Lin, D. Sim-cot: Supervised implicit chain-of-thought, 2025. URL https://arxiv.org/abs/2509.20317.

Xu, K. and Sato, I. On expressive power of looped transformers: Theoretical analysis and enhancement via timestep encoding, 2025. URL https://arxiv.org/abs/2410.01405.

Xu, K. and Sato, I. A formal comparison between chain of thought and latent thought, 2026. URL https://arxiv.org/abs/2509.25239.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Yang, L., Lee, K., Nowak, R., and Papailiopoulos, D. Looped transformers are better at learning learning algorithms, 2024a. URL https://arxiv.org/abs/2311.12424.

Yang, L., Lee, K., Nowak, R., and Papailiopoulos, D. Looped transformers are better at learning learning algorithms, 2024b. URL https://arxiv.org/abs/2311.12424.

Zhu, R.-J., Wang, Z., Hua, K., Zhang, T., Li, Z., Que, H., Wei, B., Wen, Z., Yin, F., Xing, H., Li, L., Shi, J., Ma, K., Li, S., Kergan, T., Smith, A., Qu, X., Hui, M., Wu, B., Min, Q., Huang, H., Zhou, X., Ye, W., Liu, J., Yang, J., Shi, Y., Lin, C., Zhao, E., Cai, T., Zhang, G., Huang, W., Bengio, Y., and Eshraghian, J. Scaling latent reasoning via looped language models, 2025. URL https://arxiv.org/abs/2510.25741.

A. Related Work

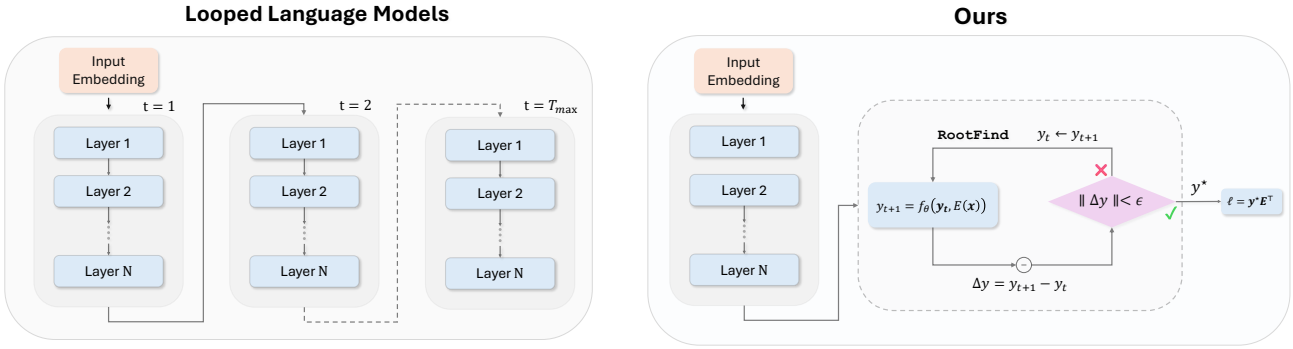

Figure 8. Comparison of standard looped language models vs. our architecture

Looped and recurrent language models. Weight-tied recurrence has re-emerged as an alternative to deeper feed-forward stacks. Universal Transformers (Dehghani et al., 2019) share a single block across depth, while looped language models (Zhu et al., 2025; Prairie et al., 2026; Sapunov, 2026; Yang et al., 2024a; Labovich, 2026; Saunshi et al., 2025a; Xu & Sato, 2025; Saunshi et al., 2025b; Geiping et al., 2025a; Bae et al., 2025) iterate a recurrent update on a hidden state to enable latent reasoning. Coconut (Hao et al., 2025) and similar latent reasoning methods (Teoh et al., 2025; Chen et al., 2025; Xu & Sato, 2026; Jeddi et al., 2026; Fu et al., 2026; Geiping et al., 2025b) perform reasoning in continuous representation space rather than through chain-of-thought tokens. These approaches unroll the loop for a fixed number of steps at training time, causing training memory to grow linearly in depth, coupling inference iterations to training, and often degrading quality when iterations are extended at test time (Zhu et al., 2025). Mechanistic analysis (Blayney et al., 2026) shows that the recurrent updates of looped LMs in fact converge to a fixed point for the vast majority of tokens, directly motivating our formulation.

