Life of Research · Published 10 November 2026 · DOI 10.5281/zenodo.24094666

A Loop Count Is Not a Compute Dial: What the Claim That Looped Models Trade Inference Iterations for Accuracy Measures, Why Truncation, Continuation and Easy-to-Hard Iteration Come Apart, and the Controls That Would Separate Them

Pranay Mahendrakar
SONYTECH · ORCID 0009-0003-7224-029X
pranaymahendrakar@sonytech.in · mahendrakarpranay@gmail.com · +91 6361723454

Abstract. Looped and recurrent-depth models apply one shared block many times, and a growing literature describes the number of applications as a test-time compute axis: run more loops, get a better answer, with no new parameters. A June 2026 speech recognition study is a clear recent case. It reports that naively looping a shared encoder block does not fully exploit extra recurrent compute, introduces a depth-conditioned design whose word error rate falls as more loops are executed, and concludes that test-time compute scaling extends beyond language-model reasoning to speech. In this analysis the author reviews and synthesises published results to show that "more loops at test time" names four different operations with different evidence behind them. Truncation, stopping before the depth the model was trained for, is an anytime property that early-exit networks already provide; the speech result is of this kind, because its depth conditioning is normalised to the trained budget and every reported exit lies inside it. Continuation, running past the training depth on inputs like the training inputs, has a ceiling at the training depth on general benchmarks in every model measured except those of one controlled study: gains flatten for models trained with sampled depth and reverse for models trained at fixed depth, by up to 26 points on a competition mathematics set, while in that study fixed-depth looped models gained up to about 3 points on a group of reasoning benchmarks at twice the training depth as their knowledge scores fell. Easy-to-hard iteration, running longer on inputs that need more sequential steps than training supplied, is where extra loops buy their largest gains past the training depth, and the published conditions for it are specific: a task whose required steps grow with the instance, an operator that settles, and a training contract that does not tie the step count to the training depth. Refinement, dividing a normalised trajectory into finer steps, is a fourth reading that one 2026 study supports. The paper consolidates 27 published results in Table 1 and Figure 1, states a reading procedure as Algorithm 1, and specifies the matched controls that would settle what remains open.

Keywords: looped transformers; recurrent depth; test-time compute; early exit; depth extrapolation; overthinking; speech recognition; adaptive computation

1. Introduction

The recent interest in test-time compute grew out of language models that spend more tokens before answering. Snell et al. (2024) showed that allocating inference compute adaptively per prompt could beat a best-of-N baseline by more than a factor of four in efficiency, and that a smaller model given extra inference compute could, in a FLOPs-matched comparison and on prompts where it already had some success, outperform a model 14 times its size. Muennighoff et al. (2025) fine-tuned a model on 1,000 reasoning traces and controlled how long it thought by cutting its reasoning off or, when it tried to stop, appending "Wait". In both cases the extra computation is serial and visible: it is written out as tokens.

A second line of work proposes to move that computation inside the network. A looped or recurrent-depth model applies one block of layers repeatedly to a hidden state, so its computational depth is decoupled from its parameter count (Dehghani et al., 2018; Giannou et al., 2023; Saunshi et al., 2025). Geiping et al. (2025) trained a 3.5-billion-parameter model of this kind on 800 billion tokens and reported that it can improve on reasoning benchmarks "up to a computation load equivalent to 50 billion parameters" by iterating its recurrent block more at test time. Ouro (Zhu et al., 2025a) trained looped models on 7.7 trillion tokens. Parcae (Prairie et al., 2026) derived scaling laws for looping. The phrase that recurs across these papers is that loop depth is a new axis of test-time scaling. A 2025 survey of latent reasoning (Zhu et al., 2025b) groups such models under activation-based recurrence and notes their extrapolation on algorithmic tasks, without separating the operations this paper separates.

The newest place the phrase has travelled is speech. Kaloga et al. (2026) start from the observation that an end-to-end speech recogniser's acoustic encoder has a fixed depth at inference, so there is no way to trade extra computation for better recognition without training a bigger model. They loop a four-block Transformer encoder, find in early experiments that plain reuse of the block is not competitive, and introduce LARM, which

© 2026 Pranay Mahendrakar. Licensed under CC BY 4.0.

adds sparse connectionist temporal classification (CTC) checkpoints, a clock embedding that marks each loop's position between checkpoints, feature-wise (FiLM) conditioning on the loop's depth, and feedback of the previous loop's soft token posteriors shifted by one frame. On LibriSpeech, LARM's greedy word error rate falls as more loops are executed, and it is competitive with deeper encoders that have unshared parameters. The authors conclude that test-time compute scaling can extend beyond autoregressive language-model reasoning to non-autoregressive speech recognition.

Each piece of that result is sound, and the paper is careful about what it measured. The question this analysis asks is what the phrase "test-time compute axis" licenses when it is attached to a looped model, because the literature attaches it to experiments that measure different things. In one kind of experiment, a model trained to run K loops is stopped earlier than K and gets worse; this shows that it can be run cheaply at some cost, not that it can be run expensively at some gain. In a second kind, a model trained at depth K is run past K on the same benchmark; here the evidence from several independent groups is that gains stop near K and frequently reverse. In a third, a model trained on easy instances is run past K on harder ones; here extra loops do convert into accuracy, in some architectures and on some tasks, by large margins. In a fourth, a model whose loops are indexed by a normalised time is run with a finer grid. These are four operations. A claim of test-time scaling is a claim about the second, third or fourth, and most of the reported evidence for looped language and speech models is evidence about the first.

In this analysis the author reviews and synthesises published results to show that the four operations separate cleanly in the published record, that the separation explains why studies of looping appear to disagree about whether extra loops help, and that it implies specific controls a test-time scaling claim should carry. The paper makes seven contributions. Section 2 defines the four operations and the comparators each needs (Table 2). Section 3 reads the speech anchor closely and shows, from its own design and tables, that its test-time results are truncation results, and that at matched executed compute its advantage over an unshared encoder reverses when training data grow from 100 to 960 hours. Section 4 places truncation in the early-exit literature, where it is an established capability with a known failure called overthinking. Section 5 reads the evidence on continuation and finds a ceiling at the training depth on general benchmarks in every model measured except those of one controlled study, in which fixed-depth looped models gained up to about 3 points on a reasoning benchmark group at twice the training depth while their knowledge scores fell (Figure 1). Section 6 reads the evidence on easy-to-hard iteration and states the published conditions under which extra loops convert. Section 7 identifies a design tension between conditioning a loop on its step index, which helps at a fixed budget, and keeping it step-agnostic, which is what extrapolation needs, and shows how normalised-time conditioning turns extra loops into refinement instead. Sections 8 and 9 examine adaptive halting and the comparison with token-level test-time compute. Section 10 consolidates 27 published results (Table 1), Section 11 states a reading procedure (Algorithm 1), and the remaining sections give five studies that would settle the open part, the strongest case against this paper, what is not known, limitations and a conclusion.

1.1 Relation to the author's earlier papers

Two earlier papers by the author touch looped models from other directions, and this one is scoped against both. "Tiny in Parameters, Not in Compute" asks what a recursive reasoning model's puzzle score credits to recursion (Jolicoeur-Martineau, 2025), on Sudoku and ARC-AGI, and argues that recursion there serves computation, regularisation and candidate generation for a learned selector. "A Steeper Slope Is Not Yet a New Law" asks whether looping changes the exponent of a training-compute scaling law. Neither asks the question here: whether, for a trained looped model in language, speech or control, executing more iterations at inference is a usable compute dial, and which of several things that phrase can mean the evidence supports. Where the earlier papers' sources reappear (Geiping et al., 2025; Bansal et al., 2022; Schwethelm et al., 2026), they are used here for their test-time results.

2. Four operations called "more loops at test time"

Write a looped model as $h(k) = F(h(k-1), x, c(k))$, where F is the shared block, x is the input (re-injected at every step in some designs and not in others), and c(k) is any conditioning on the step index. A readout maps h(k) to a prediction y(k). Training fixes a depth regime: a single depth K (Ouro, LARM), a depth sampled per step from a distribution with mean m (Huginn, the retrofitted models of McLeish et al. (2025), Parcae, RD-VLA), or a schedule over training. Call the training depth D: K for fixed-depth training, the mean m for sampled-depth training, with the caveat that a sampled distribution has a tail beyond its mean. A test-time run executes T iterations.

Truncation is $T < D$, with a readout at T. It answers the question: how much does the model lose when stopped early? A good truncation result is a curve that loses little. It is a property of an anytime system and does not by itself show that more computation helps, since the model was trained to use D iterations and is being given fewer.

