How Much Can Loops Replace Parameters? Sharp Storage–Computation Frontiers for Looped Neural Networks

Anonymous authors
Paper under double-blind review

Abstract

Reusing neural modules across depth is common in recurrent networks, iterative refinement, and recurrent-depth language models, as it enables more computation without a proportional increase in parameter storage. Despite promising recent results, how the amount and timing of reuse jointly determine approximation accuracy remains largely open. We study multi-stage looped ReLU networks with J stored modules and T total calls. We first establish a sharp *storage–computation frontier*: for bounded unit-Lipschitz functions on $[0,1]^d$, the optimal worst-case uniform error is of order $(JT)^{-1/d}$ for fixed d and every $1 \le J \le T$, achieved by an explicit recurrent construction with $\widetilde{O}(d^2)$ parameters per module, where $\widetilde{O}$ hides logarithmic factors in d. Keeping the product JT fixed allows fewer stored modules to be offset by proportionally more calls, preserving the approximation order. We then show that this frontier is sensitive to *loop allocation*—how calls are distributed and ordered across modules: even two schedules with identical per-module call counts can attain different rates. The optimal $(JT)^{-1/d}$ rate is achieved precisely when a constant fraction of the call budget remains after a constant fraction of the modules has been visited. In particular, both calling each module for an equal number of consecutive steps and repeatedly cycling through the modules meet this condition, whereas spending most calls before reaching later modules may not. Together, these results characterize how parameter sharing and call order shape approximation under exact arithmetic.

1 Introduction

Reusing a neural module across depth allows a model to perform more computation without storing new weights at every step. Recurrent networks and deep equilibrium models repeatedly apply a shared state update (Elman, 1990; Bai et al., 2019). Algorithm unrolling and iterative refinement use closely related repeated-update structures, with either shared or step-specific parameters (Gregor & LeCun, 2010; Monga et al., 2021). Language models apply parameter sharing through cross-layer recurrence and recurrent depth (Lan et al., 2020; Dehghani et al., 2019; Geiping et al., 2025).

Despite this widespread use, how parameter sharing shapes approximation accuracy remains insufficiently understood. Practical looped architectures vary where and how modules are reused. Studies compare looping a middle block with looping the full stack (Saunshi et al., 2025, Section 3.3), and grouped with interleaved calls to shared blocks (Csordás et al., 2024, Section 3). These choices motivate two questions: *how much computation can replace parameter storage, and how does loop allocation—the distribution and order of calls across modules—affect approximation?*

We study uniform approximation of bounded unit-Lipschitz functions on $[0,1]^d$ using J stored ReLU modules and at most T calls, with $1 \le J \le T$. Each call evaluates one complete module, whose size depends only on d, not on the budgets or target accuracy. To address the first question, we consider multi-stage networks: each module is repeated within one stage before passing its state to the next module (Figure 1). For the second, we allow returns to earlier modules, including cyclic reuse, while keeping the call sequence independent of the input and target. Our main results characterize the storage–computation frontier and the call allocations that retain it.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

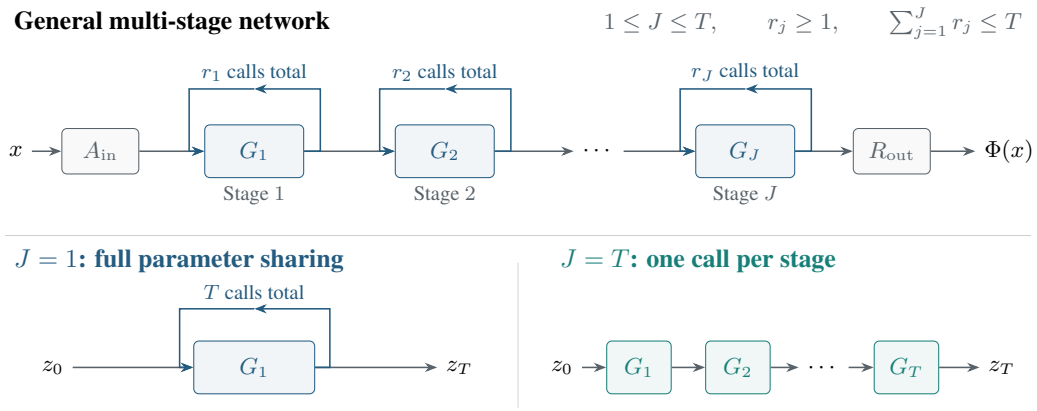


Figure 1: Multi-stage parameter sharing and its endpoints. Stage j reuses G_j for r_j calls. The endpoints use one shared module ($J = 1$) or one call per separately stored module ($J = T$). Input/output maps are omitted in the endpoint panels.

▷ **A sharp storage–computation tradeoff.** We establish matching upper and lower bounds of order $(JT)^{-1/d}$ for every $1 \le J \le T$. Keeping JT fixed trades stored modules for calls while preserving the approximation order. The two endpoints recover known regimes: $J = 1$ recovers the $T^{-1/d}$ scale of repeated fixed-size composition (Zhang et al., 2023), while $J \asymp T$ recovers the $T^{-2/d}$ scale of fully untied deep ReLU networks (Yarotsky, 2018). The upper-bound construction stores local target values in module parameters and decodes them over shifted fine grids; shared spatial features keep each dense module at width $O(d)$, hidden depth $O(\log(d+1))$, and $\widetilde{O}(d^2)$ parameters, adapting established spatial encoding and sequential decoding (Yarotsky & Zhevnerchuk, 2020). The lower bound combines signed Lipschitz grid data, which require VC dimension of order ε^{-d} at error ε, with the $O_d(JT)$ capacity allowed by the parameter and call budgets (Yarotsky, 2017; Goldberg & Jerrum, 1995).