Implicit fixed-point models. Deep Equilibrium Models (DEQs) (Bai et al., 2019) replace finite unrolling with a fixed-point equation $z^\star = f_\theta(z^\star, x)$ and train through it via implicit differentiation, decoupling effective depth from training memory. Solvers typically use Anderson acceleration (Anderson, 1965), with backward passes ranging from full implicit differentiation to cheap surrogates such as one-step (Fung et al., 2021) and phantom (Geng et al., 2022) gradients. Standard DEQs solve for a hidden state initialized from zero and decode through a separate output head. In Attractor Models, the equilibrium instead lives in the tied output-embedding space (Press & Wolf, 2017; Inan et al., 2017) and is warm-started from a meaningful backbone prediction; we find this structure essential for stable training at language-modeling scale and show it gives rise to equilibrium internalization (Section 4.3).

Tiny recursive reasoners. On small-data algorithmic benchmarks such as Sudoku and maze solving, hierarchical and tiny recursive networks (HRM (Wang et al., 2025), TRM (Jolicoeur-Martineau, 2025)) achieve strong accuracy with few

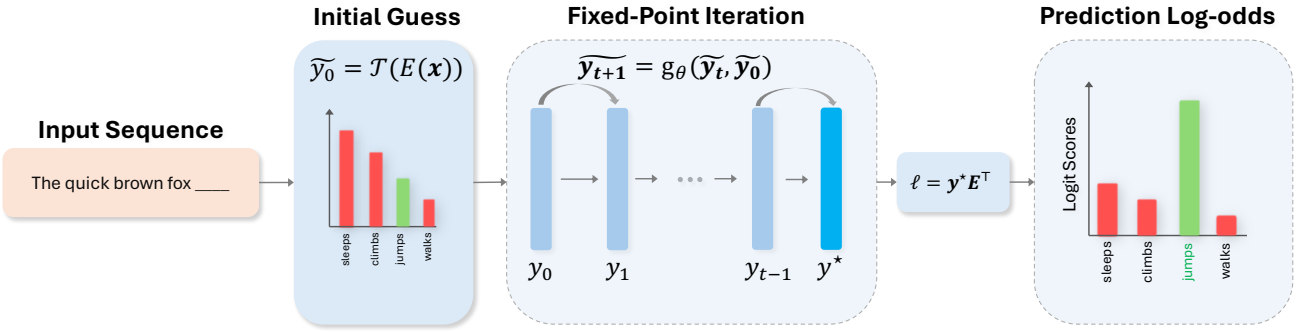

Figure 9. Overview of our method. The backbone maps the input embeddings to an output-side proposal $\tilde{y}_0 = \mathcal{T}_{\theta_b}(E(x))$. The attractor module then refines this proposal through the proposal conditioned iteration $\tilde{y}_{t+1} = \mathcal{T}_{\theta_a}(\tilde{y}_t, \tilde{y}_0)$ until convergence to an equilibrium $\tilde{y}^\star$ or a maximum number of solver steps. The equilibrium is decoded with the tied unembedding $\tilde{y}^\star E^\top$.

parameters, but exhibit a "less is more" behavior where performance collapses as model size grows. Attractor Models retain the iterative-refinement benefit of these architectures while scaling cleanly with parameter count).

Positioning. Looped LMs couple three quantities through unrolled BPTT: inference depth, training depth, and training memory. Attractor Models decouple all three: the equilibrium is defined by a residual equation in output-embedding space, the number of solver evaluations is chosen adaptively by tolerance ε, and training memory in the recurrent block is constant in effective depth. Empirically, this combines the strengths of feed-forward and recurrent models—better perplexity than parameter-matched Transformers, 25–31% lower training FLOPs and constant memory relative to looped LMs, and clean scaling on hard reasoning where specialized recursive networks collapse.