Continuation is $T > D$ on inputs drawn from the training distribution. It answers: once the model has the depth it was trained for, does more depth help on the same kind of problem? A positive result would be the

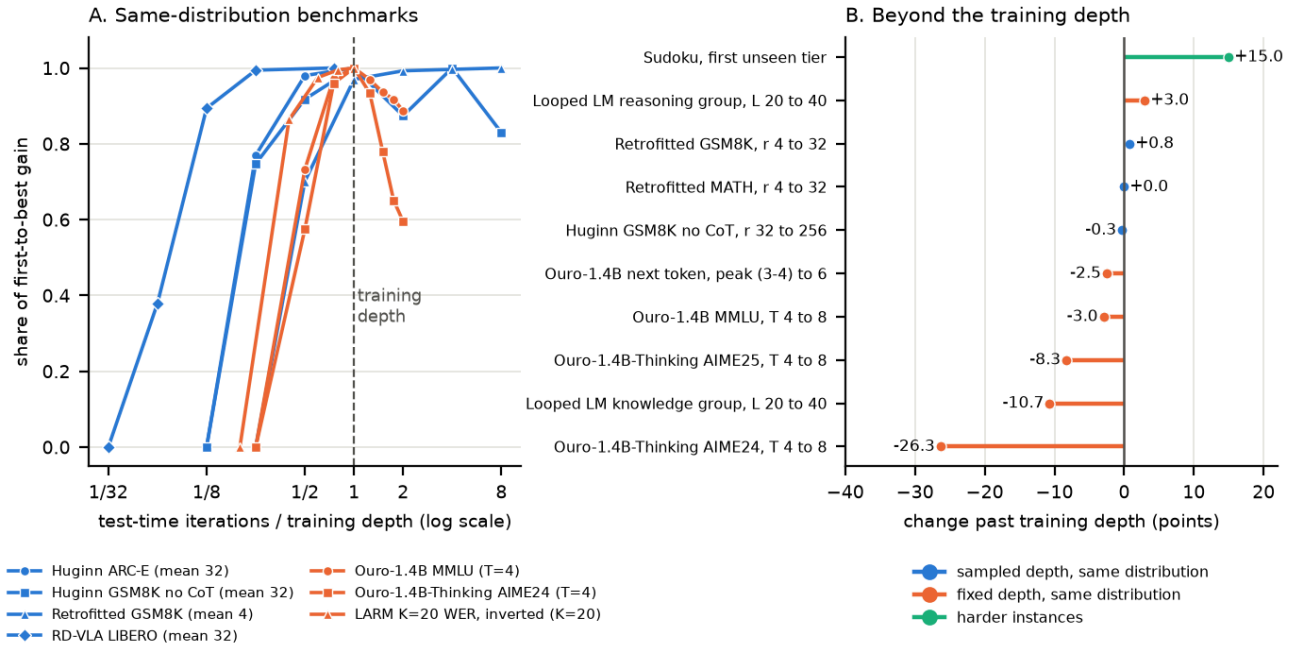


Figure 1. Panel A: share of each model's first-to-best gain reached at each test-time depth, plotted against test iterations divided by training depth (the normalisation is the author's; values from Geiping et al. 2025, Lu et al. 2025, McLeish et al. 2025, Tur et al. 2026, Zhu et al. 2025a and Kaloga et al. 2026, whose WER is inverted so that higher is better). Sampled-depth models (blue) flatten at or before their training mean; fixed-depth models (orange) peak at the training depth and fall after it. Panel B: change in points from the training depth (for Olszowiec et al. 2026, the best iteration) to the deepest reported test depth, from McLeish et al. 2025, Lu et al. 2025, Olszowiec et al. 2026, Zhu et al. 2025a, Zhuang et al. 2026 and, for the harder-instance row, Viakhirev et al. 2026.

clean form of the "new axis" claim.

Easy-to-hard iteration is T > D on inputs that require more sequential computation than any training input did: longer prefix sums, larger mazes, more hops of composition, longer state-tracking sequences, harder Sudoku tiers. It answers: can the model use extra iterations to execute more steps of a procedure it learned on short instances? This is the setting in which Schwarzschild et al. (2021) first described recurrent networks "thinking for longer".

Refinement is a change in how finely a normalised trajectory is sampled. If the step conditioning is c(k) = k/T rather than k, increasing T at test time divides the same interval into more, smaller steps, as an ODE solver does when its step size is reduced. Zhuang et al. (2026) evaluate timestep-conditioned looped language models in exactly this way, on rescaled time grids. Refinement can improve accuracy past the trained step count without the trajectory being any longer.

Adaptive halting is not a fifth operation. It is a policy that chooses T per input, and so chooses, input by input, among truncation, continuation and easy-to-hard iteration. Section 8 takes it up.

Each operation needs a different comparator. Truncation should be compared with a model of the same parameter count and training compute that has unshared layers and early exits, since that is the existing way to obtain an anytime encoder. Continuation needs only the model itself at T = D, because the claim is that T > D beats T = D. Easy-to-hard iteration needs a difficulty ladder whose top rungs were never seen in training, and a baseline that rules out coverage. Refinement needs a control that holds the number of distinct step indices fixed. Table 2 collects these requirements.

The separation matters because the four can come apart in one model. A model can truncate gracefully and continue badly; Ouro-1.4B-Thinking, as Section 5 shows, loses 2.7 points on AIME 2024 when stopped one step before its training depth and 26.3 points when run four steps past it. A model can continue safely and convert nothing: in the analysis of Olszowiec et al. (2026), looped models that re-read the prompt at every iteration never lose by looping longer, yet of the eight models examined only Huginn improves steadily with more iterations. And a model can convert on a hard tier of a task while failing to continue on the easy tier it was trained on, because the easy tier is already solved.

3. The speech anchor: what is measured and what the design permits

3.1 What was reported

LARM's reference configuration is a four-block shared Transformer encoder of width 384, run for K = 12 loops with CTC supervision at loops 4, 8 and 12 (Kaloga et al., 2026). Trained on the 100-hour LibriSpeech subset, it reaches 11.34 percent greedy word error rate on test-clean, against 26.78 for the same four blocks run once and 14.43 for a 16-block encoder with unshared

Table 2. Four operations named by "more loops at test time", what each tests, and what evidence would support a positive claim.

Operation	Test depth	Inputs	Question answered	Needed comparator	Example evidence
Truncation	T < D	as training	cost of stopping early	unshared early-exit model, matched compute	LARM exits; HuBERT-EE
Continuation	T > D	as training	gain from extra depth	same model at T = D	Huginn, Ouro, Parcae
Easy-to-hard	T > D	harder than training	executing more steps	difficulty ladder, coverage control	DT-Recall, PAR
Refinement	finer grid	as training	gain from smaller steps	fixed number of indices	timestep-conditioned LoopLMs

parameters. Trained on 960 hours, it reaches 4.59 against 14.07 and 4.79. With a 4-gram language model, the 960-hour LARM and 16-block encoder tie on test-clean at 3.51, and LARM is ahead on test-other, 9.38 against 9.87. The abstract's summary - performance competitive with deeper unshared-parameter baselines - is an accurate description of those rows.

The test-time scaling evidence is Figure 1 and Table 4 of the source. Within one trained model, greedy word error rate at successive supervised checkpoints falls: for the K = 12 model, 15.20, 11.57 and 11.34 at loops 4, 8 and 12; for a K = 20 model, 16.95, 11.83, 11.17, 11.05 and 11.02 at loops 4 through 20. Between checkpoints, the 100-hour model's error is not monotonic, while the 960-hour model's trajectory is smoother.

3.2 Every reported exit lies inside the trained budget

LARM's depth conditioning is a FiLM modulation driven by the normalised loop index d(k) = (k-1)/(K-1), where K is the loop budget fixed at training. Its clock embedding is indexed by k mod c. Training applies CTC at k in {c, 2c, ..., K}. Nothing in the reported evaluation runs a model for more than its training K, and the normalised index as defined is not given a meaning for k > K. The test-time curves are therefore truncation curves in the sense of Section 2: they show what a model trained for K loops delivers when read out at K/3 or 2K/3.

That is a useful property. The source's broader-impact section describes exactly this use: one stored model exposing several operating points by changing how many loops run. It is the anytime property of an early-exit encoder, obtained with shared weights. What it is not is evidence that the encoder can turn computation beyond its training budget into accuracy, and the source does not say that it is. The gap is in the phrase "test-time compute scaling", which in the language-model literature it cites refers to spending more than the default, not less.

3.3 Training budget is a training-time axis

The source's Figure 3(a) varies K across separately trained models: word error rate improves from K = 1 to K = 20 and degrades at K = 24, whose final greedy error is 13.34 against 11.02 for K = 20. Its extended results (Table 5) show further gains from K = 16 with 100 epochs. These are training-time results: each point is a different model. They resemble the finding of Saunshi et al. (2025) that a k-layer block looped L times nearly matches a kL-layer model, which is also a comparison among separately trained models. They show that loop budget is a useful architectural hyperparameter, with an optimum. Parcae reports a similar shape for training recurrence in language models, with a qualification: with truncated backpropagation fixed at eight steps, models trained with mean recurrence above eight performed worse at both low and high test recurrence despite more pretraining FLOPs, and the authors attribute this to the truncation, since raising the backward depth at a fixed training recurrence of 20 improved validation loss at test recurrences of 16 and 64 (Prairie et al., 2026).

There is one result in Table 4 that bears on the inference-time question directly, and it is mixed. For a fixed inference budget of eight loops, the K = 12 model's 8-loop exit (11.57) beats a model trained at K = 8 (12.07), so training for longer improved an early exit. At four loops, the ordering reverses: the K = 12 model's 4-loop exit (15.20) and the K = 20 model's (16.95) are worse than a model trained at K = 4 (14.07). For training budgets up to K = 20, a shared model run at a small budget costs up to 2.88 points of word error rate against a dedicated model trained for that budget, which is the standard price of an anytime model; for the over-long K = 24 model the cost at four loops is 5.98 points (20.05 against 14.07).

3.4 Most of the gain is looping, not the mechanisms

The source's ablation compares three 100-hour models at K = 12: the four-block encoder run once (26.78), a naive looped encoder without the loop-structuring mechanisms (12.70), and LARM (11.34). Computed from those three rows, looping alone removes 14.08 of the 15.44 points that separate one pass from LARM, or 91 percent; the checkpoints, conditioning and feedback remove the remaining 1.36. The abstract's statement that naive looping "does not fully exploit additional recurrent compute" is true, and the source's text notes that naive recurrence already improves substantially. A reader of the abstract alone would not learn that the mechanisms account for under a tenth of the improvement over a single pass. Among the components, removing feedback costs 0.97 points (12.31), removing depth conditioning costs 0.79 (12.13), and fixing the mixing scalars costs 0.57 (11.91).

3.5 At matched executed compute, the comparison depends on data

The source includes a 48-block unshared encoder as a reference, chosen to match the d = 1024 LARM both in parameters (85.7 million) and in Transformer-block applications (48). The parameter counts also fix this baseline's width. The 4-block and 16-block encoders have 7.6 and 28.9 million parameters, a difference of about 1.78 million per block, and 7.6 + 44 x 1.78 is about 85.7, so the 48-block encoder has the d = 384 width of the reference LARM. That makes the 48-block encoder the matched-compute comparator for the reference LARM: same width, same 48 block applications, about eleven times the parameters. The comparison is the author's reading of the source's Table 2, and the width is inferred from the parameter arithmetic rather than stated.