▷ **Loop allocation: why module order matters.** Beyond the baseline multi-stage looped network, we allow any fixed call order, including returns to previously visited modules. The budgets J and T alone no longer determine the rate. We show that a sequence retains the $(JT)^{-1/d}$ frontier precisely when a constant fraction of the calls remains after a constant fraction of the modules has been reached. Thus calling each module for roughly T/J consecutive steps and cycling repeatedly through all modules both retain the frontier, whereas spending most calls before reaching later modules may not. The upper-bound construction loads target information on first visits and lets later calls execute a common decoder; the lower bound tracks how many parameter variables are available to each call, adapting VC-dimension techniques for piecewise-linear networks to shared modules and first-visit times (Bartlett et al., 2019).

2 Background and Related Work

2.1 Parameter Sharing in Architectures and Applications

Parameter sharing increases effective depth without storing new weights at every step, as in shared-layer and recurrent-depth language models (Lan et al., 2020; Dehghani et al., 2019; Geiping et al., 2025). Loops also support in-context algorithm learning (Yang et al., 2024) and combine with denoising flows or shortcut-consistency training (Suleymanzade et al., 2026; Jeddi et al., 2026). A related approach, algorithm unrolling, turns iterative solvers for signal and image processing into trainable shared or step-specific updates (Gregor & LeCun, 2010; Monga et al., 2021). Viewing depth as state evolution connects these designs to residual networks and neural ODEs (He et al., 2016; Chen et al., 2018). Equilibrium models define outputs through fixed points (Bai et al., 2019; 2020), while adaptive computation learns when to stop (Graves, 2016; Banino et al., 2021).

These designs raise questions about how much reuse helps and where it should occur. Studies compare sharing at matched depth (Schwethelm et al., 2026), its effects on reasoning and memorization (Saunshi et al., 2025), middle-layer versus full-stack repetition at matched parameter and FLOP budgets (Saunshi et al., 2025, Section 3.3), and interleaved versus grouped calls (Csordás et al., 2024,

Table 1: Uniform approximation of scalar functions under the original resource budgets. Recurrent sequence models are discussed in the related work.

Work	Target	Network map[a]	Width / depth / budget[a]	Guarantee / error rate[b]
Yarotsky (2018)	$\mathcal{C}_d$	FF_θ	$w = 2d + 10$, $h = O(P/d^2)$; P: NN parameter count	L^∞: $O_d(\omega_f(c_d P^{-2/d}))$
Zhang et al. (2023)	$\mathcal{C}_d$	$A_{\mathrm{out}} \circ G^{3r+2d-1} \circ A_{\mathrm{in}}$	$w_g = 4^{d+5}d$, $h_g = 2d + 3$; r: loop index	L^∞: $\leq 6\sqrt{d}\,\omega_f(r^{-1/d})$
Zhang (2026)	$\mathcal{H}_d^\alpha$	$\mathrm{FF}_{A\xi_f + a}$; $\xi_f \in \mathbb{R}^M$	$w \leq N_d$, $h \leq L_{d,P}$; $N_d = O(d3^d)$; M: target latent coordinates; P: NN parameter count	L^∞: $\asymp_{\alpha,d} [P \min\{M, P\}]^{-\alpha/d}$
This work[c]	$\mathcal{F}_d^{\mathrm{Lip}}$	$R_{\mathrm{out}} \circ G_J^{r_J} \circ \cdots \circ G_1^{r_1} \circ A_{\mathrm{in}}$	$w_g = 108d + 42$; $h_g = 8 + \lceil \log_2 d \rceil$; J: stored modules; T: total calls	L^∞: $\asymp_d (JT)^{-1/d}$

Notation. $\mathcal{C}_d = C([0,1]^d)$; $\mathcal{F}_d^{\mathrm{Lip}}$ and $\mathcal{H}_d^\alpha$ are bounded unit Lipschitz and Hölder classes on the cube ($0 < \alpha \leq 1$). ω_f is the modulus of continuity; $\omega_f(t) \leq t$ for unit-Lipschitz targets.

[a] $\mathrm{FF}_\theta = A_{h+1} \circ \rho \circ A_h \circ \cdots \circ \rho \circ A_1$, with affine A_i and coordinatewise ReLU $\rho(t) = \max\{t, 0\}$. A indicates an affine input map; R_{out} and G indicate bounded ReLU network modules; subscript g refers to one module; G^r is r-fold composition.

[b] In the affine-latent row, w, h denote width and hidden depth. P counts the generated network's parameter slots, but not the coefficients of the fixed generator (A, a). The matching rate requires $P \geq P_d^\star$ and $\min\{M, P\} \geq 4$; see Appendix A.5.

[c] $1 \leq J \leq T$, integer $r_j \geq 1$, and $\sum_j r_j \leq T$. Each module G_j has $\widetilde{O}(d^2)$ parameters. The affine input and ReLU output maps are evaluated once and add $\widetilde{O}(d^2)$ parameters and operations; $\widetilde{O}$ suppresses logarithmic factors in d.

Section 3). These comparisons motivate separating call counts from call order; their attention and input-dependent routing remain outside our ReLU model.

