Dynamic Parameter Reuse Augments Reasoning via Latent Chain of Thought

Standard large language models often rely on massive parameter counts for their performance, utilizing each parameter only once per inference pass. This post explores the emerging paradigm of recursive structures, where models reuse parameters across time or depth to achieve greater computational efficiency and algorithmic generalization. We draw connections in the landscape of parameter reuse, from growing models via stacking to recursive looping, and postulate that these architectural priors act as a form of Latent Chain of Thought (LCoT), allowing models to reason in a continuous state space rather than a discrete one. By shifting towards deeper and dynamic computation, grown and recursive architectures offer a path toward improved reasoning in compact networks, ascending beyond scaling laws of standard architectures.

Contents

Reduce, Reuse, Recycle.
Recur, Refine, Reason.

Introduction

In the current AI era, progress has been dominated by scale: increase the model size and thus the cost in exchange for improved performance . In standard sequence modeling, exemplified by the transformer architecture , each learnable parameter is used once per processed token An exception is using the transpose of the embedding table to decode the final embeddings into token space, a technique common in some language models.. This “single-pass” paradigm effectively treats the network as a rigid, acyclic circuit. After standard training, this model cannot improve its own performance to achieve a more capable model: instead the model must be discared and a new one reinitialized in a larger architecture size with more parameters to learn from scratch. This is computationally and algorithmically inefficient.

Decoupling parameter count from computation, parameter reuse within the architecture allows the training process to occur in a reduced search space, forcing the model to learn generalizable functions within the repeated modules rather than memorizing location-specific heuristics. A classic example is Convolutional Neural Networks (CNNs) and, more broadly, equivariant neural networks in geometric deep learning. By sliding the same kernel across an image, CNNs enforce translational symmetry that unlocked new levels of computer vision performance beyond what was possible by scaling up previous architectures without such parameter reutilization.

We are now seeing a similar renaissance of reuse, both in architectural elements and in training methods, burgeoning in sequence modeling, but in different and thus far unconnected flavors. In the taxonomy of AI architectures, we focus this blog post on the order containing Transformers, State Space Models (SSMs), and (Time) Recurrent Architectures for sequence modeling. Broadly, such models process and generate sequences of data through a depth of layers, often autoregressively. These layers follow repeated block patterns across the depth but are individually parametrized. We will discuss how blocks of these layers may be looped, grown, and reused to unlock efficiency and, crucially, reasoning capabilities. The intent of this blogpost is to discuss the parallels of these flavors of recursion and related methods across various axes, both those which are inherent to architectural genuses such as time recursion as well as those which can augment any models within this order that follow the blocked layer structure. These flavors have thus far mostly only been considered in isolation, but exploring their analogues from a unified lens may unlock further synergies.

Reuse in Time vs. Depth

To understand recent advances in looping and reuse, we first distinguish between two orthogonal axes of recursion: recursion over the input sequence (Time) and recursion over the computation (Depth).

The Time Axis (RNNs and SSMs)

This is the classical domain of Recurrent Neural Networks (RNNs), including models containing Long Short-Term Memory (LSTMs) and Gated Recurrence Units (GRUs). Here, the model reuses the same function f_θ at every position in the text. A latent state h_{t-1}, representing a compressed memory of the past, is fed along with the current token x_t into the network to produce h_t (possibly in addition to an output y_t).

While Transformers largely overtook RNNs due to the latter's training inefficiencies such as the inability to parallelize, the concept has returned in State Space Models (SSMs) . These models offer the inference efficiency of RNNs, achieving linear time complexity in sequence length opposed to quadratic complexity of transformers, with the training parallelizability of Transformers. Hybrid models interleave time-recursive layers with attention-based layers throughout the depth of the architecture.

We note that the recursion here is strictly for memory compression. The model is still performing a fixed amount of compute per token. This limitation may be bypassed via Chain-of-Thought (CoT) , utilizable in any autoregressive model beyond just those that use explicit time recursion. By generating extra intermediate tokens, a model effectively uses the time axis to "buy" more compute, treating the generated sequence as an extended memory buffer for reasoning. We will revisit this concept later when discussing latent reasoning.

The Depth Axis (Looping)

Instead of moving forward in time, looped models portray recurrence in computation depth on the current token. The most basic approach loops the entire model (or specific blocks) on the same input representation multiple times.

This traces back to Universal Transformers , which dynamically loops a transformer block until a halting mechanism triggered. Bai et al. takes this to the extreme with Deep Equilibrium Models (DEQs), defining the output of a network not via a finite sequence of layers, but as the fixed point z^* of an infinitely applied function, found via root-finding algorithms. These earlier renditions of looped models repeated a single transformer layer, which is an attention layer followed by a two layer MLP. More recent methods often loop larger blocks of layers .