B. Theory

B.1. Algorithm Pseudocode

Algorithm 1 Training Attractor Models.

Require: input tokens x, parameters $\theta = (\theta_a, \theta_b, E)$, tolerance ε, max iterations $T_{\max}$

// Forward pass
$\tilde{x} \leftarrow E(x)$ {tied token embedding}
$\tilde{y}_0 \leftarrow \mathcal{T}_{\theta_b}(\tilde{x})$ {backbone proposal / initialization}
Define $\mathcal{A}_{\theta_a}(\tilde{y}; \tilde{y}_0) \leftarrow \mathcal{T}_{\theta_a}(\tilde{y}, \tilde{y}_0) - \tilde{y}$
$\tilde{y}^\star \leftarrow \texttt{RootFind}(\mathcal{A}_{\theta_a}(\cdot; \tilde{y}_0);\ \tilde{y}_0,\ \varepsilon,\ T_{\max})$
$p \leftarrow \mathsf{Softmax}(\tilde{y}^\star E^\top)$ {tied unembedding and softmax}
// Backward pass, given $v = \partial \mathcal{L}/\partial \tilde{y}^\star$
solve $\left(I - J_{\tilde{y}}^\top\right) u = v$ via $\texttt{RootFind}\big(u' \mapsto (I - J_{\tilde{y}}^\top)\, u' - v;\ u_0 = v\big)$
$\partial \mathcal{L}/\partial \theta \leftarrow u^\top \partial \mathcal{T}_{\theta_a}(\tilde{y}^\star, \tilde{y}_0)/\partial \theta$

B.2. Well-posedness and the implicit gradient

Fix an input x and parameters θ. Throughout this section, we write $F(y) \triangleq f_\theta(y, E(x))$ for brevity, so that the fixed-point equation $y^\star = f_\theta(y^\star, E(x))$ becomes $y^\star = F(y^\star)$. Let $J_F(y) \triangleq \partial F/\partial y$ denote the Jacobian of F with respect to its state input. We work in a generic norm $\|\cdot\|$ on $\mathbb{R}^{L\times d}$ and its induced operator norm; for concreteness, can take both to be Frobenius and spectral, respectively. Assumptions are borrowed from (Bai et al., 2020; Krantz & Parks, 2002).

Assumption B.1 (Local contraction)**.** There exist a point $\bar{y} \in \mathbb{R}^{L\times d}$, a radius $r > 0$, and a constant $L \in [0, 1)$ such that

1. F maps the closed ball $B_r(\bar{y}) \triangleq \{y : \|y - \bar{y}\| \le r\}$ into itself, and
2. F is L-Lipschitz on $B_r(\bar{y})$:

$$\|F(y) - F(y')\| \ \le\ L\,\|y - y'\| \qquad \text{for all } y, y' \in B_r(\bar{y}).$$

When F is continuously differentiable, a sufficient form of (ii) is $\sup_{y\in B_r(\bar{y})} \|J_F(y)\| \le L$.

Theorem B.2 (Well-posedness and implicit gradient)**.** *Under Assumption B.1, the following hold.*

1. Existence and uniqueness: *There is a unique $y^\star \in B_r(\bar{y})$ with $y^\star = F(y^\star)$.*
2. The picard iteration converges linearly: *For any $y_0 \in B_r(\bar{y})$, the iterates $y_{k+1} = F(y_k)$ remain in $B_r(\bar{y})$ and satisfy*

$$\|y_k - y^\star\| \ \le\ L^k\,\|y_0 - y^\star\|. \tag{1}$$

3. Validity of the implicit gradient: *If F is continuously differentiable on $B_r(\bar{y})$, then $I - J_F(y^\star)$ is invertible, $y^\star$ depends continuously differentiably on θ in a neighborhood of the current parameters, and*

$$\frac{\partial y^\star}{\partial \theta} \ =\ \big(I - J_F(y^\star)\big)^{-1}\ \frac{\partial f_\theta}{\partial \theta}\bigg|_{y^\star}. \tag{2}$$

Proof. Parts (i) and (ii) follow from the Banach fixed-point theorem applied to the contraction F on the closed (hence complete) ball $B_r(\bar{y}) \subset (\mathbb{R}^{L\times d}, \|\cdot\|)$.