2.2 Theoretical perspectives on repeated computation

Universal approximation guarantees expressiveness but not resource requirements (Cybenko, 1989; Leshno et al., 1993). Matching approximation bounds establish optimal rates for a given function class, norm, and architecture. For Lipschitz targets at fixed dimension, P regular-grid values give error $O(P^{-1/d})$, whereas P-parameter deep ReLU networks attain $O(P^{-2/d})$ through potentially discontinuous weight selection (Yarotsky, 2018). Encoding grid values in real weights and decoding them through layers explains this gain (Yarotsky & Zhevnerchuk, 2020); see Section A for further background.

Accuracy can also improve with fixed module size: repeated ReLU composition and fixed recurrent neural networks (RNNs) establish this for continuous and one-dimensional targets, respectively (Zhang et al., 2023; Abadie et al., 2026). Sequence-model guarantees concern maps between sequences, under length, regularity, and error assumptions that differ from scalar uniform approximation (Schäfer & Zimmermann, 2007; Jiao et al., 2024; Xu & Sato, 2025). Other constructions separate target information from computation, either through a few target-dependent (intrinsic) parameters and decoders whose size depends on accuracy (Shen et al., 2022), or through composition over a fixed module vocabulary (Cai, 2024). Affine-latent theory generates the weights of a feedforward network with P parameter slots from M target-dependent coordinates through a fixed affine map, and gives a sharp joint minimax rate (Zhang, 2026). After inserting fixed ReLU interfaces to carry the state between calls, unrolling gives an affine tying of the stored weights, with all deployed parameter slots counted. This explains the connection between the lower bounds. However, the general affine-latent upper construction does not impose bounded-size recurrent modules or track when stored modules are first accessed; our construction and call-order theorem address these limitations. Table 1 compares uniform approximation of scalar functions under the original budgets; dimension-dependent costs appear in Sections A.4 and A.5.

The approximation rates above assume exact arithmetic, while storage can be measured in more than one way. Capacity bounds connect real parameter counts and computation to approximation lower bounds (Goldberg & Jerrum, 1995; Koiran & Sontag, 1998; Bartlett et al., 2019; Siegel, 2023), whereas encoding entropy measures the bits needed to describe an accurate approximant (Yarotsky & Zhevnerchuk, 2020; Gühring & Raslan, 2021). Quantized-weight networks involve a separate depth–precision tradeoff (Ou et al., 2024).

Beyond approximation rates, computational and dynamical analyses study what repeated updates can compute and retain. A looped Transformer can execute programs supplied in its input, which also serves as working memory (Giannou et al., 2023); a fixed recurrent ReLU network can approximate computable functions whose codes are supplied in its initial input (Bournez et al., 2025). Hard-

attention Transformers are Turing complete under arbitrary-precision computation (Pérez et al., 2021), while looped Transformers can simulate chain-of-thought steps using additional tokens and model capacity (Saunshi et al., 2025). Circuit bounds for fixed-depth, log-precision Transformers further clarify the role of depth and precision (Merrill & Sabharwal, 2023). Fixed-weight recurrent models also approximate continuous functions from prompts whose length supplies both target information and recurrent steps (Petrov et al., 2024). From a dynamical perspective, contractive RNNs forget older inputs (Miller & Hardt, 2019), equilibrium and modern Hopfield networks converge to fixed points or attractors (Winston & Kolter, 2020; Ramsauer et al., 2021), and idealized self-attention exhibits long-time clustering (Geshkovski et al., 2025). Implicit-model theory characterizes locally Lipschitz maps as fixed points of target-dependent operators shared across iterations, subject to regularity and contraction conditions but no finite ReLU parameter budget (Liu et al., 2026). Together, these results describe both the expressive benefits of iteration and the information it can lose.

These perspectives suggest that both the stored information and the time at which it becomes available can matter, but they do not quantify the joint effect within a bounded-size shared-module network. We therefore ask how the best uniform approximation error for bounded Lipschitz functions scales with J stored ReLU modules and T calls, and which fixed call orders preserve that scale. Our results answer both questions while keeping the dimension-dependent local costs explicit.

3 MULTI-STAGE LOOPED NETWORKS

We first formalize the baseline multi-stage model in Figure 1. The number J counts distinct stored modules, while T bounds their complete evaluations. The size of each local component is bounded in terms of d alone, independently of J, T, and the target accuracy.

Stages and calls. Write G^r for r-fold composition. A multi-stage network is

$$\Phi = R_{\mathrm{out}} \circ G_J^{r_J} \circ \cdots \circ G_1^{r_1} \circ A_{\mathrm{in}}, \qquad 1 \le J \le T, \quad r_j \ge 1, \quad \sum_{j=1}^{J} r_j \le T. \tag{1}$$

The affine map $A_{\mathrm{in}} : \mathbb{R}^d \to \mathbb{R}^m$ initializes a state $z \in \mathbb{R}^m$. Each ReLU module $G_j : \mathbb{R}^m \to \mathbb{R}^m$ updates this state, and the ReLU readout $R_{\mathrm{out}} : \mathbb{R}^m \to \mathbb{R}$ returns the approximation. Stage j applies G_j for r_j consecutive calls before passing the state to the next stage. A module call evaluates all of its internal layers; the input and output maps are each evaluated once. Repeated calls reuse the same parameters. Distinct stored modules are counted separately. Thus $J = 1$ gives full sharing. When $J = T$, the budget forces $r_j = 1$ for every j, giving one separately stored module per call.

