Learning Stable Update Rules: Iteration-Consistency Beyond the Training Horizon

M. Fikret Yalcinbas *

Posted Date: 3 March 2026

doi: 10.20944/preprints202603.0048.v1

Keywords: iterative neural networks; algorithmic generalization; time generalization; variable-depth computation; step-conditioned learning; adaptive computation

 doi:10.20944/preprints202603.0048.v1

Article

Learning Stable Update Rules: Iteration-Consistency Beyond the Training Horizon

M. Fikret Yalcinbas

Independent Researcher, Turkey; mfyalcinbas@gmail.com

Abstract

We study time-generalization in neural networks by training a shared iterative cell under explicit supervision of computation length. Rather than treating depth as fixed or learned implicitly, we provide a deterministic target step schedule and penalize deviations from the prescribed execution length, enabling controlled evaluation beyond the training horizon. We perform experiments on three representative dynamical regimes: contracting (Euclidean GCD), attractor-aligned (Log-Fibonacci), and expanding additive (Log-Factorial). We find that models can generalize to larger inputs when the effective iteration depth remains within the trained regime, yet often fail when required computation length increases, even if input magnitudes are moderate. Failure modes track the stability properties of the underlying update dynamics: contraction dampens errors, attractor alignment bounds them over finite horizons, and additive accumulation induces systematic drift. These results suggest that algorithmic generalization depends not only on function approximation but on the stability of learned update rules under composition. Explicit step conditioning improves interpretability and stability of computation depth, but does not by itself guarantee robust extrapolation to longer iterative chains.

Keywords: iterative neural networks; algorithmic generalization; time generalization; variable-depth computation; step-conditioned learning; adaptive computation

---

1. Introduction

1.1. Motivation

Neural networks often generalize in input space yet fail to generalize in computation length when solving tasks that require input-dependent iteration. This failure is easy to expose on simple algorithmic problems where the required number of update steps grows with input size: fixed-depth architectures can fit the training range by compressing iterative structure into shallow approximations, but then behave in a brittle way when the required depth exceeds the training regime. We study whether enforcing explicit iterative structure improves generalization in the time dimension. Rather than learning a direct input–output mapping, we train a repeated computational cell whose activity is modulated by a step profile derived deterministically from the input. The objective matches both the final output and the effective number of executed steps, encouraging step-consistent execution and penalizing shortcut solutions that solve large instances in a few updates.

We evaluate this framework on three representative dynamical regimes:

- **Log-Factorial**: an additive accumulation process in log space (error-accumulating dynamics). We also include a controlled variant where the per-step increment is provided to isolate long-horizon stability from increment discovery.
- **Log-Fibonacci**: a recurrence whose long-run behavior becomes dominated by a stable growth mode (attractor-aligned dynamics).
- **Euclidean GCD**: a norm-reducing iteration with highly variable step counts and well-known worst-case chains (contracting dynamics).

 doi:10.20944/preprints202603.0048.v1

These tasks separate input-magnitude extrapolation from iteration extrapolation. We find that models can generalize to larger input magnitudes in stable regimes, yet extrapolation in required step count can be fragile—suggesting that numerical scaling can generalize even when deeper iterative dynamics do not. Overall, explicit step conditioning improves stability and interpretability of computation depth, but does not by itself guarantee strong algorithmic generalization.

1.2. Related Work

Neural networks with adaptive or dynamic computation have been studied in several forms. Adaptive Computation Time (ACT) [1] introduced a mechanism allowing recurrent networks to learn input-dependent halting, enabling variable computational depth. Related ideas appear in Universal Transformers [2] and other recurrent-depth architectures that reuse parameters across refinement steps. A complementary line of work interprets deep networks as dynamical systems. Neural ODEs [3] interpret depth as the numerical integration of a continuous-time dynamical system, while Deep Equilibrium Models (DEQs) [4] define network outputs as fixed points of a learned iterative transformation. These approaches relax the constraint of fixed architectural depth, either through adaptive halting or implicit infinite-depth formulations.

Algorithmic reasoning benchmarks have been used to evaluate systematic generalization, including arithmetic, sorting, and formal language tasks [5,6]. Transformers applied to arithmetic and symbolic tasks [7,8] demonstrate strong interpolation performance but often exhibit brittle extrapolation when sequence length or iteration depth exceeds training regimes, motivating controlled probes of behavior beyond the training horizon.