Generally, most methods either loop immediately from initialization or only loop after the full pretraining as a method of inference-time scaling. Next, we will explore related methods that yield similar effects to looping but utilize dynamic architectures across their training lifecycle.

Implicit Looping

If depth is the goal, how do we reach it without the training instability and cost of massively deep networks?

At initialization, signal propagation in a very deep network is hindered compared to a shallow one. While residual connections solve the gradient vanishing problem to an extent , they do not handle the "training efficiency" problem.

Equating training steps or FLOPs, smaller models tend to train much more efficiently at the beginning but plateau early. Model growth tries to follow the rapid training curve of a small model, then expands the loss landscape dimensionality to surpass the plateau towards the training curve of the larger architecture as if it were trained from scratch.

For the context of depth recursion, we focus the following subsection on depth growth (and particularly on one established strategy for it), although models may also be grown in many other dimensions .

Depth Growth via Stacking

Depth growth can be accomplished via effectively "progressive initialization via looping". MIDAS (Middle Gradual Stacking) proposed by Saunshi et al. takes a trained block of N layers from the middle of the model depth, copies it, and stacks it on top of itself. This process repeats multiple times periodically throughout training, growing the model from a single block of layers to full standard model depths. The MIDAS strategy, depicted below, of initializing the new block using pretrained parameters from a neighboring block outperforms other depth-growth strategies including interwoven layer insertions and random initializations .

The Middle Gradual Stacking (MIDAS) Strategy . Left: This visualization shows how model depth grows during training, where each blue block is a block of layers. Right: Weight similarity, measured by cosine similarity, across the six blocks each consisting of four layers of a MIDAS-grown model. Figures from Saunshi et al. .

Because parameters are deep-copied instead of merely shared, they may diverge upon further training. Each block thus shares at least part of its training history with other blocks throughout the depth of the network. The weight similarity between blocks of a MIDAS-grown model are depicted above: blocks near the middle of the network were forked more recently than at the beginning and end, thus yielding more similar weights.

Such initialization of each newly inserted block provides an inductive bias as the model continues training at the increased depth. The final model is a standard single-pass network (albeit trained more efficiently), but the initialization scheme simulates soft looping with refinement due to the shared training histories. Thus, this style of depth stacking yields a model that is akin to a looped model, where the loop depth is grown over the course of training, but each loop has a parametrization that is most similar between middle blocks and most customized for the initial and final blocks. This patterning is also seen in looping methods that utilize a prelude and/or coda .

Other depth-growth methods stack the entire sequence of layers, multiplying the depth at each growth . The resulting weights are thus more aligned across the depth, more akin to looping methods that loop the entire depth of layers.

Induced Looping via Surgery

Depth growth via stacking is not the only strategy to achieve this effect. Bae et al. and McLeish et al. perform parameter surgery on pretrained unlooped models to fit a soft looping architecture before continuing training. The useful inductive bias for mid-training initialization is thus also present, but the efficiency gains come from reusing parameters of previously trained models instead of within the training process itself.

These surgery techniques induce (soft) looping across the depth axis, where intermediate layers now share weights. During subsequent finetuning, the model trains within the constrained parameter space, enforcing this recurrence in the final architecture and at inference time. Other surgery techniques perform surgery to initialize deeper models from trained shallow ones, but without weight tying constraints during further training , thus closer to the paradigm of stackign depth growth.

Both progressive stacking and parameter surgery are effective strategies for imposing a recursive prior onto models: stacking achieves this by gradual growth during pre-training, while surgery does so by structurally modifying a pre-trained vanilla model.

Depth Recursion as Latent Chain of Thought

Beyond efficiency, a rather exciting implication of looping and parameter reuse is reasoning.

In standard Chain-of-Thought (CoT) , the model explicitly materializes its reasoning into discrete tokens (e.g., “Let’s think step by step…”), which are recycled as context for subsequent steps. This mechanism effectively allows the model to trade time for compute at inference-time, expanding its effective depth by utilizing the context window as a scratchpad. However, this introduces a bottleneck: to generate a token, the model must collapse the high-dimensional internal state into a single discrete symbol before reuse in future token generations. This discretization discards the nuanced probabilistic information held in the hidden states, forcing the model to commit to a specific textual path.

Looping enables the model to reason entirely within the continuous embedding space, avoiding what may be “lost in translation” via the discretization noise inherent in token generation. In this regime, the input to the repeated block comes directly from the previous iteration’s latent state. Through superposition and uncertainty, this may simultaneously represent multiple conflicting reasoning paths, refining the probability distribution continuously rather than committing to a single, potentially erroneous branch early in the derivation. The ability to hold multiple hypotheses simultaneously within the vector space is a key advantage, circumventing the information loss associated with discrete token sampling.