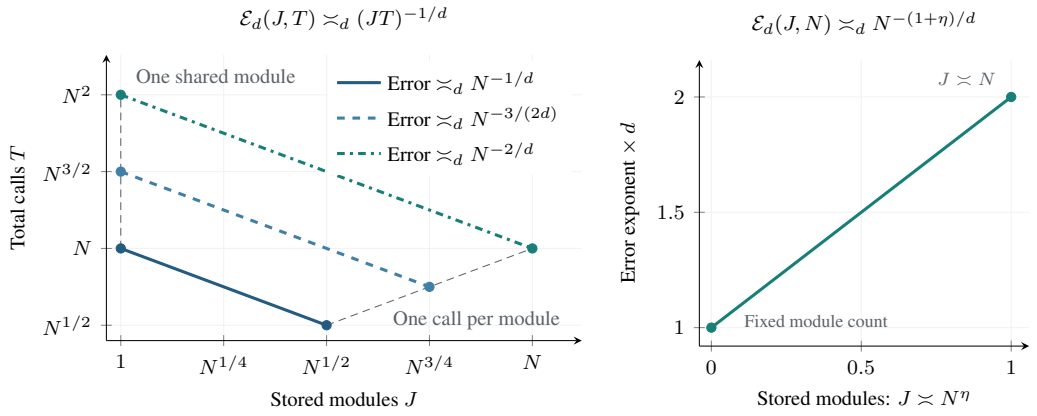
Module size and resource counts. We fix an upper bound $m \le m_d = O(d)$ on the state dimension, and bounds $q_d, u_d = \widetilde{O}(d^2)$ on the stored parameters and scalar operations of each module and each input or readout map. Here $\widetilde{O}$ suppresses logarithmic factors in d. The constants in these bounds are fixed throughout; the constructions below satisfy them. Let P count all stored scalar parameters, including the input and output maps, and let U count all executed scalar arithmetic and ReLU operations. Then $P \le q_d(J+2)$ and $U \le u_d(T+2)$. Thus J controls parameter storage and T controls executed computation, up to dimension-dependent local costs.

Functions and error. Fix $d \ge 1$ and consider $\mathcal{F}_d^{\mathrm{Lip}} = \{f : [0,1]^d \to [-1,1] : |f(x) - f(y)| \le \|x - y\|_2 \text{ for all } x, y\}$. We measure error by $\|f - \Phi\|_\infty = \sup_{x \in [0,1]^d} |f(x) - \Phi(x)|$. A common Lipschitz bound $L_0 \ge 1$ changes only the error constant: apply the unit-Lipschitz result to f/L_0 and rescale the output. We write $a \lesssim_d b$ for $a \le C_d b$, where $C_d > 0$ depends only on d and the fixed module-size bounds; $\gtrsim_d$ reverses the inequality and $\asymp_d$ means both hold. Other subscripts allow the indicated additional dependence.

Worst-case approximation error. For fixed local size bounds m_d, q_d, u_d and budgets J, T, define

$$\mathcal{E}_d(J,T) = \inf_{\mathcal{A}} \sup_{f \in \mathcal{F}_d^{\mathrm{Lip}}} \inf_{\theta} \|f - \Phi_{\mathcal{A},\theta}\|_\infty. \tag{2}$$

Here the outer optimization ranges over admissible contiguous multi-stage designs of the form (1) within these size bounds. A design $\mathcal{A}$ specifies the architectures of A_{in}, R_{out}, and the modules $G_1, \ldots, G_J$, the stage lengths $r_1, \ldots, r_J$, and all target-independent coefficients. These choices may depend on (d, J, T) but not on the target function. Once $\mathcal{A}$ is fixed, its counted target-dependent real parameters θ are fitted separately to each target. Thus the outer infimum may choose the stage lengths, while calls to each module remain in one consecutive stage.



(a) Storage–computation tradeoff (b) Accuracy at fixed calls

Figure 2: Two views of the storage–computation tradeoff. (a) At a fixed error order, more stored modules require fewer calls. (b) At a fixed call budget $T = N$, storing $J \asymp N^\eta$ modules gives error of order $N^{-(1+\eta)/d}$. Both panels fix d and the local size bounds; N is a common reference scale.

4 The approximation frontier of multi-stage loops

Theorem 1 (Uniform approximation frontier)**.** *For all integers $T \geq 1$ and $1 \leq J \leq T$, the optimal worst-case uniform approximation error satisfies*

$$c_d(JT)^{-1/d} \leq \mathcal{E}_d(J,T) \leq C\sqrt{d}\,(JT)^{-1/d}, \tag{3}$$

where $c_d > 0$ depends only on d and the fixed local size bounds, and $C > 0$ is absolute. The upper bound is attained by a fixed local ReLU architecture, independent of J and T, with dense modules of state dimension and width $O(d)$, hidden depth $O(\log(d+1))$, and $O(d^2\log(d+1))$ parameters per module. The input and output maps are stored and evaluated once and together add $O(d^2\log^2(d+1))$ parameters and operations. Parameter counts include all matrix entries and biases. The lower bound applies to every network under the same local size bounds.