2. Methods

2.1. Iterative Cell Architecture

We model computation as the repeated application of a shared neural cell (Figure 1), unrolled for a maximum of $T_{\max}$ steps. Given an input x (e.g., $x = n$ for log-factorial and log-Fibonacci, or $x = (a, b)$ for GCD), we first compute a fixed context vector

$$c = \mathrm{Ctx}_\phi(x), \tag{1}$$

where Ctx_ϕ is a small MLP producing $c \in \mathbb{R}^{d_c}$.

The recurrent state consists of (i) a continuous step counter $k_t \in \mathbb{R}$, (ii) an accumulator $s_t \in \mathbb{R}$ (also the default output), and (iii) a scratchpad vector $h_t \in \mathbb{R}^{d_h}$. We initialize

$$k_0 = 0, \quad s_0 = 0, \quad h_0 = \mathbf{0}. \tag{2}$$

At each step t, we compute a differentiable activity gate (soft halting)

$$\alpha_t = \sigma(\beta(T - k_t)), \tag{3}$$

where T is the target step count for the example, β controls gate sharpness, and $\sigma(\cdot)$ is the logistic sigmoid. Intuitively, $\alpha_t \approx 1$ while $k_t < T$ and decays toward 0 as k_t approaches T.

We then form the cell input

$$u_t = [\,k_t,\ s_t,\ h_t,\ c\,] \quad \text{(optionally augmented with a progress signal } p_t = k_t/(T + \varepsilon)), \tag{4}$$

and predict per-step updates with an MLP cell

$$[\Delta s_t,\ \Delta h_t] = f_\theta(u_t), \tag{5}$$

 doi:10.20944/preprints202603.0048.v1

shared across all steps. Updates are applied under the gate:

$$s_{t+1} = s_t + \alpha_t \, \Delta s_t, \qquad h_{t+1} = h_t + \alpha_t \, \Delta h_t. \tag{6}$$

Finally, the continuous counter advances proportionally to activity,

$$k_{t+1} = k_t + \alpha_t, \tag{7}$$

so the effective number of executed steps is $\sum_{t=0}^{T_{\max}-1} \alpha_t$. We take the final prediction as $\hat{y} = s_{T_{\max}}$.

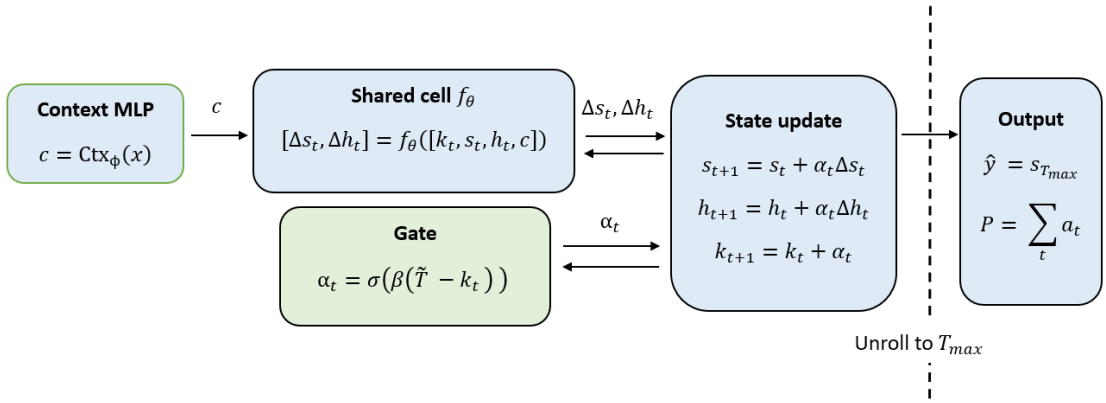

Figure 1. Iterative cell unrolled for up to $T_{\max}$ steps. Input x is embedded into a fixed context c. A shared cell produces gated updates to the accumulator s_t and scratchpad h_t. A differentiable activity gate α_t controls whether updates apply and defines the effective number of executed steps.

Default architecture parameters.