On 100 hours, the tied model wins: 11.34 against 12.24 greedy on test-clean, 31.84 against 34.06 on test-other. On 960 hours, the unshared model wins: 3.87 against 4.59 on test-clean, 10.58 against 11.75 on test-other, and 3.20 against 3.51 with the language model. The reversal with a ninefold increase in data is what one would expect if part of the tied model's advantage at 100 hours were regularisation, since a model with eleven times fewer free parameters overfits less on less data. The source does not draw this comparison, and the author's reading of it is an inference from two data points, not a measured effect of regularisation. It does show that "competitive with deeper unshared baselines" depends on which baseline: against the 16-block encoder, which executes a third of the compute, LARM is ahead; against the 48-block encoder at equal compute and width, it is ahead on less data and behind on more.

3.6 The deployed decoder does not see a monotone dial

The source reports language-model-decoded word error rate both at the final loop and at the best loop. In Table 4, the best decoded result occurs before the final loop for every K from 12 to 24: for K = 12, 8.38 at loop 7 against 8.66 at loop 12; for K = 20, 8.03 at loop 7 against 8.43. The authors treat greedy decoding as the cleaner diagnostic and say the interaction with the language model is complex. For a deployment, the decoder that matters is the one deployed, and under that decoder the 100-hour models' best operating point is not their deepest.

3.7 What the anchor establishes

Read against Section 2, LARM establishes four things: a looped encoder with shared weights is much better than its one-pass counterpart; structured supervision and conditioning add a further 1.4 points; one stored model can be read out at several budgets inside its training depth with graceful loss; and with less training data it beats an unshared encoder of equal compute. It does not establish continuation, easy-to-hard iteration or refinement, because none was tested, and its design, with a depth index normalised to the trained K, does not define them. The authors' own limitations section names the missing piece as an input-dependent halting rule. Sections 5 to 7 suggest that the larger missing piece is any evidence about T > K.

4. Truncation is an old capability under a new name

4.1 Early exit and its failure mode

Stopping a deep network before its last layer is a mature technique. Elbayad et al. (2019) trained Transformer decoders that can emit predictions at several depths and reported matching a well-tuned baseline on IWSLT German-English translation while using fewer than a quarter of the decoder layers. Schuster et al. (2022) built per-token early exit into language-model decoding with confidence measures and reported speedups of up to three times while keeping quality within stated bounds. Elhoushi et al. (2024) trained Llama models with layer dropout and a shared early-exit loss so that early layers become usable exits, and paired the exits with self-speculative decoding for speedups up to 2.16 times on summarisation.

The same literature found the failure that the looped-model literature later rediscovered. Kaya et al. (2019) named it overthinking: a network often reaches a correct prediction before its final layer, so the remaining layers are wasted, and sometimes the final layers change a correct prediction into a wrong one. On natural inputs, they found the destructive form behind half of misclassifications. Zhou et al. (2020) exited when intermediate classifiers stopped changing their prediction, and reported that this improved accuracy and robustness as well as speed, because it avoided overthinking. In both, the deepest computation is not the best computation for every input.

4.2 Early exit in speech recognition

Speech has its own version. Yoon et al. (2022) added exit branches to HuBERT's intermediate layers and stopped when a branch was confident, trading accuracy against latency on LibriSpeech. Berrebbi et al. (2022) measured overthinking in self-supervised speech models directly. For every evaluation set they examined, fewer than 10 percent of utterances needed the last layer to reach their best word error rate. On the in-distribution set an oracle could skip up to 20 percent of computation with no average loss, while on the out-of-distribution sets more than 70 percent of utterances needed the last four layers, and on Common Voice English more than 22 percent of predictions got worse at the final layer. Wright et al. (2023) trained exit-equipped speech encoders from scratch and found that they kept their performance when fewer encoder layers ran and reached

higher accuracy than single-exit models. Lasbordes et al. (2025) added parallel downsampled layers to make early exits stronger on edge devices.

Two of these findings bear on LARM. First, a speech encoder trained with intermediate exits can be read out early with graceful loss; that is what LARM's sparse checkpoints also deliver, with shared rather than unshared weights. Second, Wright et al. found that exit-trained encoders were more accurate than single-exit models, and LARM's Table 4 shows a related effect, with longer training budgets strengthening intermediate exits. Neither is evidence about running past the training depth. Berrebbi et al.'s result that out-of-distribution utterances need more depth is the opposite direction: it suggests that harder inputs would benefit from more layers than the model has, which is precisely what a fixed-depth encoder cannot give and what a looped encoder could, if it continued usefully. Section 12 proposes the test.

Block reuse in speech also predates LARM. Tang et al. (2023) reused Transformer blocks on AISHELL-1 and reported 9.3 percent character error rate without their small adapters (7.6 million parameters) and 6.63 percent with them (8.3 million). Nozaki and Komatsu (2021) fed intermediate CTC predictions back into later layers to relax CTC's conditional independence, reporting more than 20 percent relative word error reduction on WSJ, and Komatsu (2022) folded such a self-conditioned encoder so that blocks are reused. LARM's delayed soft-posterior feedback descends from this line. What LARM adds is the combination with sparse checkpoints and depth conditioning, and the framing as a compute axis.

4.3 What looping adds to truncation

If truncation is old, the contribution of a looped model to it is the parameter count: one block serves every exit. Bae et al. (2024) made this point for language models, converting pretrained Transformers into recursive ones with layer-wise LoRA relaxations and proposing continuous depth-wise batching, which they estimated could raise throughput two to three times when combined with early exit. Mixture-of-Recursions (Bae et al., 2025) assigns each token a recursion depth through a learned router, so that tokens leave the loop at different times. Both are explicit that the benefit is efficiency. A recursive model with exits is a smaller anytime model, which is valuable for devices whose compute budget varies, the use LARM's authors describe.

The comparator for a truncation claim follows. If the claim is that one shared model offers several operating points, the competing design is an unshared encoder with exits at the same block applications, trained with the same exit losses. LARM is compared with one-pass encoders of 4, 16 and 48 blocks without exits. The early-exit speech literature makes the stronger comparator available, and Section 12 specifies it.

4.4 When the answer is already fixed

A recent analysis of two large looped language models adds a precise reason why truncation often costs little. Luo and Yu (2026) study prediction dynamics in Huginn-3.5B and Ouro-1.4B and explain how intermediate answers can agree with the final one while their scores keep moving; in a retrospective analysis of completed trajectories, accounting for the direction of each update relative to the winning answer and for the pairing of each competitor's update with its score gap moved the earliest depth that their margin bound certifies earlier by a further 22.5 to 34.4 percent of total depth, beyond the gain from removing the common translation of scores, under full answer-text scoring. Olszowiec et al. (2026), examining eight looped language models from four families, report that in six of the eight, 76 to 89 percent of generated tokens already take their final top-1 token at the first iteration. Their separate measure of hidden-state settling reaches 98 to 100 percent of tokens in the three re-read loops, but they caution that "Settling, however, does not mean the prediction is fixed": Huginn's state settles while its top-1 token changes 1.4 to 1.7 times per token. A loop whose answer is already fixed offers nothing more to buy. In such a model a long trajectory is not evidence of useful computation; the useful computation may have ended much earlier.

5. Continuation: a ceiling at the training depth, with one exception

5.1 Models trained with sampled depth flatten

Huginn was trained with a recurrence count sampled each step from a log-normal Poisson distribution with mean 32 and a heavy tail, with gradients through only the last eight iterations (Geiping et al., 2025). Its Table 1 reports accuracy at r = 4, 8, 16 and 32. On ARC-Easy the values are 49.07, 65.11, 69.49 and 69.91; on HellaSwag, 43.46, 58.54, 64.67 and 65.21. Almost all of the gain is reached by r = 16, half the training mean. The paper's Figure 9 shows that on ARC-Challenge without few-shot examples the model saturates at around 8 to 12 iterations, and that with 25 to 50 examples in context it saturates at around 32. Its exponential-moving-average checkpoint reaches 47.23 percent flexible on GSM8K at r = 64, a value given for that checkpoint only, without a matched r = 32 number in the same place.

Lu et al. (2025) ran Huginn on GSM8K with chain-of-thought suppressed and varied the recurrence from 4 to 256. Accuracy rose from 3.11 at r = 4 to 4.93 at r = 32, and then stayed between 4.62 and 4.93 out to r = 256. With chain of thought, at r = 64, Huginn scored 24.87 strict and 38.13 flexible. The latent loop gave under two points; the written chain gave over twenty.

McLeish et al. (2025) retrofitted recurrence into pretrained models and trained them with a sampled

recurrence of mean 4. Their single-phase TinyLlama-based model's GSM8K accuracy at test recurrences 1, 2, 4, 8, 16 and 32 is 25.6, 44.1, 51.2, 51.8, 51.9 and 52.0; MATH is 8.8, 13.8, 14.5, 14.1, 14.4 and 14.5. The curve rises to the training mean and is flat for an eightfold increase beyond it.

Parcae states the pattern as a law. Prairie et al. (2026) fit test-time validation loss as a saturating exponential in the test recurrence, L(T) = L_inf + Z exp(-zT), and report that when their models are evaluated at twice the training recurrence, gains plateau near it, which they read as the training depth setting the ceiling on test-time scaling. RD-VLA, a recurrent action head for a vision-language-action model trained with sampled depth of mean 32, reports average LIBERO success of 8.4, 40.5, 84.1 and 92.6 percent at one, two, four and eight iterations, saturation between 8 and 12, and a peak of 93.1 at 24 (Tur et al., 2026). Its authors list depth generalisation as the open boundary: past an optimum, extra recurrence may saturate or degrade the state.