Trading storage for computation. The product JT acts as the effective approximation budget. At fixed dimension, the relation $JT \asymp_d \varepsilon^{-d}$ describes the tradeoff between storage and computation at accuracy ε: using fewer stored modules requires proportionally more calls. Equivalently,

$$T \asymp_d \frac{\varepsilon^{-d}}{J}, \quad \forall\, 1 \leq J \lesssim_d \varepsilon^{-d/2},$$

up to integer rounding. The endpoints make this tradeoff concrete: a single shared module needs $T \asymp_d \varepsilon^{-d}$ calls, while one call per module gives $J = T \asymp_d \varepsilon^{-d/2}$. For error $N^{-1/d}$, the intermediate choice $(J,T) \asymp (N^{1/4}, N^{3/4})$ lies between $(1,N)$ and $(N^{1/2}, N^{1/2})$; all three have $JT \asymp N$, as shown in Figure 2(a). Including the input and output maps, the construction uses $\widetilde{O}(Jd^2)$ stored parameters and $\widetilde{O}(Td^2)$ scalar operations; Equation (19) in Appendix B.2 gives the explicit totals.

The approximation cost of sharing. At a fixed call budget T, the cost of sharing appears in three representative rates:

$$\mathcal{E}_d(1,T) \asymp_d T^{-1/d}, \qquad \mathcal{E}_d(J,T) \asymp_d T^{-3/(2d)} \text{ for } J \asymp T^{1/2}, \qquad \mathcal{E}_d(T,T) \asymp_d T^{-2/d}. \tag{4}$$

The middle case illustrates partial sharing. More generally, Figure 2(b) shows the full interpolation: if $J \asymp T^\eta$, the error is $T^{-(1+\eta)/d}$. Relative to the untied endpoint, parameter sharing incurs an error ratio $\mathcal{E}_d(J,T)/\mathcal{E}_d(T,T) \asymp_d (T/J)^{1/d}$. Consequently, matching the untied error within a budget-independent constant requires $J \asymp T$. $J \asymp T/\log T$ has the same polynomial exponent but still loses a factor $(\log T)^{1/d}$. At $J = 1$, the rate matches the uniform upper bound for repeated fixed-size composition (Zhang et al., 2023, Theorem 1.3); the matching lower bound follows from Theorem 1. The untied endpoint agrees with the optimal deep-ReLU scale (Yarotsky, 2018).

4.1 PROOF SCHEME

The upper bound stores encoded local function values in module parameters and recovers them through repeated calls, adapting real-weight encoding and digit extraction (Yarotsky, 2018; Yarotsky & Zhevnerchuk, 2020; Zhang et al., 2023). Sharing spatial features across shifted grids ensures uniform accuracy with small modules. The lower bound combines Lipschitz grid packings with parameter–runtime bounds on VC dimension (Yarotsky, 2017; Goldberg & Jerrum, 1995). The complete proof appears in Appendix B.

Upper bound. We first construct a uniform approximant for a given grid, then fit its resolution to the storage and call budgets.

(i) Encode and decode local values. Divide the domain into B coarse blocks, each containing F fine cells. The fine cells have side length of order $(BF)^{-1/d}$, so Lipschitz continuity bounds the variation within each cell by $O(\sqrt{d}\,(BF)^{-1/d})$. Along a path of adjacent cells, grid-scale quantization makes successive integer values differ by only -1, 0 or 1. Each block's first value (the anchor) and increments are stored in a real-valued code. The network scans the blocks to select the input's code, then repeats the last module to recover the value for its fine cell, reading one digit per update. Loading takes B calls, anchor recovery takes $L = O(\log(BF + 1))$, and fine-cell traversal takes F. Figure 5 in Appendix B illustrates this one-grid process. Longer codes therefore require more calls without enlarging the module (Section B.1 and Lemma 5).

(ii) Ensure uniform accuracy. Continuous selectors may mix codes near grid boundaries. We combine $2d + 1$ shifted grids so that at least $d + 1$ predictions are accurate at every input, then take their median with a ReLU output network. A coordinatewise median-of-three construction provides a related boundary-repair precedent (Lu et al., 2021, Lemma 3.4); the simultaneous-shift guarantee used here is proved in Lemma 4. The copies share coordinate features while retaining separate codes and decoded values, keeping state dimension and width at $O(d)$. All copies advance in one call; the median is evaluated once. The module realization has depth $O(\log(d + 1))$ (Section B.2).

(iii) Fit the resource budgets. The construction gives

$$\text{error} \le C\sqrt{d}\,(BF)^{-1/d}, \qquad \text{stored modules} = B, \qquad \text{calls} = B + L + F. \tag{5}$$

For sufficiently large budgets, choose $B \asymp_d J$ while reserving a constant fraction of calls for fine-cell updates, giving $BF \gtrsim_d JT$. The budget conversion and small-budget completion extend the bound to every $1 \le J \le T$ (Section B.3 and Theorem 9). At $J = 1$, the same module loads and decodes. At $J = T$, about $T/8$ modules supply codes, up to grid rounding; splitting repeated calls into separately stored copies and adding identity stages where needed gives one call per stage (Lemma 7).

Lower bound. We compare the labelings required for approximation with network capacity.

(i) Approximation requires many labelings. On a suitably spaced grid of order ε^{-d} points, every assignment of values $\pm\varepsilon$ extends to a unit-Lipschitz function. Uniform error below ε requires realizing every binary labeling of these points, hence VC dimension at least of order ε^{-d} (Yarotsky, 2017, proof of Theorem 4, Eq. (38)).