Unless otherwise specified, experiments use the following configuration:

- Context embedding dimension: $d_c = 32$
- Recurrent hidden (scratchpad) dimension: $d_h = 32$
- Cell MLP width: $w \in \{128, 256\}$
- Cell MLP depth: $L \in \{2, 4\}$

2.2. *Operator-Specific Stabilization*

For log-factorial, we include a controlled variant that supplies the analytic per-step increment - treating the latter as known to examine composition stability. Specifically, at step index k_t we add

$$\mathrm{term}(k_t) = \log(k_t + 1), \tag{8}$$

and restrict the learned component to a bounded residual that compensates for distortions introduced by soft halting. Concretely, the cell produces Δs_t but we convert it into a small bounded residual

$$r_t = \rho \, \tanh(\Delta s_t), \tag{9}$$

where ρ is a fixed constant. The accumulator is then updated as

$$s_{t+1} = s_t + \alpha_t \big(\mathrm{term}(k_t) + r_t\big). \tag{10}$$

We regularize the residual magnitude via $\sum_t r_t^2$, rather than $\sum_t \Delta s_t^2$, ensuring that the learned component remains a bounded perturbation of the analytic increment. For log-Fibonacci and GCD, we use the generic gated update without analytic assistance.

 doi:10.20944/preprints202603.0048.v1

2.3. Step-Consistency Objective

Training optimizes both output accuracy and step consistency. For a batch with targets y and target step counts T, the model returns $(\hat{y}, \text{aux})$, where aux includes the effective executed steps

$$P = \sum_{t=0}^{T_{\max}-1} \alpha_t. \tag{11}$$

We minimize a weighted sum of three losses:

$$\mathcal{L}_{\text{out}} = \text{MSE}(\hat{y}, y), \tag{12}$$

$$\mathcal{L}_{\text{steps}} = \text{MSE}\big(P,\ \max(T, T_{\max})\big), \tag{13}$$

$$\mathcal{L}_{\text{upd}} = \frac{1}{T_{\max}}\,\mathbb{E}\Big[\sum_{t=0}^{T_{\max}-1} \|\Delta s_t\|_2^2\Big], \tag{14}$$

where $\mathbb{E}[\cdot]$ is taken over the minibatch of training examples, and

$$\mathcal{L} = w_y\,\mathcal{L}_{\text{out}} + w_s\,\mathcal{L}_{\text{steps}} + w_u\,\mathcal{L}_{\text{upd}}. \tag{15}$$

The step-consistency term penalizes shortcut solutions: a model that produces the correct $\hat{y}$ while executing substantially fewer steps than prescribed incurs a penalty (e.g., it cannot solve a large-instance target in only a handful of iterations without increasing $\mathcal{L}_{\text{steps}}$). The update penalty acts as a mild stability regularizer, penalizing overly large per-step changes.

2.4. Target Step Schedules

A central ingredient is the deterministic step target $T(x)$ provided with each training example. This target defines the desired computation length and is used both to parameterize the activity gate (via T) and to supervise the effective executed steps $P = \sum_t \alpha_t$ in the step-consistency loss.

For log-factorial and log-Fibonacci with scalar input $x = n$, we set the target steps to scale with problem size,

$$T(n) = \max\big(\lceil g(n) \rceil,\ T_{\max}\big), \tag{16}$$

where $g(\cdot)$ is a monotone increasing schedule. In our experiments we use the identity schedule $g(n) = n$.

For Euclidean GCD with input $x = (a, b)$, the natural computation length is the number of Euclidean iterations; accordingly, we set

$$T(a, b) = \max\big(N_{\text{euclid}}(a, b),\ T_{\max}\big), \tag{17}$$

where $N_{\text{euclid}}(a, b)$ counts the number of modulo updates required by the standard Euclidean algorithm. In all cases, clipping ensures the supervised target is compatible with the finite unroll horizon $T_{\max}$.

3. Dynamical Regimes of Iterative Computation

The behavior of step-consistent iterative models depends not only on the input–output mapping being learned, but also on the stability of the underlying update dynamics under repeated composition. Abstractly, let an iterative computation evolve as

$$s_{t+1} = F(s_t; x, t), \tag{18}$$

where s_t denotes the recurrent state at step t and x is the input. When the same (or parameter-tied) update rule is applied for many steps, extrapolation to longer computation lengths is governed by how perturbations to s_t propagate over time. In particular, local stability properties of the update—e.g.,

 doi:10.20944/preprints202603.0048.v1

whether it is contracting, attractor-aligned, or integrative—strongly shape whether errors shrink, remain bounded, or accumulate.