For (iii), Lipschitz continuity with constant L gives $\|J_F(y^\star)\| \le L < 1$, so the spectral radius satisfies $\rho(J_F(y^\star)) \le \|J_F(y^\star)\| < 1$. The Neumann series $\sum_{k\ge 0} J_F(y^\star)^k$ therefore converges to $(I - J_F(y^\star))^{-1}$, and in particular $I - J_F(y^\star)$ is invertible. Define $G(y,\theta) \triangleq f_\theta(y, E(x)) - y$, which is C^1 in both arguments and satisfies $G(y^\star,\theta) = 0$ with

$$\left.\frac{\partial G}{\partial y}\right|_{y^\star} \;=\; J_F(y^\star) - I, \tag{3}$$

which is invertible. The implicit function theorem applied to $G = 0$ yields a unique C^1 map $\theta \mapsto y^\star(\theta)$ in a neighborhood of the current parameters, with

$$\frac{\partial y^\star}{\partial \theta} \;=\; -\left(\frac{\partial G}{\partial y}\right)^{-1}\frac{\partial G}{\partial \theta} \;=\; \big(I - J_F(y^\star)\big)^{-1}\left.\frac{\partial f_\theta}{\partial \theta}\right|_{y^\star}. \tag{4}$$

□

Equation (2) is exactly the gradient computed in the backward pass of Section 3. The linear solve is convergent for the same reason: since $\|J_F^\top(y^\star)\| = \|J_F(y^\star)\| \le L < 1$, the Neumann iteration $u \leftarrow J_F^\top(y^\star)\,u + v$ converges geometrically with rate L to the unique solution $u = (I - J_F^\top(y^\star))^{-1}v$.

B.3. Looped language models are fixed-point iterators

A LoopLM (Zhu et al., 2025) of depth T with the same weight-tied block f_θ and warm start $y_0 = E(x)$ produces

$$y_T^{\text{loop}} \;\triangleq\; F^{(T)}\big(E(x)\big) \;=\; \underbrace{F\circ F\circ\cdots\circ F}_{T\text{ times}}\big(E(x)\big). \tag{5}$$

This is exactly T Picard iterations of our residual $g_\theta(\cdot, x)$ warm-started from the input embedding. See the following corollary:

Corollary B.3 (LoopLM as a truncated approximation)**.** *Suppose Assumption B.1 holds and the warm start $E(x) \in B_r(\bar{y})$. Then for every $T \ge 0$,*

$$\|y_T^{\text{loop}} - y^\star\| \;\le\; L^T\,\|E(x) - y^\star\|. \tag{6}$$

In particular, $y_T^{\text{loop}} \to y^\star$ as $T \to \infty$, and the discrepancy between a LoopLM of depth T and our model decays geometrically in T.

Proof. Apply Theorem B.2(ii) with $y_0 = E(x)$. □

Two consequences are worth highlighting:

The limit: Our fixed-point model is the $T \to \infty$ limit of LoopLM with shared parameters. Training a LoopLM at any finite depth T implicitly trains an approximation to the same equilibrium $y^\star$, with the approximation error controlled by (6).

The iteration count at inference: Picard iteration converges at the linear rate L. Our forward pass uses Anderson acceleration, which under standard regularity conditions converges *superlinearly* near $y^\star$ (Anderson, 1965). To reach a target residual tolerance ε, the root finder therefore typically requires fewer evaluations of f_θ than the unroll depth T a LoopLM would need for a comparable residual. The exact crossover depends on L and on the solver hyperparameters.

Caveat: Assumption B.1 is local and is not guaranteed to hold for an arbitrarily trained transformer block. In our experiments, we generally notice that $\|J_F(y^\star)\| \le L < 1$, the authors of DEQ also find that using a regularizer on the Jacobian to enforce this helps (Bai et al., 2021).