5.2 Models trained at fixed depth reverse

Ouro was trained with four recurrent steps after an initial stage with eight was abandoned for loss spikes (Zhu et al., 2025a). Its tables evaluate steps 1 to 8 and label 5 to 8 as extrapolation. For Ouro-1.4B base, MMLU is 41.21, 60.43, 66.71 and 67.45 at steps 1 to 4, then 66.64, 65.77, 65.28 and 64.49 at steps 5 to 8; ARC-Challenge falls from 60.92 at step 4 to 58.19 at step 8. For Ouro-1.4B-Thinking, AIME 2024 is 65.00 at step 4, 60.67 at step 5 and 38.67 at step 8; AIME 2025 is 46.30 at step 4, 47.00 at step 5 and 38.00 at step 8. The Ouro authors describe performance as peaking at the trained depth for the base model and around steps 4 or 5 for the thinking model. They also report that one property, safety on HEx-PHI, kept improving at extrapolated steps; accuracy did not.

Two later studies of Ouro confirm and extend this. Yang et al. (2026) report that on GSM8K Ouro-1.4B loses 20.47 percent from its peak after eight recurrent steps, and that their stability-driven training reduces the loss to 8.26 percent while raising the peak by 4.01 percent. Olszowiec et al. (2026) ran the Ouro models two steps past training and found the base models best at step 3 or 4: for Ouro-1.4B, next-token accuracy on prompt tokens falls from 0.605 to 0.580, and calibration error at the last step (0.087) is well above its minimum along the loop (0.057). They describe the loop as having learned a four-step pipeline rather than a process that keeps improving. LARM's K = 24 model (Section 3.3) is the training-time form of the same failure in speech.

5.3 Why the same-distribution ceiling is expected

The ceiling has a simple reading. On a benchmark drawn from the training distribution, the model was trained to produce its answer within the training depth, and nothing in the input demands more steps. Once the state has reached the region from which the readout gives the trained answer, further iterations either leave it there (if the operator settles) or move it out (if it drifts). Neither improves on what training optimised. The phrase "test-time scaling" suggests a curve with no built-in ceiling, but the ceiling sits wherever the training procedure put it.

Two qualifications keep this from being a universal negative. First, the training distribution's tail matters for sampled-depth models: Huginn's distribution occasionally samples well beyond 32, so its r = 32 point is not strictly a ceiling, and McLeish et al.'s mean-4 models saw some depths above 4. That makes the flat region past the mean more striking, not less, since those depths were partly trained. Second, few-shot context shifts the saturation point: Huginn uses more iterations when more examples are in context (Geiping et al., 2025). This is an input that demands more processing, which is the easy-to-hard condition of Section 6 appearing inside a benchmark.

5.4 Safe is not the same as useful

The fixed-depth and sampled-depth cases differ in whether continuation harms, and that difference has been traced to design. Olszowiec et al. (2026) separate carry loops, in which the prompt enters only through the initial state (Ouro, LoopUS), from re-read loops, in which the prompt is injected at every iteration and the depth is randomised in training (Huginn, Rec-OLMo, Parcae). For re-read loops, best and last iterations agree within 0.01 on every set they tested, so looping longer never costs anything; for carry loops it often does. Yang et al. (2026) describe a related split as a trade-off: with normalisation inside the residual branch, information flows but the state drifts with extra iterations; with normalisation outside it, the state is bounded but converges to an attractor that limits what each step can add.

Viakhirev et al. (2026) give the cleanest statement of why safety and usefulness are different properties. They define depth-safety, that further iterations neither change nor degrade the decoded answer, and conversion, that extra iterations raise accuracy on instances harder than training. They give a sufficient condition for depth-safety, that the per-step displacement is small relative to the decoder's margin, and show that one terminal fixed-point objective moves their operators between settling and drifting. Applied to Huginn, their measurements place it in the non-settling family; a label-free objective repaired its depth-safety on digit copying, from 0.00 to 1.00, without making it convert. A settled operator is safe because it has stopped computing. Safety is necessary for a test-time dial and is not sufficient.

5.5 The exception: a reasoning benchmark group

One controlled study cuts against a uniform ceiling, and it should be stated at full strength. Zhuang et al. (2026) trained looped models following the Llama3.1-1B and

Qwen3-0.6B architectures, each configuration at a fixed effective training depth, and evaluated them below, at and beyond the training horizon on separate knowledge and reasoning groups. The reasoning group is not a niche subset: it is the mean of eight benchmarks, among them ARC-Challenge, WinoGrande, HellaSwag, CommonsenseQA and BBH, the same kind of general benchmark used for the Huginn and Ouro rows above. For the 4 x 5 BaseLoop configuration on the Llama3.1-1B architecture, the reasoning score rose from 28.52 at the training depth of 20 to 31.49 at 40, a gain of 2.97 points, while the knowledge score fell from 62.80 to 52.11 and the overall score from 37.87 to 37.11. With extra test-time depth, two configurations passed the reasoning score of the unshared 20-layer model (30.25): 5 x 4 reached 31.88 at 1.5 times the training depth and 4 x 5 reached 31.49 at twice it. The authors conclude that "the training horizon does not impose a strict ceiling on effective recurrence" (Zhuang et al., 2026). The gains depended on the configuration: allocating more non-recurrent layers before the loop sustained or further improved reasoning under extended unrolling in some configurations. They also observe that convergence alone does not explain the useful cases, since the update size between iterations kept shrinking past the horizon while the variance of the recurrent state kept growing and downstream performance could fall. This is a genuine instance of continuation helping on a natural benchmark group, in models trained at fixed depth, so it qualifies both the ceiling and the finding that fixed-depth models reverse. It sits close to the easy-to-hard condition of Section 6, because reasoning items are the ones whose difficulty is most plausibly a matter of steps, and it came with a loss on knowledge, so it is a property of particular configurations and task groups rather than of looping as such.

6. Easy-to-hard iteration: where extra loops convert

6.1 The original result and its repair

The setting in which extra iterations clearly help is older than looped language models. Schwarzschild et al. (2021) trained recurrent convolutional networks on small prefix-sum, maze and chess problems with few iterations and showed that running more iterations at inference let them solve larger instances. Bansal et al. (2022) found that this broke down on much harder problems, because the networks' behaviour degenerated after many iterations, and fixed it with two changes: recall, which feeds the input to every iteration, and a progressive loss, which trains from randomly chosen intermediate states so that the network cannot learn behaviour specific to an iteration number. Trained on 32-bit prefix sums with at most 30 iterations, the plain network solved none of the 512-bit test strings, while the recall network with progressive loss reached 97.12 percent at its peak, at iteration 237. Their maze models, trained on 9 x 9 mazes, solved 801 x 801 mazes with 20,000 iterations.

Anil et al. (2022) explained why. In equilibrium models, which iterate to a fixed point (Bai et al., 2019), performance on harder instances correlated strongly with path independence: whether the model converges to the same state from any initialisation. Path-independent models improved monotonically with test-time iterations; path-dependent ones degraded once the test iterations exceeded the training iterations. Both weight tying and input injection were necessary for path independence, and interventions that raised or lowered it moved accuracy in the same direction. Bear et al. (2024) constrained the Lipschitz constant of the recurrent block so that iteration provably converges to a unique solution, and reported more reliable extrapolation to harder problems with fewer parameters, including on the travelling salesperson problem where the unconstrained network failed to learn.

6.2 Transformers, state tracking and composition

The same structure appears in looped Transformers. Fan et al. (2024) trained looped Transformers with an adaptive number of steps on tasks with known iterative solutions and obtained strong length generalisation. Wang et al. (2026) trained a looped Qwen2.5-0.5B on permutation state tracking with at most 32 moves; supervising iteration k to match the state after k moves and rescaling the state to its first-iteration norm gave 95.5 percent full-state accuracy at 512 moves, against 55.7 percent reported for a prior recurrent method and 0.8 percent for explicit chain of thought on the same backbone. Removing the norm anchoring cut the extrapolation factor from 16 to below 2.2; replacing the progress-aligned targets with final-answer supervision cut it to at most 8.

Zhang et al. (2026) measured how training fixes the speed of the learned procedure. On group word problems, free training installs a frontier that advances a fixed number of positions per loop, with that speed scaling as the training length divided by the training loop count (exponent 0.98, R-squared 0.99). Giving more loops at test time than were ever trained then rescues late positions at a fixed input length, and the learned speed gives a halting rule. At matched depth and parameters, untied models extrapolated worst. Kohli et al. (2026) trained recurrent-depth Transformers on multi-hop composition of parametric facts, with a curriculum that reached 13 to 16 hops depending on the recurrence and, in a matched-data setting, up to 12 hops, and found that more inference iterations enabled deeper composition than training reached, with depth sampled during training working best.

6.3 The conditions, and the counter-evidence

Across these results, extra loops past the training depth convert when three things hold together. The task has a sequential procedure whose required number of steps grows with the instance: more bits, more moves, more hops, a larger maze. The operator settles, or is path independent, so that once a step has been executed it stays executed. And training does not tie the procedure's speed or its stopping point to the training depth: randomised depth, progressive losses and stochastic stopping are the published ways of preventing that (Bansal et al., 2022; Kuo et al., 2026; Kohli et al., 2026).

Each condition has counter-evidence, and it is not small. Kohli et al. (2026) report that beyond about 15 iterations their best model's out-of-distribution accuracy stopped improving, that the prediction margin rose and then fell for every model and task, and that input injection, the remedy of Bansal et al. and Geiping et al., did not resolve overthinking in implicit reasoning. Liang et al. (2026) found that two different looped mechanisms for length generalisation both become unreliable at greater depths, through decay of a readout state in one and of state propagation in the other, with local errors persisting and compounding. Kuo et al. (2026) traced the brittleness of looped length generalisation to a spurious correlation between sequence length and loop count in simple algorithmic tasks, which stochastic loop counts reduced. Zhuang et al. (2026), evaluating looped language models below, at and beyond their training horizon, found that recurrence can improve reasoning beyond the horizon while degrading knowledge, and that harder reasoning instances did not consistently benefit more. Viakhirev et al. (2026) found that which architecture settles is task-dependent, and that on Sudoku the settled operators raised first-unseen-tier exact match from 0.19 to 0.34 past the training horizon, a real gain but far from solving the tier.