We therefore use three representative tasks as proxies for distinct dynamical regimes. Although each task differs in semantics, they are chosen to expose qualitatively different error-propagation behavior in iterative computation: (i) norm-reducing contraction (GCD), (ii) convergence to a dominant growth mode (Fibonacci), and (iii) pure accumulation without stabilizing structure (log-factorial). Table 1 summarizes this taxonomy and the qualitative generalization behavior we expect. "Input-magnitude extrapolation" refers to generalization to larger input values at similar effective depth, whereas "step-count extrapolation" refers to generalization to longer computation chains than seen in training.

Table 1. Dynamical classification of tasks and their qualitative error-propagation behavior.

Task	Dynamical Type	Error Propagation	Generalization Behavior
GCD (Euclid)	Contracting	Shrinking / damped	Input-magnitude extrapolation; step-count fragile
Log-Fibonacci	Attractor-aligned	Bounded / aligned	Large-n with fixed depth; drift beyond horizon
Log-Factorial	Expanding additive	Accumulating	Horizon-sensitive; needs tight bias control

3.1. Contracting Dynamics: Euclidean GCD

The Euclidean algorithm updates integer pairs via

$$(a, b) \;\mapsto\; \big(b,\; a \bmod b\big), \tag{19}$$

iterating until $b = 0$. Under standard norms (e.g., $\|(a, b)\|_\infty = \max(|a|, |b|)$), the magnitude of the state decreases monotonically: after each iteration, the second component strictly decreases in magnitude, and the pair approaches $(\gcd(a, b), 0)$. This produces a contracting or norm-reducing computation in the sense that the state trajectory moves toward a small set near the origin, and perturbations are naturally damped as the state shrinks.

For learned iterative models, such contraction tends to support input-magnitude extrapolation: even if (a, b) exceed the training range, intermediate states often enter familiar magnitude regimes after a few iterations. However, contraction alone does not guarantee step-count extrapolation. Worst-case inputs (e.g., consecutive Fibonacci pairs) induce unusually long Euclidean chains, and models trained on shorter chains may fail once required depth exceeds the observed computation-length distribution.

3.2. Attractor-Aligned Dynamics: Log-Fibonacci

The Fibonacci recurrence admits a linear state-space form:

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}}_{A} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}. \tag{20}$$

This system has two eigenmodes: a dominant expanding mode with eigenvalue $\lambda_1 = \varphi > 1$, and a subdominant decaying mode with $|\lambda_2| < 1$. Any state can be decomposed as a linear combination of these two modes. Under repeated iteration, the component along the decaying eigenvector shrinks geometrically as λ_2^n, while the component along the dominant eigenvector grows as λ_1^n. As a result,

 doi:10.20944/preprints202603.0048.v1

trajectories align with the dominant growth direction, the ratio F_{n+1}/F_n converges to φ, and transient deviations from this direction vanish over time.

Using the eigen-decomposition of the Fibonacci state transition matrix, the sequence admits the closed-form representation

$$F_n = \alpha\varphi^n + \beta\psi^n, \tag{21}$$

where $\varphi = \frac{1+\sqrt{5}}{2}$ is the dominant eigenvalue, $\psi = \frac{1-\sqrt{5}}{2}$ is the subdominant eigenvalue with $|\psi| < 1$, and α, β depend on the initial conditions. As n increases, the term $\beta\psi^n$ decays geometrically to zero, yielding the asymptotic approximation

$$F_n \approx \alpha\varphi^n. \tag{22}$$

Taking logarithms,

$$\log F_n \approx \log\alpha + n\log\varphi, \tag{23}$$

which shows that in log space the increments $\log F_{n+1} - \log F_n$ become approximately linear in n with constant slope $\log\varphi$. Consequently, perturbations that misalign the state tend to be suppressed, which is consistent with bounded or aligned errors observed over long rollouts when depth is held fixed. However, iteration extrapolation remains sensitive to small systematic bias in the learned effective growth rate (e.g., an eigenvalue mismatch in the learned update). Such bias compounds multiplicatively with depth and can induce drift once the rollout horizon exceeds the regime encountered during training.