Saunshi et al. and Hao et al. further explore the similarities of looping as Latent Chain-of-Thought, or LCoT. However, as these works only consider strict looping for LCoT, the repeated function must be identical with each loop.

Depth growth via stacking and induced soft looping present an even more powerful evolution of LCoT: By initializing layers as copies (stacking) during depth growth or sharing parameters with learned offsets (soft-looping), we start with the useful inductive bias of a loop but allow for computational specialization throughout the depth of recurrence. This adaptive approach means the model can dynamically learn specialized transformations, such as early layers focusing on feature extraction and later layers focusing on logical inference, to different stages of the softly recurrent reasoning process, which is structurally impossible in a strictly weight-tied looped model. The weight similarity pattern of MIDAS portrays this depth-wise specialization.

Ultimately, this suggests that depth recurrence is not just a compression trick, but a functional equivalent to time recurrence operating in a richer, continuous, and potentially adaptive state space.

Sequential Refinement

To complement refinement in the depth axis, how can refinement be useful in the token axis? Most models are autoregressive: they process and generate one token at a time. This “hard commit” nature of autoregressive decoding means that earlier errors cannot be corrected based on later context, limiting the quality of sequential generation. To overcome this, sequential refinement strategies may be employed to allow the model to review and refine tokens.

Geiping et al. propose diffusion forcing sampling for depth recurrent models to enable recent token refinement at each step. This ameliorates the token discretization issue previously discussed for standard CoT.

Hierarchical Recursive Models and their successor Tiny Recursive Models feed the entire latent sequence back through its hierarchical modules, looping multiple times on the latent “scratchpad” for each output refinement.

In between sequential refinement and explicit CoT, a simple approach lies in the use of a “pause” token </d cite>. Although such a token still requires a hard commit, incorporating learnable tokens allows the model to delay the output generation both token by token during generation as well as after processing the input and beginning to generate the output, effectively utilizing the additional computation to refine its internal state before committing to a final answer nor using an explicit chain of thought.

Dynamic Routing

Another axis that breaks the hard-coded paradigm is dynamic routing. This family of techniques customizes the computational path for each input based on its content, seeking to enhance both efficiency and capacity via specialization. One of the most well-known dynamic routing techniques is portrayed by Mixture-of-Experts (MoEs) , which replaces the standard Feed-Forward Network (FFN) with several specialized subnetworks (experts) and a router that selects only a few experts per token. However, classic MoE architectures often still fix the total depth and FLOPs per token, merely selecting which parameter sets to use within these computations.

The concept of input-dependent layer-dropping pushes dynamic routing further by aiming to customize the computational depth per token. Early-exit strategies aim to accelerate inference in large autoregressive models by allowing tokens to skip later blocks. Depth-recursive models are even more naturally adept at modulating the effective depth per token.

This optimization introduces significant complications particularly related to maintaining the Key-Value (KV) cache, a mechanism utilized by autoregressive attention-based models that stores the computed keys and values from attention layers corresponding to previously generated tokens to accelerate sequence generation at inference time. When a token exits early, the KV caches for all subsequent (skipped) layers are missing, complicating the generation of future tokens that might require a deeper path. Solutions include duplicating the cached items from current token’s exit layer to subsequent layers or using batched forward passes upon cache miss . Depth recursion methods with partial or full weight sharing can counteract some of these inefficiencies, allowing for depth-wise batching such that tokens at different recursion depth may be re-batched together .

While dynamic routing optimizes where extra computation happens, and sequential refinement optimizes when it happens, a lucrative path forward lies in combining these mechanisms with the structural priors of recursion discussed earlier.

Towards Compositional Recursion

Many of the high-level methods presented are orthogonal and thus composable, perhaps even synergistically, offering a flexible toolkit for designing resource-efficient reasoning models. One such example is Zamba , which alternates blocks of time-recurrent SSM layers with a shared-weight attention module. As a further hypothetical example, an MoE with a looped block that shares attention layer weights but not experts could be grown in recursion depth during pre-training, using stacking to initialize the new experts. A fully trained hybrid model could undergo parameter surgery to induce soft looping, followed by inference-time utilization of sequential refinement and depth-wise batching. The convergence of these methods, where efficiency meets algorithmic capability, points toward a future of modular and resource-adaptable models rather than brute-force scale.

Conclusion

With the desire towards deeper reasoning, the rigid, single-pass transformer architecture appears increasingly insufficient and ineffective. By reusing parameters through looping and simulating recursion depth through growing, models can affect deeper computation without necessarily becoming larger or more costly to train. The synergies of depth growth, parameter sharing, and looping can be utilized across the training cycle of a model to achieve reasoning capabilities that ascend beyond scaling laws of standard models.

Enjoy Reading This Article?

Here are some more articles you might like to read next:

- Sample Blog Post