(ii) Storage and calls limit capacity. For each fixed design $\mathcal{A}$, the parameter–runtime bound gives (Goldberg & Jerrum, 1995, Theorem 2.3)

$$\mathrm{VCdim}\{x \mapsto \mathbf{1}[\Phi_{\mathcal{A},\theta}(x) > 0] : \theta\} \le C(P + 1)(U + 1) \le C_d JT.$$

The last inequality uses the fixed local size bounds: stored parameters scale with J, and executed computation scales with T. Combining the two estimates gives $\varepsilon^{-d} \lesssim_d JT$, hence the lower rate $(JT)^{-1/d}$. This holds for every admissible contiguous multi-stage design, whereas the upper bound uses a particular call allocation. The full lower-bound argument appears in Section B.4.

5 HOW CALL ALLOCATION AFFECTS APPROXIMATION

Looped architectures differ not only in how often they reuse modules, but also in how they arrange the calls. For example, interleaved $(G_1, G_2, G_1, G_2, G_1, G_2)$ and grouped $(G_1, G_1, G_1, G_2, G_2, G_2)$ calls have been compared for two shared blocks (Csordás et al., 2024). We therefore extend the multi-stage model to any predetermined sequence of T calls to J stored modules, allowing returns

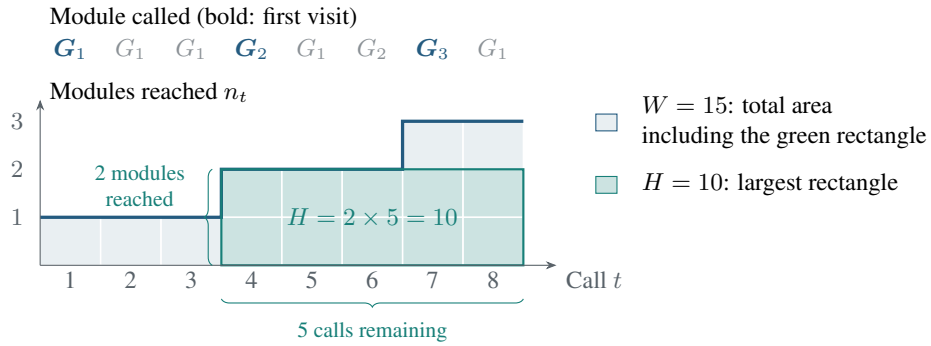


Figure 3: Access curve for $\sigma = (1, 1, 1, 2, 1, 2, 3, 1)$. First visits raise the curve; repeated visits leave it unchanged. The total area is $W = 15$, and the largest right-aligned rectangle has area $H = 2 \times 5 = 10$. Remaining calls include the current call.

to earlier modules. We ask which sequences retain the storage–computation frontier and whether changing call order alone can change the approximation rate.

Fix $\sigma = (\sigma_1, \ldots, \sigma_T)$, where $\sigma_t \in \{1, \ldots, J\}$ identifies the module called at step t. The sequence is independent of the input and target, and each module appears at least once. All calls to the same module share its parameters:

$$z_0 = A_{\text{in}}(x), \qquad z_t = G_{\sigma_t}(z_{t-1}), \qquad \Phi(x) = R_{\text{out}}(z_T). \tag{6}$$

For fixed σ, let $\mathfrak{D}_d(\sigma)$ be the admissible network designs, including target-independent coefficients, under the same local size bounds as before. As in (2), the optimal worst-case error for this prescribed sequence is

$$\mathcal{E}_d(\sigma) := \inf_{\mathcal{A} \in \mathfrak{D}_d(\sigma)} \sup_{f \in \mathcal{F}_d^{\text{Lip}}} \inf_{\theta} \|f - \Phi^{\sigma}_{\mathcal{A},\theta}\|_\infty, \tag{7}$$

where $\Phi^{\sigma}_{\mathcal{A},\theta}$ denotes the network above. The design is chosen for the whole function class, while θ is fitted separately to each target function.

Beyond the total storage and call budgets, the sequence determines when each module first becomes available. Calls made before a module's first visit cannot use its parameters; later calls can build on the information it has introduced into the state. We therefore track how many modules have been visited at each call and how much computation remains once they are available.

Relabel modules by first appearance and define $\tau_j = \min\{t : \sigma_t = j\}$, so $1 = \tau_1 < \cdots < \tau_J$. Let $n_t = \#\{j : \tau_j \le t\}$ count the modules visited by call t. The step function $t \mapsto n_t$ is the *access curve*, illustrated in Figure 3. We measure both its total area W and the area H of its largest right-aligned rectangle:

$$W(\sigma) = \sum_{t=1}^{T} n_t, \quad H(\sigma) = \max_{1 \le t \le T} n_t(T - t + 1). \tag{8}$$

Here $T - t + 1$ includes call t. Since $n_t \le J$, we also have $W(\sigma) \le JT$.

Theorem 2 (Uniform approximation with a prescribed call sequence)**.** *For every prescribed sequence σ of T calls that visits all J modules,*

$$c_d W(\sigma)^{-1/d} \le \mathcal{E}_d(\sigma) \le C\sqrt{d}\, H(\sigma)^{-1/d}, \tag{9}$$

where $c_d > 0$ depends only on d and the fixed local size bounds, and $C > 0$ is absolute. In particular, every prescribed sequence satisfies $\mathcal{E}_d(\sigma) \ge c_d(JT)^{-1/d}$ as $W(\sigma) \le JT$. Moreover,

$$W(\sigma) \le H(\sigma)\left[1 + \log \frac{JT}{H(\sigma)}\right].$$

Thus the ratio of the upper to the lower bound is at most $C_d[1 + \log(JT/H(\sigma))]^{1/d}$, where C_d depends only on d and the fixed local size bounds (Equation (31) in Appendix C.3). The upper bound is attained with state dimension and module width $O(d)$, hidden depth $O(\log(d+1))$, and $\widetilde{O}(d^2)$ parameters per module, while executing every call in σ. Exact dimensions are given in Lemma 13 of Appendix C.1.