3.3. Expanding Additive Dynamics: Log-Factorial

In log space, factorial computation is an additive accumulation:

$$\log(n!) = \sum_{k=1}^{n} \log k. \tag{24}$$

Viewed as an iterative process, this corresponds to an integrator-like update

$$s_{t+1} = s_t + \log(t+1), \tag{25}$$

with steadily increasing increment magnitude. Unlike the GCD and Fibonacci regimes, there is no contraction toward a bounded set and no attractor manifold that suppresses perturbations. Errors introduced at any step persist and accumulate with depth. Under typical conditions, additive per-step errors lead to approximately linear growth of total error with the number of iterations.

This makes long-horizon behavior fragile: stable extrapolation to larger computation lengths requires either (i) learning the increment law with extremely low bias, or (ii) constraining the update so that drift cannot grow unchecked. For this reason, we treat log-factorial as an error-accumulating regime and include a controlled assisted variant: the analytic increment is supplied and the learned component is restricted to a bounded residual, decoupling accumulation stability from learning the increment law.

4. Experiments

4.1. Log-Fibonacci: Attractor Generalization and Drift

We study whether exposure to late, mostly-inactive iterations during training affects long-horizon stability on Log-Fibonacci. Inputs are integers n, with training on $n \in [1, 31]$ and evaluation on $n \in [32, 101]$. The model is an unrolled shared cell run for up to $T_{\max}$ iterations with a differentiable activity gate. We use the progress feature (`use_prog=True`), i.e., the cell input is augmented with a normalized progress signal $p_t = k_t/(T_{\text{target}} + \varepsilon)$.

We train two otherwise-identical models that differ only in the maximum unroll length used during training:

 doi:10.20944/preprints202603.0048.v1

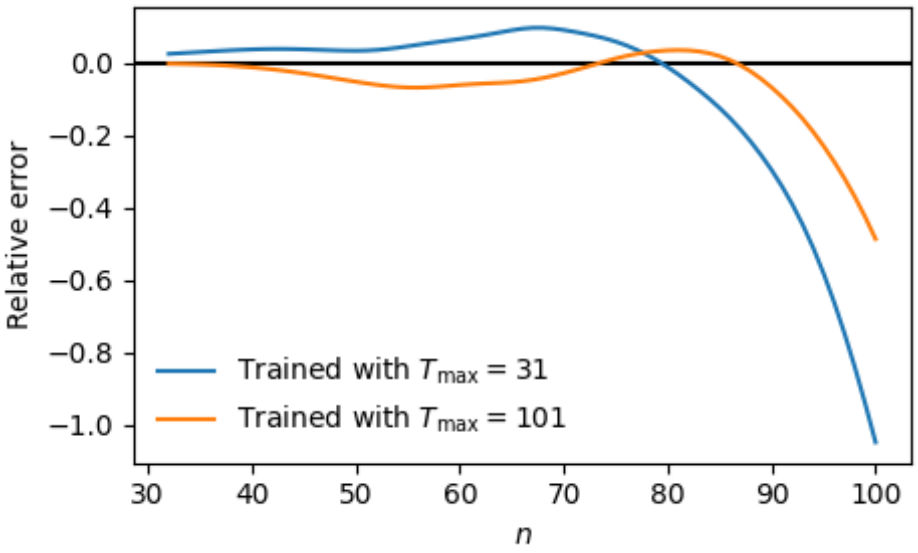

Figure 2. Test-range relative drift $(\hat{y} - y)/y$ for Log-Fibonacci, evaluated with $T_{\max} = 101$.

- **Full-horizon training:** $T_{\max} = 101$ (matches test unroll).
- **Short-horizon training:** $T_{\max} = 31$ (matches the training n-range).

At evaluation time, both models are unrolled to $T_{\max} = 101$ on both train-range ($n \in [1, 31]$) and test-range ($n \in [32, 101]$) inputs.

For both training regimes, the model's effective executed-step count tracks the target schedule nearly identically. In particular, the steps_used$/T_{\text{target}}$ curves overlap and remain close to 1 (roughly $1.03 \to 1.01$ across n).