The difficulty condition explains why continuation on language benchmarks flattens while easy-to-hard iteration on algorithmic tasks converts. A GSM8K problem or an ARC-Easy question does not come with a count of the steps it needs, and the training distribution already spans the benchmark's difficulty. A 512-bit prefix sum needs sixteen times the steps of a 32-bit one. Where an input really does need more serial work, as more in-context examples do in Huginn, the saturation point moves. Speech recognition sits closer to the language case: a LibriSpeech utterance does not need more sequential steps because it is longer, since attention processes the whole utterance at once. The kind of input that might need more depth in speech is the acoustically hard one - noisy, accented, far-field - and Berrebbi et al.'s out-of-distribution result points in that direction. Whether a looped acoustic encoder converts extra loops on such inputs has not been tested.

7. Specialisation against extrapolation: what step conditioning does

7.1 The tension

One of LARM's larger component ablations is depth conditioning: without it, word error rate rises from 11.34 to 12.13, and FiLM conditioning beats additive embeddings (11.98) and an MLP (11.77). The clock embedding similarly tells each loop where it sits between checkpoints. These mechanisms let a shared block act differently at different depths, which the source describes as letting shared weights specialise across recurrent steps.

The language-model and algorithmic literatures report that this kind of specialisation works against extrapolation. Geiping et al. (2025) tried a core block that takes the current step as input and found it interacted badly with path independence, producing models unable to extrapolate. Bansal et al. (2022) designed their progressive loss specifically to stop the network learning behaviour tied to the iteration number, so that it learns behaviour that can be repeated indefinitely. Anil et al. (2022) found path independence necessary for upward generalisation, and, in the author's reading, an operator whose update depends on an absolute step index is not in general path independent in their sense, because the same state maps to different successors at different steps. Kuo et al. (2026) frame the choice of when to stop as a training-time design decision for the same reason.

The two findings are consistent. Step conditioning helps when the budget is fixed, because a budgeted computation benefits from stages with different roles: early loops can gather context, late loops can commit. It hurts when the budget is extended, because there are no trained roles for steps past the end. LARM optimises for the first case and its results show the benefit. A test-time scaling claim is about the second case.

7.2 Normalised time turns more loops into refinement

There is a third option, and it is closer to LARM's design than to either extreme. Zhuang et al. (2026) condition looped language models on normalised time t = l / L_infer with step size 1 / L_infer, so that increasing the inference budget divides the same interval into more, smaller steps. Under this scheme, on a Qwen3-0.6B backbone evaluated at twice its training depth, branch gating raised the overall score of one configuration (the 7 x 4 loop arrangement) to 38.46 against 36.59 for the unconditioned loop, while in other configurations the same schemes gave much smaller or negative gains, and training-depth gains in one of them disappeared at the doubled depth. Their proposed combination of timestep conditioning with injection of the history of states outperformed both components and the unconditioned loop under extended budgets in their main comparison. They also report that conventional initial-state injection

gave limited robustness to varying depth.

LARM's FiLM index is normalised too, but by the training K and only on the training grid. Read under Zhuang et al.'s scheme, a LARM model given K' > K loops would be evaluated with d(k) = (k-1)/(K'-1), a finer grid over the same trained interval. Whether this would help is unknown. It would not be continuation, since the trajectory would not be longer; it would be refinement, and it would need the control in Table 2 that separates "more, smaller steps" from "more steps". The supervision clock, indexed by k mod c with c fixed, has no such reinterpretation, and the sparse CTC checkpoints would have to be placed somewhere on the new grid. A test-time scaling claim for LARM would have to choose among these and report the choice.

7.3 Stability is a precondition with a cost

Several 2026 designs make looping stable by bounding the operator's spectrum: Parcae constrains the spectral norm of its injection parameters (Prairie et al., 2026), and STARS regularises the spectral radius of the Jacobian so that states approach stable fixed points (Yang et al., 2026). Both report that stability makes behaviour across test-time recurrence predictable. Parcae's own test-time result is that the predictable behaviour is saturation. STARS reports that its regulariser reduces Ouro-style collapse on mathematics and yields reliable scaling on arithmetic. Viakhirev et al. (2026) say why stability alone cannot deliver conversion: a settled operator's answer stops changing, so it is depth-safe whether the answer is right or wrong. What converts extra loops into accuracy is the combination of stability with an input that still needs work. Stability guarantees that work, once done, is not undone.

8. Adaptive halting chooses among the operations; it does not create one

8.1 The lineage

Letting a network decide how many steps to take is older than the looped Transformer. Graves (2016) introduced adaptive computation time for recurrent networks, with large gains on synthetic parity, logic, addition and sorting tasks and, by the paper's own account, no large gains on character-level language modelling. The Universal Transformer added a per-position halting mechanism to a shared-weight Transformer and reported that it improved accuracy on several tasks (Dehghani et al., 2018). PonderNet learned the number of steps end-to-end and reported success on extrapolation tests where standard networks failed (Banino et al., 2021). In the current generation, Huginn exits per token when the KL divergence between successive steps falls below a threshold, and the number of steps differs across MMLU categories, with high-school mathematics exiting earlier than logical fallacies or moral scenarios (Geiping et al., 2025). Pappone et al. (2025) found that iterates within a looped block become smaller and more orthogonal over training, and that an exit based on the second-order difference of step size beat the KL exit. Ouro trains an exit distribution with an entropy-regularised objective (Zhu et al., 2025a), and Mixture-of-Recursions trains a router that assigns each token a depth (Bae et al., 2025).

8.2 What halting can and cannot add

A halting rule picks T for each input. If T never exceeds the training depth, the policy selects among truncation points, and its value is efficiency: the average computation falls while accuracy is held. That is what Huginn's zero-shot exit is built to deliver, and what the early-exit literature has reported for years. If T can exceed the training depth, the policy's value depends on whether continuation or easy-to-hard iteration works for the inputs it sends deeper, which is the question of Sections 5 and 6. A halting rule cannot make continuation help on inputs where it does not.

The published evidence on halting adds two cautions. Popescu et al. (2026) studied learned halting in looped Transformers, on synthetic tasks and on Ouro checkpoints, and found that simple post-hoc confidence readouts often matched or outperformed learned gates; fitting gates on frozen trajectories located the failure mainly in the trajectory that joint gate training induces, not in the gate. Ouro's pretrained ponder gates were competitive but not uniformly Pareto-optimal. In other words, training a halting gate changes what the loop computes. Bansal et al. (2022) reported, in preliminary appendix experiments, that a loss penalising run time by weighting later iterations more heavily hurt extrapolation: such models solved 72.45 percent of 512-bit prefix sums against 97.12 for their best method, and no 59 x 59 mazes. Kuo et al. (2026) draw the general lesson that when to stop is a training-time design choice. In speech, LARM's authors list the absence of an input-dependent halting rule as a limitation; the evidence suggests that adding one would make LARM cheaper on average without, on its own, adding a scaling axis.

The positive case for adaptivity is that some inputs genuinely need more depth. Berrebbi et al. (2022) found this for out-of-distribution speech, Huginn for few-shot prompts and some MMLU categories, and RD-VLA for its harder manipulation tasks, where single-iteration success of zero rose above 90 percent at four iterations while simpler tasks saturated quickly (Tur et al., 2026). Viakhirev et al. (2026) warn that this signal is easy to counterfeit: the latent state of any non-degenerate map moves for longer on harder inputs, so adaptivity alone shows nothing unless the extra steps convert into correct answers. Frey et al. (2026) found that learned per-layer looping mainly benefited mathematical reasoning while added memory banks recovered commonsense performance, which suggests that what the loop adds is computation and not storage.

9. Against token-level test-time compute, and at what price

9.1 Latent loops versus written chains

The claim that looping is a new test-time axis invites a comparison with the axis it is meant to rival. The direct comparisons favour the written chain on natural tasks. Lu et al. (2025) found Huginn's latent loop gave under two points on GSM8K out to 256 iterations, against 38.13 flexible accuracy with chain of thought at 64. Their probing found little evidence of interpretable latent chain-of-thought across recurrence steps. On structured state tracking the comparison reverses: Wang et al. (2026) report 0.8 percent for explicit chain of thought on the same backbone that their progress-aligned loop takes to 95.5 percent at 16 times the training length. Saunshi et al. (2025) prove that a looped model can simulate T steps of chain of thought with T loops, and observe downstream accuracy growing with the logarithm of effective depth across separately trained models. Merrill and Sabharwal (2025) show that Transformers whose depth grows logarithmically with input length can recognise regular languages and decide graph connectivity, which fixed-depth Transformers cannot under standard conjectures, and that growing depth is a more efficient route to these problems than growing width or the number of chain-of-thought steps.

Between the two extremes lie methods that add computation without writing words. Goyal et al. (2023) trained with pause tokens and found gains when the model was both pretrained and fine-tuned with them, for a 1B model gains of 18 percent exact match on SQuAD and 1 percent on GSM8K. Pfau et al. (2024) showed that filler tokens can replace a chain of thought on two hard algorithmic tasks but that learning to use them required specific, dense supervision. Coconut feeds the last hidden state back as the next input and reports advantages over chain of thought on logical tasks that need search (Hao et al., 2024). Encode-Think-Decode trains a base model during mid-training to iterate over a small set of layers and reports a 28.4 percent relative gain on GSM8K for a 1B OLMo-2 model when those layers are iterated at inference (Koishekenov et al., 2025). Across these, extra latent computation helps when the model was trained to use it, and the size of the help depends on whether the task's difficulty is the kind more serial computation addresses.

9.2 Counting the compute

A test-time scaling claim is a claim about accuracy per unit of compute, and loop counts are not compute units. Geiping et al.'s 50-billion-parameter figure describes computation load at high recurrence, not quality: the same paper describes its $r = 32$ model as roughly comparable to the first OLMo-7B on most metrics. Schwethelm et al. (2026) measured how much one recurrence is worth in equivalent unique parameters and found a recurrence-equivalence exponent of 0.46: at $r = 4$, a 410M looped model matched a 580M non-looped model while costing as much to train as a 1B one. In their sweep, truncated backpropagation, which Huginn and RD-VLA both use, lowered the exponent to 0.38. Csordas et al. (2025) add that even unshared deep language models use their later layers for fine adjustments rather than new computation, so the unshared baseline is not an efficient use of depth either. This is the background against which LARM's matched-compute result in Section 3.5 should be read: a loop that is worth less than a fresh block per application can plausibly still win when data are scarce enough that fresh blocks overfit.