When is the frontier retained? When $H(\sigma) \asymp W(\sigma)$, the bounds give a matching approximation rate. For other sequences, they bound the error but need not determine its exact order. Specifically, for fixed d and varying J, T, the bounds above give a criterion for retaining the $(JT)^{-1/d}$ frontier:

$$\mathcal{E}_d(\sigma) = O((JT)^{-1/d}) \quad \iff \quad W(\sigma) = \Omega(JT) \quad \iff \quad H(\sigma) = \Omega(JT). \tag{10}$$

For every $1 \le J \le T$, three families of call sequences retain the frontier: *balanced grouped calls* (roughly T/J consecutive calls per module), *repeated cycles* (with a final partial cycle if needed), and *an initial full pass* (visit all J modules once before any further reuse). All three families have $H(\sigma) \asymp W(\sigma) \asymp JT$ and therefore attain error of order $(JT)^{-1/d}$. Their access curves are shown in Figure 6(b–d) of Appendix C.3.

These patterns arrange the calls differently, but share one feature: *a fixed fraction of the modules has been visited while a fixed fraction of the calls remains*. Those calls can use information supplied by the modules already reached, even if the sequence returns to an earlier one. More generally, this condition is necessary and sufficient for retaining the frontier, with fractions that may depend on the constant in the error bound. The precise statement is proved in Corollary 15 of Appendix C.3.

Same counts, different orders. The condition asks whether enough calls remain after many modules have been visited. Per-module call counts alone cannot answer this: the two sequences below have identical counts for every module. Both call G_1 $T - J + 1$ times and every other module once, but place the extra calls to G_1 before or after the other modules are reached:

$$\sigma_A = \big(\underbrace{1, \dots, 1}_{T-J+1 \text{ calls}}, 2, \dots, J \big), \qquad \sigma_B = \big(1, 2, \dots, J, \underbrace{1, \dots, 1}_{T-J \text{ calls}}\big). \tag{11}$$

Modules $2, \dots, J$ are first visited after $T - J + 1$ calls in σ_A but within the first J calls of σ_B. Consequently, H and W are both of order $T + J^2$ for σ_A, and of order JT for σ_B. The bounds of Theorem 2 match in both cases, giving

$$\mathcal{E}_d(\sigma_A) \asymp_d (T + J^2)^{-1/d}, \qquad \mathcal{E}_d(\sigma_B) \asymp_d (JT)^{-1/d}. \tag{12}$$

For $J \asymp \sqrt{T}$, the two error orders are $T^{-1/d}$ and $T^{-3/(2d)}$. The extra calls to G_1 thus support a better approximation rate when the other modules have already been visited. In σ_A, the long initial loop can use information stored in G_1, but not information first encountered in G_J.

These are best-approximation rates for separately designed and fitted networks, as specified by (7); they do not compare two call orders of one trained network. For this pair, the order gap appears only in the intermediate regime when both J and T/J grow without bound, such as $J \asymp \sqrt{T}$. If J stays bounded, every sequence has error of order $T^{-1/d}$; if $J \asymp T$, every sequence has error of order $T^{-2/d}$, matching the untied deep-ReLU endpoint (Yarotsky, 2018). Thus, with $J \asymp T$, call order may affect constants but cannot change the minimax approximation rate.

5.1 Proof scheme

We adapt the decoder to the prescribed call sequence for the upper bound and track parameter access in the capacity estimate for the lower bound. The complete proof appears in Appendix C.

Upper bound. The rectangle attaining H pairs the modules already visited with the calls still available. Moving the loading cutoff earlier preserves the product of these two counts up to a constant and reserves calls for loading. For sufficiently large budgets, the grid-budget lemma fits loading and decoding into the remaining calls and gives B coarse blocks and F fine cells with $BF \gtrsim_d H$. A state-dependent gate makes repeated visits before the last load preserve the state; afterward every called module performs the common decoding update, including returns to earlier modules. Calls beyond those needed for decoding preserve the result. The resulting error is $O_d(H^{-1/d})$. When this grid cannot be formed, H is bounded in terms of d, and constant output gives the same bound (Appendix C.1).

Lower bound. Call t can involve only the n_t modules visited by then; calling one again adds computation but no new parameter variables. Grouping calls with comparable values of n_t lets activation-pattern and polynomial-sign counting charge the main parameter–operation cost to $\sum_t n_t = W$ (Bartlett et al., 2019, Section 4 and Lemma 17). The capacity lemma also controls an additional logarithmic term and gives VC dimension $O_d(W)$ (Lemma 14 in Appendix C.2). The same Lipschitz grid argument as before then yields error $\gtrsim_d W^{-1/d}$.

6 EXTENSIONS TO OTHER NORMS AND REGULARITY

We also extend the multi-stage approximation result (Theorem 1) to finite L^p error and Hölder targets.