Despite similar step usage, the relative error differs qualitatively (Figure 2):

- **Full-horizon training ($T_{\max} = 101$).** Predictions remain largely stable across n, with mild oscillations and a gradual negative drift emerging late in the range ($n \gtrsim 90$). Median: -3.68×10^{-2}; Mean: -5.97×10^{-2}; Absolute P90: 1.63×10^{-1}.
- **Short-horizon training ($T_{\max} = 31$).** Drift is stronger and appears earlier. The small-n regime exhibits larger positive oscillations, while the extreme tail ($n \gtrsim 95$) develops a more pronounced negative deviation. Median: 3.44×10^{-2}; Mean: -8.12×10^{-2}; Absolute P90: 4.62×10^{-1}.

Overall, training with a smaller $T_{\max}$ leads to more severe long-range drift when the same model is unrolled to the full horizon at test time. These results suggest that exposure to the late-iteration (mostly inactive) regime can help stabilize attractor-aligned tasks.

4.2. GCD: Input Magnitude vs. Step Extrapolation

The model was trained on integer pairs (a, b) sampled uniformly from the range $[1, 50]$. Evaluation was performed on pairs in the extended range $[1, 300]$. We fix the maximum allowed Euclidean iterations to $T_{\max} = 10$. As signed relative error is ill-conditioned for very small targets, test pairs were constructed such that

$$\gcd(a, b) > 20.$$

We first evaluated performance on random test pairs whose magnitudes exceed the training range but whose Euclidean step counts remain comparable to those seen during training. As shown in Figure 3, the model exhibits strong input-magnitude extrapolation in this regime, with predominantly small but occasionally negative deviation: Median: -5.55×10^{-2}; Mean: -1.66×10^{-1}; Absolute P90: 4.19×10^{-1}. This suggests that the learned computation is not tied to absolute input scale, but instead transfers to larger magnitudes provided the underlying algorithmic trajectory remains within the trained horizon.

 doi:10.20944/preprints202603.0048.v1

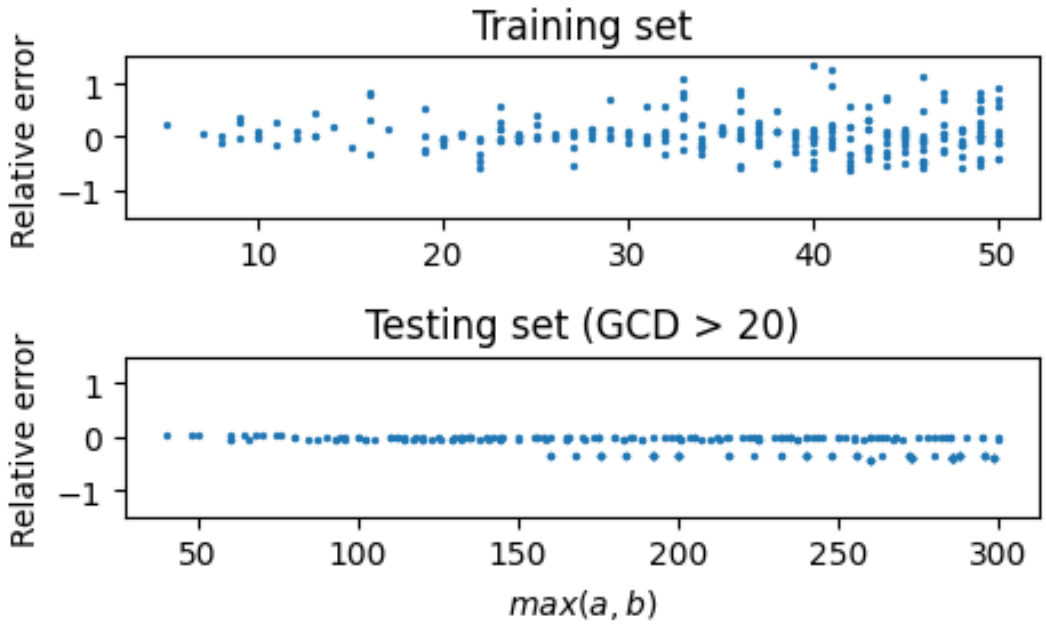

Figure 3. Input magnitude extrapolation for GCD, evaluated with $T_{\max} = 10$. The vertical axis shows signed relative error $(\hat{y} - y)/y$.

In contrast, when evaluation inputs require longer Euclidean iteration chains than those observed during training, performance degrades. The model does not reliably extrapolate to longer algorithmic trajectories, even when input magnitudes remain moderate. Because this failure mode is consistent and not yet systematically analyzed, we defer a detailed study of step-count extrapolation to a subsequent version of this work.