10. Consolidated evidence

Table 1 brings together the published values the argument rests on. Each row names its source and, in the last column, the operation of Section 2 that the measurement tests. Where a value is computed or inferred by the author rather than printed by the source, the row says so.

Three patterns stand out. Every continuation row on a same-distribution general benchmark is flat or negative except the Zhuang et al. (2026) continuation row, where the reasoning group gained about 3 points at twice the training depth while the knowledge group lost about 11 (Section 5.5). Every row that shows a large gain from iterating past the training depth is an easy-to-hard row on a task with an explicit step count. And the speech rows are all truncation, training-budget or matched-compute rows. Figure 1 draws the first two patterns.

11. A procedure for reading a test-time depth claim

Algorithm 1 states the reading the paper applies. Its steps are ordered so that the cheapest disqualifying check comes first.

Applied to the speech anchor, Step 1 ends the procedure: all reported exits satisfy $T \leq K$, so the result is a truncation result, and the missing control is the matched-compute early-exit encoder. Applied to Huginn, Steps 3 and 7 apply: the curve is already flat from about half the training mean, and continuation past the mean is safe and flat on the benchmarks reported, and compute is best counted against unshared models and written chains, where Lu et al.'s comparison favours the chain. Applied to Ouro, Step 3 classifies continuation as depth-unsafe. Applied to PAR or DT-Recall, Step 4 is satisfied and the claim of conversion past the training depth stands for those tasks.

12. Studies that would settle the open part

Study 1: continuation and refinement for the speech anchor. Take trained LARM models at $K = 12$ and $K = 20$ and run them to $K' = 1.5K$ and $2K$ under three

Table 1. Published measurements of looped models at different test-time depths, by operation. D is the training depth (fixed K, or the mean of a sampled distribution). Values are as printed by each source unless marked.

Source	System and task	Train depth	Test depths	Reported values	Operation
Kaloga et al. 2026	4-block encoder, LS-100h, WER	K = 12	1 vs 12	one pass 26.78; naive loop 12.70; LARM 11.34	loop vs pass
Kaloga et al. 2026	LARM exits, LS-100h, WER	K = 12	4 / 8 / 12	15.20 / 11.57 / 11.34	truncation
Kaloga et al. 2026	dedicated LARM, LS-100h, WER	K = 4; K = 8	4; 8	14.07; 12.07	comparator
Kaloga et al. 2026	LARM K = 24 vs K = 20, WER	24; 20	24; 20	13.34 vs 11.02	train budget
Kaloga et al. 2026	LARM vs 48 unshared blocks, WER	K = 12	12	100h 11.34 vs 12.24; 960h 4.59 vs 3.87 (width inferred)	matched compute
Kaloga et al. 2026	LARM with 4-gram LM, WER	K = 12	best vs final	8.38 (loop 7) vs 8.66	truncation
Geiping et al. 2025	Huginn-3.5B, ARC-E	mean 32	4 / 8 / 16 / 32	49.07 / 65.11 / 69.49 / 69.91	truncation
Geiping et al. 2025	Huginn, ARC-C saturation	mean 32	sweep	8-12 iters at 0-shot; about 32 at 25-50 shot	continuation
Lu et al. 2025	Huginn, GSM8K, no CoT	mean 32	4 / 32 / 256	3.11 / 4.93 / 4.62; with CoT 38.13	continuation
McLeish et al. 2025	retrofitted 4,8,4, GSM8K	mean 4	1 / 4 / 32	25.6 / 51.2 / 52.0	continuation
Prairie et al. 2026	Parcae, validation loss	mean mu	to 2 mu	gains plateau near mu	continuation
Tur et al. 2026	RD-VLA, LIBERO success	mean 32	1 / 4 / 8 / 24	8.4 / 84.1 / 92.6 / 93.1	truncation
Zhu et al. 2025a	Ouro-1.4B, MMLU	4	4 / 8	67.45 / 64.49	continuation
Zhu et al. 2025a	Ouro-1.4B-Thinking, AIME24	4	4 / 5 / 8	65.00 / 60.67 / 38.67	continuation
Yang et al. 2026	Ouro-1.4B, GSM8K	4	peak vs 8	20.47% drop; STARS 8.26%	continuation
Olszowiec et al. 2026	Ouro-1.4B, next token	4	peak (3-4) vs 6	0.605 vs 0.580	continuation
Olszowiec et al. 2026	re-read loops (3 models, 2 families)	sampled	best vs last	within 0.01 on every set	continuation
Zhuang et al. 2026	Llama3.1-1B arch., 4 x 5 BaseLoop	20 layers	20 / 40	reasoning 28.52 / 31.49; knowledge 62.80 / 52.11	continuation
Viakhirev et al. 2026	settling operators, Sudoku	horizon	to 128	first unseen tier 0.19 to 0.34	easy-to-hard
Bansal et al. 2022	DT vs DT-Recall, prefix sums	at most 30	peak 237	512-bit: 0.00 vs 97.12	easy-to-hard
Bansal et al. 2022	run-time penalty (prelim.)	at most 30	-	512-bit 72.45; 59 x 59 mazes 0	halting
Wang et al. 2026	PAR, S5 state tracking	32 moves	512 moves	95.5 vs 55.7 (RLT) vs 0.8 (CoT)	easy-to-hard
Zhang et al. 2026	group word problems	T_train	above T_train	speed ∼ n_train / T_train, exp. 0.98	easy-to-hard
Kohli et al. 2026	multi-hop composition	dynamic	beyond 15	no further OOD gain	easy-to-hard
Berrebbi et al. 2022	SSL speech, per layer	fixed	each layer	under 10% need last; over 22% degraded (CV)	truncation
Schwethelm et al. 2026	iso-depth looped LMs	r = 1-8	= train	phi 0.46; truncated BPTT 0.38	accounting
Zhuang et al. 2026	Qwen3-0.6B, 7 x 4, branch gating	28 layers	28 / 56	37.27 / 38.46 overall; 4 x 7 and 14 x 2 fall	refinement

conditionings: the FiLM index extrapolated linearly past 1, the index rescaled to (k-1)/(K'-1), and the index clamped at 1. Keep the clock embedding cycling and read CTC at every fourth loop. Report greedy and language-model word error rate on test-clean and test-other. The rescaled run is refinement; the other two are continuation. A result in which any of them improves on the K-loop exit would be the first evidence of a test-time scaling axis in a speech encoder in the sense the language-model literature uses.

Study 2: the anytime comparator. Train an unshared 48-block encoder at width 384 with CTC exits at blocks 16, 32 and 48, using the same sparse supervision schedule, on 100 and 960 hours. Compare its word error rate at each exit with LARM's at loops 4, 8 and 12, which execute the same number of block applications. If the unshared encoder's exit curve dominates at 960 hours, LARM's contribution is parameter efficiency under data scarcity; if LARM's dominates, it is something more.

Study 3: a difficulty ladder for speech. Evaluate the K-loop and K'-loop models from Study 1 on graded acoustic difficulty: additive noise at decreasing signal-to-noise ratio, reverberation, accented and conversational speech. Record, per utterance, the loop at which word error rate is lowest, following the oracle analysis of Berrebbi et al. (2022). If the best loop moves deeper as difficulty rises, and past K on the hardest rungs, the easy-to-hard condition holds for acoustics. If not, the speech loop behaves like the language-model loops of Section 5.

Study 4: convergence time against intrinsic step count in a language model. Take a re-read looped language model (Huginn or Parcae) and a carry model

Algorithm 1. Reading a claim that a looped model scales test-time compute

```
Input: a model with training depth D (fixed K or sampled mean m), a set of
       reported test depths T, the benchmark, and the comparators used.
Step 1: If every reported T <= D, classify the result as TRUNCATION.
        Ask for an unshared early-exit comparator at matched compute and
        report the claim as an anytime property, not a scaling axis.
Step 2: If some T > D, split the inputs: drawn from the training
        distribution, or harder than any training input.
Step 3: For same-distribution inputs (CONTINUATION), compare accuracy at
        T > D with accuracy at T = D on the same model and decoder.
        If flat: depth-safe, no gain. If lower: depth-unsafe.
Step 4: For harder inputs (EASY-TO-HARD), require a difficulty ladder with
        unseen top rungs and an explicit step count, and check that the best
        T rises with difficulty. Rule out coverage of the hard rungs in training.
Step 5: If the step conditioning is normalised by the test budget
        (REFINEMENT), hold the number of distinct step indices fixed in a
        control run before crediting extra loops.
Step 6: If a halting rule chooses T, report its accuracy against the best fixed
        T, and state whether any input is sent past D.
Step 7: State compute in FLOPs or block applications, not loops, and compare
        with an unshared model of equal executed compute and with token-level
        computation of equal cost.
Output: the operation the evidence supports, and the controls still missing.
```

(Ouro), and evaluate them on arithmetic or state-tracking items whose number of required operations is known and extends beyond anything in training. Measure, per item, the iteration after which the decoded answer stops changing, which Viakhirev et al. (2026) call suffix stabilisation, and whether it is correct. A positive slope between required operations and stabilisation step, together with correct answers past the training depth, is the signature of a usable dial on that task family.

Study 5: equal-cost comparison with written computation. For the same looped model, compare accuracy at r iterations with accuracy when the model, or a matched unshared model, is allowed chain-of-thought tokens of equal FLOP cost. Lu et al. (2025) made this comparison informally at one point. A curve would show where, if anywhere, latent depth is the cheaper way to buy accuracy.