B.4. Where looped language models fall short

B.4.1. Attractor Models

The initialization $\tilde{y}_0$ and the fixed point $\tilde{y}^\star$ live in the same tied output-embedding space, so either can be decoded by the unembedding $E^\top$. The loss depends only on $\tilde{y}^\star$, and the implicit gradient with respect to the backbone parameters is

$$\nabla_{\theta_b}\mathcal{L} \;=\; u^\top J_{\tilde{y}_0}\,\nabla_{\theta_b}\tilde{y}_0, \qquad u = (I - J_{\tilde{y}}^\top)^{-1}v.$$

Thus, the loss may be viewed as a function of the backbone's output embedding: perturbations in $\tilde{y}_0$ affect $\mathcal{L}$ by perturbing $\tilde{y}^\star$ in the same space. The backbone is therefore optimized as a standalone next-token predictor whose embedding is decoded by $E^\top$.

When the backbone has strictly greater capacity than the weight-tied attractor, the joint optimum places the prediction work in the backbone. Consequently, $\tilde{y}_0(x;\theta_b^\star)$ approaches the loss-minimizing embedding. The fixed-point constraint

$$\mathcal{T}_{\theta_a^\star}(\tilde{y}^\star, \tilde{y}_0) = \tilde{y}^\star$$

is then consistent only when $\tilde{y}^\star = \tilde{y}_0$, yielding

$$\mathcal{T}_{\theta_a^\star}(\tilde{y}_0, \tilde{y}_0) = \tilde{y}_0.$$

B.4.2. Standard looped LMs

In standard looped language models, the latent initialization h_0 is uninformative, typically zero or noise, and occupies the role of a hidden state rather than an output embedding. Decoding $h_0E^\top$ is therefore not meaningful.

Gradients instead flow by backpropagation through time along the unrolled trajectory $(h_0, \ldots, h_T)$. There is no term in this gradient that drives h_0 toward the loss-bearing final state h_T in embedding distance: the initialization is fixed by the architecture and is not produced by a separately trained predictor. □

B.4.3. Implicit-Gradient Barrier

The implicit gradient

$$\nabla_\theta\mathcal{L}_\infty = v^\top(I - J_g^\top)^{-1}\partial_\theta g$$

contains the factor $(I - J_g)^{-1}$. This inverse is undefined when $1 \in \sigma(J_g)$, and its operator norm scales as

$$\|(I - J_g)^{-1}\| \sim \operatorname{dist}(1, \sigma(J_g))^{-1}$$

near such a singularity.

Along any continuous gradient-descent path, the eigenvalues of $J_g(y^\star;\theta)$ vary continuously. Therefore, for the dominant eigenvalue to leave the unit disk through $+1$—the canonical loss-of-contraction route for residual iteration maps—it must approach 1. This forces $\|(I - J_g)^{-1}\| \to \infty$.

Unless v is orthogonal to the offending eigendirection, a codimension-one and hence non-generic condition, the gradient norm diverges. The descent step therefore blows up before the boundary can be crossed, confining θ to the contractive region $\{\rho(J_g) < 1\}$.

For exits through complex eigenvalues, the same conclusion holds under the one-step JFB approximation, since the truncated Neumann series

$$\sum_k (J_g^\top)^k$$

diverges as $\rho(J_g) \to 1$. □

B.4.4. Interpretation

Fixed-loop training has no inherent mechanism that favors contractive iterations. An optimum in which g_θ perturbs the embedding for K steps and only happens to land accurately at step K is a valid fixed-loop solution. These are precisely the solutions that fail when extra loops are run at inference.

Equilibrium training cannot reach such solutions, because the implicit gradient creates a barrier of diverging gradients around non-contractive regions. As a result, the trajectory of θ is confined to the regime in which the solver converges, the one-step gradient is a descent direction, and additional iterations remain stable.

The mechanistic observation (Blayney et al., 2026) that fixed points emerge only late in standard looped training is consistent with this picture: the loss landscape contains a basin near contractive solutions, but only equilibrium training applies pressure toward that basin from the beginning of training.

C. Hyperparamter and Experimental Settings

We use NVIDIA H200 GPUs for all of our experiments.