The GCD task illustrates a separation between two forms of generalization: the model successfully extrapolates to larger integer magnitudes when iteration depth is similar to training, but does not extrapolate to longer computational chains.

4.3. Log-Factorial: Default vs Dedicated Accumulation

We train on inputs $n \in [1, 30]$ and evaluate on $n \in [31, 150]$ under two model configurations that differ in how the accumulator is updated and regularized.

In the default configuration, the accumulator is updated purely through the learned increment Δa_t, and residual energy is tracked on the full predicted update. The model is trained with a large unroll horizon ($T_{\max} = 150$) to allow sufficient depth. Nevertheless, the signed relative error $(\hat{y} - y)/y$ increases approximately linearly with n beyond training horizon (Figure 4), suggesting unconstrained additive drift in this expanding (log-multiplicative) regime.

In contrast, the under the operator-specific accumulator variant, which mirrors the analytic log-sum structure of the target, the absolute relative error remains below 8.29×10^{-3} across the full test range while training only with $T_{\max} = 30$ (Figure 5), even with a simplified iterative cell.

5. Discussion

Prior work in neural algorithm learning typically evaluates whether a model approximates an algorithmic input–output mapping under distribution shift. Here we shift emphasis to *time-generalization* by studying a simple setting in which an iterative cell is unrolled for an explicitly specified number of steps, potentially far beyond the training horizon. We frame learned computation as repeated application of a shared update map, so the object of interest is the behavior of the update rule under iteration rather than only the terminal input–output fit.

 doi:10.20944/preprints202603.0048.v1

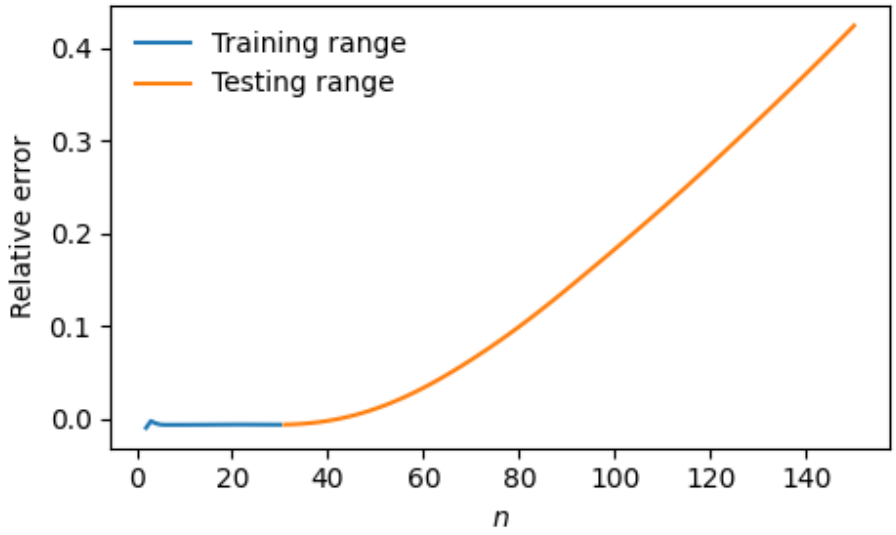

Figure 4. Log-Factorial relative error for default accumulation model, evaluated with $T_{\max} = 150$. The vertical axis shows signed relative error $(\hat{y} - y)/y$.