13. The strongest case against this paper

The first objection is that the speech anchor never claims extrapolation, so classifying its results as truncation corrects a claim nobody made. The author agrees about the body of the source, which is precise about what was run and lists fixed loop budgets as a limitation. The objection does not reach the source's title and abstract, which present the result as test-time compute scaling that "can extend beyond autoregressive language-model reasoning", or its related-work framing, which places it beside language-model methods - "search, verification, self-correction, and iterative refinement" - whose defining property is spending more than the default. A reader of the title would expect evidence about $T > K$. The paper's argument is that this expectation is natural and unmet, and that the same gap appears in the language-model literature where the phrase originated.

The second objection is that a depth-safe model that flattens is exactly what a compute dial should be: one turns it up until returns stop, and the stopping point is the optimum. The author agrees that this is a useful dial, and for deployment it may be the most useful kind. It is a dial whose top position is set at training time. Calling it a scaling axis implies that turning it further buys more, and on same-distribution benchmarks the published curves say it does not. The right description is an anytime model with a training-time ceiling.

The third objection is that the easy-to-hard results come from toy tasks - prefix sums, mazes, permutation groups - and say little about natural problems, so treating them as the one case where loops convert overweights them. The author agrees about their scope. The point of Section 6 is not that natural tasks will behave like prefix sums but that the published conversions all occur where the step count is explicit, and that natural benchmarks do not supply one. Kohli et al. (2026) and Zhuang et al. (2026) show partial conversion on composition and reasoning in language models, with limits. That is the frontier, and it is where the claim of a new axis has to be tested.

The fourth objection is that the training-depth ceiling is itself a moving target: models trained for longer have higher ceilings, as LARM's K sweep and Parcae's training recurrence results show, so the axis exists after all. It does, at training time. Each point on that curve is a different trained model, which is what Saunshi et al. (2025) also compare. That is a claim about architecture and training compute, and it is the subject of the author's earlier paper on scaling exponents, not of a test-time claim.

14. What is not known

One published study shows looped language models improving past their training depth on general benchmarks drawn from their training distribution: in Zhuang et al. (2026), fixed-depth models in some configurations gained up to about 3 points on a

reasoning benchmark group at twice the training depth, while their knowledge scores fell. Elsewhere the nearest case is under one point on GSM8K for a retrofitted model, and no such result has been reported for speech. Whether a model can be built whose overall accuracy keeps rising past its training depth is open.

It is not known whether LARM, or any looped acoustic encoder, improves when run past its training budget under any conditioning scheme. The experiment has not been reported.

It is not known whether looped encoders convert extra loops on acoustically harder speech. The early-exit evidence that harder speech uses more depth comes from unshared self-supervised models.

It is not known how much of LARM's 100-hour advantage over an unshared encoder of equal compute is regularisation. The reversal at 960 hours is consistent with that reading and does not establish it.

It is not known whether refinement, the finer-grid reading of normalised-time conditioning, helps in general. One study reports gains that depend on the configuration, and none tests it in speech.

It is not known whether a learned halting rule can send inputs past the training depth profitably. The existing rules mostly reduce average depth, and one study finds that training the gate alters the trajectory it gates.

15. Limitations

The analysis rests on values the author consolidated from published tables and figures. Several values in Table 1 are the author's readings of source tables, and two values in the paper are the author's computations: the 91 percent share of LARM's improvement attributed to looping, stated in Section 3.4, and the width of the 48-block baseline in Table 1, which is inferred from parameter counts rather than stated. Figure 1's panel A normalises each published curve to its own first and best values, which is the author's choice and removes absolute levels; panel B keeps them. Most of the 2026 sources are preprints and have not been refereed, including several of the studies on which the continuation and easy-to-hard sections rest. The selection of studies is the author's and favours those that report several test-time depths; studies reporting only one depth cannot speak to the question and were not sought out. The classification of rows into operations depends on what each source says about its training depth, which is precise for fixed-depth models and approximate for sampled-depth ones, whose tails extend past the mean.

16. Conclusion

"Run more loops, get a better answer" is four claims. Truncation, reading a model out before the depth it was trained for, is well supported and is what the speech anchor and most anytime looped models demonstrate; it is valuable, and early-exit networks have offered it for years. Continuation, running past the training depth on the same kind of input, has a ceiling at the training depth on general benchmarks in every model measured except those of one controlled study, flat for models trained with sampled depth and falling for models trained at fixed depth; in that study, fixed-depth looped models gained up to about 3 points on a reasoning benchmark group at twice the training depth while their knowledge scores fell. Easy-to-hard iteration, running longer on inputs that need more steps, converts extra loops into accuracy on tasks with an explicit step count, when the operator settles and training does not tie the procedure to a fixed depth. Refinement is a fourth possibility that one study supports and none in speech has tested. A claim of test-time compute scaling for a looped model should say which of these it is, report accuracy at depths beyond the training budget if it means more than truncation, and compare against an unshared model and a written chain of equal executed compute. The loop count is a dial. Where its top position lies is decided in training.

References

Anil, C., Pokle, A., Liang, K., Treutlein, J., Wu, Y., Bai, S., Kolter, Z., & Grosse, R. (2022). Path Independent Equilibrium Models Can Better Exploit Test-Time Computation. NeurIPS 2022. arXiv:2211.09961. doi:10.48550/arXiv.2211.09961.

Bae, S., Fisch, A., Harutyunyan, H., Ji, Z., Kim, S., & Schuster, T. (2024). Relaxed Recursive Transformers: Effective Parameter Sharing with Layer-wise LoRA. ICLR 2025. arXiv:2410.20672. doi:10.48550/arXiv.2410.20672.

Bae, S., Kim, Y., Bayat, R., Kim, S., Ha, J., Schuster, T., Fisch, A., Harutyunyan, H., Ji, Z., Courville, A., & Yun, S.-Y. (2025). Mixture-of-Recursions: Learning Dynamic Recursive Depths for Adaptive Token-Level Computation. arXiv:2507.10524. doi:10.48550/arXiv.2507.10524.

Bai, S., Kolter, J. Z., & Koltun, V. (2019). Deep Equilibrium Models. NeurIPS 2019. arXiv:1909.01377. doi:10.48550/arXiv.1909.01377.

Banino, A., Balaguer, J., & Blundell, C. (2021). PonderNet: Learning to Ponder. 8th ICML Workshop on Automated Machine Learning. arXiv:2107.05407. doi:10.48550/arXiv.2107.05407.

Bansal, A., Schwarzschild, A., Borgnia, E., Emam, Z., Huang, F., Goldblum, M., & Goldstein, T. (2022). End-to-end Algorithm Synthesis with Recurrent Networks: Logical Extrapolation Without Overthinking. NeurIPS 2022. arXiv:2202.05826. doi:10.48550/arXiv.2202.05826.

Bear, J., Prugel-Bennett, A., & Hare, J. (2024). Rethinking Deep Thinking: Stable Learning of Algorithms using Lipschitz Constraints. arXiv:2410.23451. doi:10.48550/arXiv.2410.23451.

Berrebbi, D., Yan, B., & Watanabe, S. (2022). Avoid Overthinking in Self-Supervised Models for Speech Recognition. arXiv:2211.08989. doi:10.48550/arXiv.2211.08989.

Csordas, R., Manning, C. D., & Potts, C. (2025). Do Language Models Use Their Depth Efficiently? NeurIPS 2025. arXiv:2505.13898. doi:10.48550/arXiv.2505.13898.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., & Kaiser, L. (2018). Universal Transformers. ICLR 2019. arXiv:1807.03819. doi:10.48550/arXiv.1807.03819.

Elbayad, M., Gu, J., Grave, E., & Auli, M. (2019). Depth-Adaptive Transformer. ICLR 2020. arXiv:1910.10073. doi:10.48550/arXiv.1910.10073.

Elhoushi, M., Shrivastava, A., Liskovich, D., Hosmer, B., Wasti, B., Lai, L., Mahmoud, A., Acun, B., Agarwal, S., Roman, A., Aly, A. A., Chen, B., & Wu, C.-J. (2024). LayerSkip: Enabling Early Exit Inference and Self-Speculative Decoding. ACL 2024. arXiv:2404.16710. doi:10.48550/arXiv.2404.16710.

Fan, Y., Du, Y., Ramchandran, K., & Lee, K. (2024). Looped Transformers for Length Generalization. ICLR 2025. arXiv:2409.15647. doi:10.48550/arXiv.2409.15647.

Frey, M., Shomali, B., Bashir, A. H., Berghaus, D., Koehler, J., & Ali, M. (2026). Adaptive Loops and Memory in Transformers: Think Harder or Know More? Latent and Implicit Thinking Workshop at ICLR 2026. arXiv:2603.08391. doi:10.48550/arXiv.2603.08391.

Geiping, J., McLeish, S., Jain, N., Kirchenbauer, J., Singh, S., Bartoldson, B. R., Kailkhura, B., Bhatele, A., & Goldstein, T. (2025). Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach. arXiv:2502.05171. doi:10.48550/arXiv.2502.05171.

Giannou, A., Rajput, S., Sohn, J., Lee, K., Lee, J. D., & Papailiopoulos, D. (2023). Looped Transformers as Programmable Computers. ICML 2023. arXiv:2301.13196. doi:10.48550/arXiv.2301.13196.

Goyal, S., Ji, Z., Rawat, A. S., Menon, A. K., Kumar, S., & Nagarajan, V. (2023). Think before you speak: Training Language Models With Pause Tokens. ICLR 2024. arXiv:2310.02226. doi:10.48550/arXiv.2310.02226.

Graves, A. (2016). Adaptive Computation Time for Recurrent Neural Networks. arXiv:1603.08983. doi:10.48550/arXiv.1603.08983.

Hao, S., Sukhbaatar, S., Su, D., Li, X., Hu, Z., Weston, J., & Tian, Y. (2024). Training Large Language Models to Reason in a Continuous Latent Space. COLM 2025. arXiv:2412.06769. doi:10.48550/arXiv.2412.06769.

Jolicoeur-Martineau, A. (2025). Less is More: Recursive Reasoning with Tiny Networks. arXiv:2510.04871. doi:10.48550/arXiv.2510.04871.