Table 6. Architecture hyperparameters for each model family and scale.

	Transformer			**Parcae**			**Attractor Model**		
Hyperparameter	140M	370M	770M	140M	370M	770M	140M	370M	770M
d_{model}	768	1024	1280	768	1024	1280	1024	1280	1280
d_{ff}	3072	4096	5120	3072	4096	5120	4096	5120	5120
Attention heads	6	8	10	6	8	10	8	10	10
Total layers	6	12	18	6	12	18	8	17	37
Prelude	—	—	—	2	4	6	7	15	35
Core (recurrent)	—	—	—	2	4	6	1	2	2
Coda	—	—	—	2	4	6	0	0	0
Mean recurrence T	1	1	1	8	8	8	(solver-determined)		
Sequence length	2048								
Vocab size	32768								
Tie embeddings	✓								
Norm	RMSNorm ($\epsilon = 10^{-5}$)								
Activation	ReLU^2								
QK-norm	✓								
Precision	bf16-mixed								

Table 7. Fixed-point solver hyperparameters for Attractor Model. Parcae and Transformer do not use a solver.

Hyperparameter	**140M**	**370M**	**770M**
Forward solver	Anderson	Anderson	Anderson
Max iterations (fwd)	64	64	32
Min iterations (fwd)	6	8	6
Tolerance (fwd)	3×10^{-4}	2×10^{-4}	1.5×10^{-4}
Anderson m	5	5	5
Anderson β	1.0	1.0	0.96
Backward type	one-step IFT	one-step IFT	Anderson
Max iterations (bwd)	64	64	32
Min iterations (bwd)	6	6	6
Tolerance (bwd)	3×10^{-4}	2×10^{-4}	1.5×10^{-4}
Adjoint grad clip	—	—	2.0
Layer-scale init (γ_0)	0.75	0.75	0.73
$\gamma_{\max}$	0.75	1.0	0.9
FP LR scale	0.5	0.4	0.4
FP weight decay	0.1	0.1	0.1

Table 8. Training hyperparameters. All three model families share the same optimizer, schedule, and data configuration at each scale. Differences are noted where they occur.

Hyperparameter	**140M**	**370M**	**770M**
Training tokens	11.2B	29.6B	61.6B
Dataset	FineWeb-Edu 100B (shuffled)		
Tokenizer	BPE, 32768 vocab		
Optimizer	MuonAdamW		
AdamW LR	8×10^{-3}	8×10^{-3}	5–$6 \times 10^{-3\dagger}$
AdamW (β_1, β_2)	(0.8, 0.95)		
AdamW ϵ	10^{-10}		
AdamW weight decay	0		
Muon LR	8×10^{-3}	8×10^{-3}	6–$8 \times 10^{-3\dagger}$
Muon momentum	0.95		
Muon weight decay	0.2 (linear decay to 0)		
Muon NS steps	5		
LR schedule	Trapezoid (0% warmup, 50% cooldown)		
Min LR	0		
Gradient clipping	1.0		
Global batch size (tokens)	524K ($= 256 \times 2048$)		
Micro batch size	32	32	8–32†
Gradient checkpointing	✗	✗	model-dependent‡
`torch.compile`	✓*	✓*	✓*
Strategy	DDP		
Init strategy	scaled-zero + orthogonal		
Seed	42		

†At 770M, Attractor Model uses AdamW LR 5×10^{-3} and Muon LR 6×10^{-3}; Transformer/Parcae use 6×10^{-3} / 8×10^{-3}.
‡Gradient checkpointing enabled for 770M Attractor Model and Parcae; disabled for Transformer.
*`torch.compile` enabled for Transformer and Parcae; disabled for Attractor Model (implicit-gradient hooks are not compile-compatible).

Table 9. Parcae recurrence hyperparameters.

Hyperparameter	**140M**	**370M**	**770M**
Injection type	diagonal		
State init	like-init		
Mean recurrence	8	8	8
Mean backprop depth	4	4	4
Sampling scheme	poisson-truncated-full		
Iteration method	per-sequence		
Prelude norm	✓		