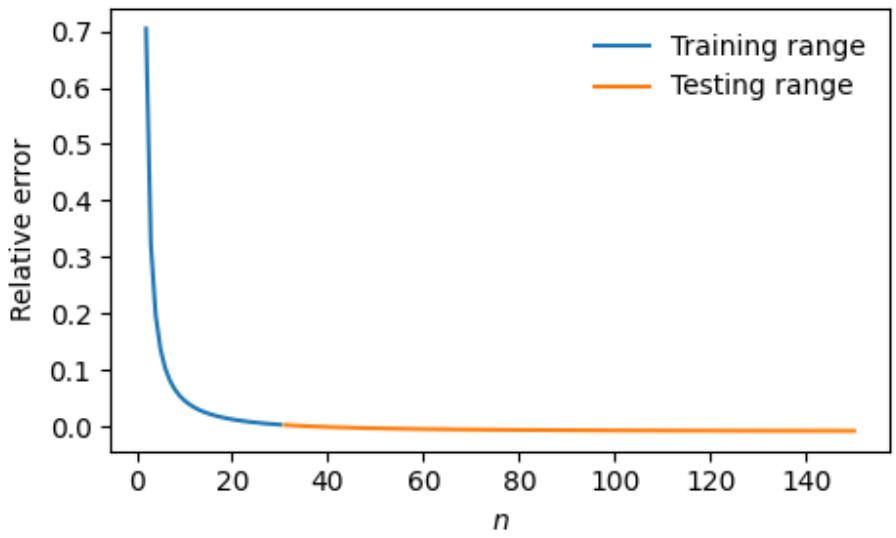

Figure 5. Log-Factorial relative error for dedicated accumulation model, evaluated with $T_{\max} = 150$. The vertical axis shows signed relative error $(\hat{y} - y)/y$.

 doi:10.20944/preprints202603.0048.v1

By treating computation length as a supervised signal, the model uses a controlled regime for probing stability beyond the training horizon. We can separate cases where failures reflect insufficient compute from cases where additional iterations actively degrade performance. The approach is complementary to Adaptive Computation Time (ACT). Whereas ACT learns to allocate computation per input—typically with pressure to reduce expected compute—step supervision enables direct tests of how performance evolves with depth and whether training objectives can induce stable behavior under longer unrolls.

Across tasks, input-magnitude extrapolation and step-count extrapolation decouple. When the underlying iteration is contracting, errors tend to damp, and the model can tolerate modest per-step imperfections, enabling strong performance on larger inputs that do not require longer chains. When the dynamics are expanding additive, small biases accumulate monotonically and horizon sensitivity emerges without additional constraints. Attractor-aligned behavior sits between these extremes: drift can remain bounded over the trained step horizon but grows once the model is asked to iterate beyond the regime it has learned to stabilize.

Taken together, this motivates evaluating algorithmic generalization along two axes: (i) input-range extrapolation at comparable effective depth, and (ii) explicit depth extension, including how error evolves over time. In particular, successes on large outputs can be misleading when those outputs arise from shallow computation chains (as in many GCD instances), while high-depth instances more directly expose the stability limits of the learned iteration.

5.1. Code Availability

The code is available at: *https://github.com/yfikret/iterative-operator-learner*. Interactive execution is available via Binder at *https://mybinder.org*.

This notebook includes the three core tasks (GCD, Log-Fibonacci, and Log-Factorial), training regimes (full and truncated horizons), and evaluation protocols. Figures in the paper can be generated from the provided scripts using fixed random seeds and configuration parameters.

Acknowledgments: The author acknowledges the use of automated tools for drafting support and conceptual discussions that contributed to improving the clarity and organization of this manuscript. All remaining errors or omissions are the sole responsibility of the author.

References

1. Graves, A. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983* **2016**.
2. Dehghani, M.; Gouws, S.; Vinyals, O.; Uszkoreit, J.; Kaiser, Ł. Universal transformers. *arXiv preprint arXiv:1807.03819* **2018**.
3. Chen, R.T.; Rubanova, Y.; Bettencourt, J.; Duvenaud, D.K. Neural ordinary differential equations. *Advances in neural information processing systems* **2018**, *31*.
4. Bai, S.; Kolter, J.Z.; Koltun, V. Deep equilibrium models. *Advances in neural information processing systems* **2019**, *32*.
5. Lake, B.M.; Ullman, T.D.; Tenenbaum, J.B.; Gershman, S.J. Building machines that learn and think like people. *Behavioral and brain sciences* **2017**, *40*, e253.
6. Nye, M.; Solar-Lezama, A.; Tenenbaum, J.; Lake, B.M. Learning compositional rules via neural program synthesis. *Advances in Neural Information Processing Systems* **2020**, *33*, 10832–10842.
7. Saxton, D.; Grefenstette, E.; Hill, F.; Kohli, P. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557* **2019**.
8. Trask, A.; Hill, F.; Reed, S.E.; Rae, J.; Dyer, C.; Blunsom, P. Neural arithmetic logic units. *Advances in neural information processing systems* **2018**, *31*.