Kaloga, Y., Kumar, S., Sheikh, S. A., Khalil, D., Motlicek, P., & Kodrasi, I. (2026). Test-Time Compute Scaling for ASR with Depth-Conditioned Looped Transformers. arXiv:2606.04678. doi:10.48550/arXiv.2606.04678.

Kaya, Y., Hong, S., & Dumitras, T. (2019). Shallow-Deep Networks: Understanding and Mitigating Network Overthinking. ICML 2019. arXiv:1810.07052. doi:10.48550/arXiv.1810.07052.

Kohli, H., Parthasarathy, S., Sun, H., & Yao, Y. (2026). Loop, Think, & Generalize: Implicit Reasoning in Recurrent-Depth Transformers. COLM 2026. arXiv:2604.07822. doi:10.48550/arXiv.2604.07822.

Koishekenov, Y., Lipani, A., & Cancedda, N. (2025). Encode, Think, Decode: Scaling test-time reasoning with recursive latent thoughts. arXiv:2510.07358. doi:10.48550/arXiv.2510.07358.

Komatsu, T. (2022). Non-Autoregressive ASR with Self-Conditioned Folded Encoders. ICASSP 2022. doi:10.1109/icassp43922.2022.9746770.

Kuo, H.-Y., Chayti, E. M., Reizinger, P., Brendel, W., & Jaggi, M. (2026). Stabilizing Extrapolation in Looped Transformers via Learned Stochastic Stopping. NeurIPS 2026. arXiv:2606.29983. doi:10.48550/arXiv.2606.29983.

Lasbordes, M., Falavigna, D., & Brutti, A. (2025). Splitformer: An improved early-exit architecture for automatic speech recognition on edge devices. arXiv:2506.18035. doi:10.48550/arXiv.2506.18035.

Liang, J., Jin, X., & Pan, L. (2026). Beyond the Training Horizon: Mechanisms and Limits of Length Generalization in Looped Transformers. arXiv:2609.33144. doi:10.48550/arXiv.2609.33144.

Lu, W., Yang, Y., Lee, K., Li, Y., & Liu, E. (2025). Latent Chain-of-Thought? Decoding the Depth-Recurrent Transformer. First Workshop on the Application of LLM Explainability to Reasoning and Planning at COLM 2025. arXiv:2507.02199. doi:10.48550/arXiv.2507.02199.

Luo, X., & Yu, F. (2026). Prediction Dynamics in Depth-Recurrent Language Models. arXiv:2609.21383. doi:10.48550/arXiv.2609.21383.

McLeish, S., Li, A., Kirchenbauer, J., Kalra, D. S., Bartoldson, B. R., Kailkhura, B., Schwarzschild, A., Geiping, J., Goldstein, T., & Goldblum, M. (2025). Teaching Pretrained Language Models to Think Deeper with Retrofitted Recurrence. arXiv:2511.07384. doi:10.48550/arXiv.2511.07384.

Merrill, W., & Sabharwal, A. (2025). A Little Depth Goes a Long Way: The Expressive Power of Log-Depth Transformers. NeurIPS 2025. arXiv:2503.03961. doi:10.48550/arXiv.2503.03961.

Muennighoff, N., Yang, Z., Shi, W., Li, X. L., Fei-Fei, L., Hajishirzi, H., Zettlemoyer, L., Liang, P., Candes, E., & Hashimoto, T. (2025). s1: Simple test-time scaling. arXiv:2501.19393. doi:10.48550/arXiv.2501.19393.

Nozaki, J., & Komatsu, T. (2021). Relaxing the Conditional Independence Assumption of CTC-based ASR by Conditioning on Intermediate Predictions. INTERSPEECH 2021. arXiv:2104.02724. doi:10.48550/arXiv.2104.02724.

Olszowiec, P., Tomaszewska, P., Zelaszczyk, M., Byra, M., & Chung, N. C. (2026). Monitoring latent reasoning in looped language models. arXiv:2610.18127. doi:10.48550/arXiv.2610.18127.

Pappone, F., Crisostomi, D., & Rodola, E. (2025). Two-Scale Latent Dynamics for Recurrent-Depth Transformers. arXiv:2509.23314. doi:10.48550/arXiv.2509.23314.

Pfau, J., Merrill, W., & Bowman, S. R. (2024). Let's Think Dot by Dot: Hidden Computation in Transformer Language Models. arXiv:2404.15758. doi:10.48550/arXiv.2404.15758.

Popescu, A. C., Saez de Ocariz Borde, H., & Lio, P. (2026). Adaptive Depth in Looped Transformers: Diagnosing Learned Halting Gates and Trajectory Readouts. arXiv:2607.20519. doi:10.48550/arXiv.2607.20519.

Prairie, H., Novack, Z., Berg-Kirkpatrick, T., & Fu, D. Y. (2026). Parcae: Scaling Laws For Stable Looped Language Models. arXiv:2604.12946. doi:10.48550/arXiv.2604.12946.

Saunshi, N., Dikkala, N., Li, Z., Kumar, S., & Reddi, S. J. (2025). Reasoning with Latent Thoughts: On the Power of Looped Transformers. ICLR 2025. arXiv:2502.17416. doi:10.48550/arXiv.2502.17416.

Schuster, T., Fisch, A., Gupta, J., Dehghani, M., Bahri, D., Tran, V. Q., Tay, Y., & Metzler, D. (2022). Confident Adaptive Language Modeling. NeurIPS 2022. arXiv:2207.07061. doi:10.48550/arXiv.2207.07061.

Schwarzschild, A., Borgnia, E., Gupta, A., Huang, F., Vishkin, U., Goldblum, M., & Goldstein, T. (2021). Can You Learn an Algorithm? Generalizing from Easy to Hard Problems with Recurrent Networks. NeurIPS 2021. arXiv:2106.04537. doi:10.48550/arXiv.2106.04537.

Schwethelm, K., Rueckert, D., & Kaissis, G. (2026). How Much Is One Recurrence Worth? Iso-Depth Scaling Laws for Looped Language Models. arXiv:2604.21106. doi:10.48550/arXiv.2604.21106.

Snell, C., Lee, J., Xu, K., & Kumar, A. (2024). Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. arXiv:2408.03314. doi:10.48550/arXiv.2408.03314.

Tang, H., Liu, Z., Zeng, C., & Li, X. (2023). Beyond Universal Transformer: block reusing with adaptor in Transformer for automatic speech recognition. arXiv:2303.13072. doi:10.48550/arXiv.2303.13072.

Tur, Y., Naghiyev, J., Fang, H., Tsai, W.-C., Duan, J., Fox, D., & Krishna, R. (2026). Recurrent-Depth VLA: Implicit Test-Time Compute Scaling of Vision-Language-Action Models via Latent Iterative Reasoning. arXiv:2602.07845. doi:10.48550/arXiv.2602.07845.

Viakhirev, I., Borodin, K., Almutairi, A., Barannikov, S., Abramov, M., & Mkrtchian, G. (2026). Think Shallow, Solve Deep: Controlling Recurrent Dynamics for Reliable Test-Time Depth. arXiv:2608.18222. doi:10.48550/arXiv.2608.18222.

Wang, P., Hu, Y., Dong, G., Lin, Y., Wang, H., Liu, Z., Wen, Z., Li, J., Yang, J.-M., Wu, W., & Tong, Y. (2026). PAR: Progress-Aligned Recurrence for Length Generalization in State Tracking. arXiv:2610.27628. doi:10.48550/arXiv.2610.27628.

Wright, G. A., Cappellazzo, U., Zaiem, S., Raj, D., Ondel Yang, L., Falavigna, D., Ali, M. N., & Brutti, A. (2023). Training dynamic models using early exits for automatic speech recognition on resource-constrained devices. ICASSP 2024 Workshop on Self-supervision in Audio, Speech and Beyond. arXiv:2309.09546. doi:10.48550/arXiv.2309.09546.

Yang, X.-W., Han, Z., Zhang, X.-H., Wei, W.-D., Shao, J.-J., Guo, L.-Z., & Li, Y.-F. (2026). Stabilizing Recurrent Dynamics for Test-Time Scalable Latent Reasoning in Looped Language Models. ICML 2026. arXiv:2605.26733. doi:10.48550/arXiv.2605.26733.

Yoon, J. W., Woo, B. J., & Kim, N. S. (2022). HuBERT-EE: Early Exiting HuBERT for Efficient Speech Recognition. INTERSPEECH 2024. arXiv:2204.06328. doi:10.48550/arXiv.2204.06328.

Zhang, T., Hu, J., Peng, Y., & Xie, T. (2026). When Does Recurrence Become an Algorithm? Convergence Selection in Weight-Tied Looped Transformers. arXiv:2607.20594. doi:10.48550/arXiv.2607.20594.

Zhou, W., Xu, C., Ge, T., McAuley, J., Xu, K., & Wei, F. (2020). BERT Loses Patience: Fast and Robust Inference with Early Exit. NeurIPS 2020. arXiv:2006.04152. doi:10.48550/arXiv.2006.04152.

Zhu, R.-J., Wang, Z., Hua, K., Zhang, T., Li, Z., Que, H., Wei, B., Wen, Z., Yin, F., Xing, H., et al. (2025a). Scaling Latent Reasoning via Looped Language Models. arXiv:2510.25741. doi:10.48550/arXiv.2510.25741.

Zhu, R.-J., Peng, T., Cheng, T., Qu, X., Huang, J., Zhu, D., Wang, H., Xue, K., Zhang, X., Shan, Y., et al. (2025b). A Survey on Latent Reasoning. arXiv:2507.06203. doi:10.48550/arXiv.2507.06203.

Zhuang, X., Wang, S., Razzak, I., & Liu, W. (2026). What Makes Recurrence Effective in Looped Language Models? arXiv:2609.36636. doi:10.48550/arXiv.2609.36636.

© 2026 Pranay Mahendrakar. Licensed under CC BY 4.0.

DOI: 10.5281/zenodo.24094666