Finite L^p error. For $1 \le p < \infty$ with Lebesgue measure on the cube, bounded errors on sufficiently thin transition regions can be tolerated, so a single decoder and an affine readout replace the shifted copies and median. A single repeated module already attains the Lipschitz $T^{-1/d}$ rate with five hidden layers (Zhang et al., 2023, Theorem 1.1). Our multi-stage construction attains the joint $(JT)^{-1/d}$ upper rate for all $1 \le J \le T$ with constant-depth modules and one decoding update per call. Its full costs are $P = O((J+1)d^2)$ and $U = O((T+1)d^2)$ (Proposition 11). The construction and proof appear in Section B.5.

Hölder targets with uniform error. For $[-1,1]$-valued targets satisfying $|f(x) - f(y)| \le \|x-y\|_2^\alpha$, $0 < \alpha \le 1$, rescaling the quantization yields the sharp uniform rate $(JT)^{-\alpha/d}$ without changing the decoder architecture or resource counts (Corollary 12). Thus lower regularity slows approximation, while $\alpha = 1$ recovers the Lipschitz result. The proof appears in Section B.6.

7 DISCUSSION

Together, our results answer how much repeated computation can replace stored parameters in these looped networks. For bounded unit-Lipschitz targets, J stored modules and T calls attain the sharp uniform-error order $(JT)^{-1/d}$ when the calls are suitably arranged; keeping JT fixed lets more calls compensate for fewer modules. The arrangement matters because a module's parameters cannot be used before its first call. Even with identical call counts per module, delaying access to some modules can worsen the rate, whereas returning to an earlier module after the others have been reached can preserve the frontier.

The frontier counts real-valued parameters under exact arithmetic, as is standard in these approximation results (Yarotsky, 2018; Yarotsky & Zhevnerchuk, 2020; Zhang et al., 2023). Thus a small parameter count need not mean a short description of the target. If the complete target-dependent parameter assignment is limited to I bits, the minimax error is of order $[1 + \min\{I, JT\}]^{-1/d}$ (Appendix D.1). Once $JT > I$, more calls cannot improve this order because there is no more target information to decode. This bit limit does not address the stability of finite-precision execution. Universal approximation with floating-point operations, including interval guarantees for rounded target maps, has been studied (Park et al., 2024; Hwang et al., 2025); obtaining our storage–computation rates under such execution remains open.

Limitations and future work. The main theorems concern the best worst-case approximation of bounded Lipschitz functions by looped neural networks built from bounded-size ReLU modules. For each budget, the architecture and call schedule are fixed across the function class, while the target-dependent weights may be chosen separately for each function. The bounds do not show that training finds those weights. They also do not guarantee that running the same trained network for more calls improves its output at test time.

Stored parameters and module calls do not directly determine inference latency or energy, which also depend on hardware and implementation. The call sequence is fixed across inputs and targets; adaptive stopping or input-dependent routing would add controller and execution costs. Finally, the $(JT)^{-1/d}$ rate describes how error scales at fixed d; dimension-dependent constants may matter at practical budgets. Extending the analysis to adaptive schedules, structured target classes, or average error under a data distribution would address some of these gaps.

For looped language models, the call-order result raises a concrete question: at matched storage and computation budgets, does it matter when each block is first called? Empirical studies have compared middle with full-stack looping (Saunshi et al., 2025, Section 3.3) and grouped with interleaved calls (Csordás et al., 2024, Section 3). In our ReLU model, balanced grouped calls and repeated cycles attain the same $(JT)^{-1/d}$ approximation order. These bounds do not rank language-model call sequences, where attention, token interactions, and prediction loss lie outside our model. Testing whether first-access timing helps explain language-model performance would require training each sequence separately under matched conditions.

AI USE STATEMENT

Agentic AI tools were used to assist with basic literature searches, provide feedback on the authors' theoretical claims, and refine the presentation of text and figures. The authors take full responsibility for the final manuscript, including its claims, citations, and figures.

REFERENCES

Valentin Abadie, Clemens Hutter, and Helmut Bölcskei. Recurrent neural networks approximate continuous functions. *arXiv preprint arXiv:2606.20325*, 2026. URL https://arxiv.org/abs/2606.20325.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://arxiv.org/abs/1909.01377.

Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Multiscale deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://arxiv.org/abs/2006.08656.

Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021. URL https://arxiv.org/abs/2107.05407.

Andrew R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993. doi: 10.1109/18.256500.

Peter L. Bartlett, Nick Harvey, Christopher Liaw, and Abbas Mehrabian. Nearly-tight VC-dimension and pseudodimension bounds for piecewise linear neural networks. *Journal of Machine Learning Research*, 20(63):1–17, 2019. URL https://jmlr.org/papers/v20/17-612.html.

Kenneth E. Batcher. Sorting networks and their applications. In *Proceedings of the Spring Joint Computer Conference*, volume 32 of *AFIPS Conference Proceedings*, pp. 307–314, 1968. doi: 10.1145/1468075.1468121. URL https://www.cs.kent.edu/~batcher/sort.pdf.

Olivier Bournez, Johanne Cohen, and Adrian Wurm. A universal uniform approximation theorem for neural networks. In *50th International Symposium on Mathematical Foundations of Computer Science*, volume 345 of *Leibniz International Proceedings in Informatics*, pp. 29:1–29:20, 2025. doi: 10.4230/LIPIcs.MFCS.2025.29. URL https://doi.org/10.4230/LIPIcs.MFCS.2025.29.

Yongqiang Cai. Vocabulary for universal approximation: A linguistic perspective of mapping compositions. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5189–5208. PMLR, 2024. URL https://proceedings.mlr.press/v235/cai24a.html.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://arxiv.org/abs/1806.07366.

Róbert Csordás, Kazuki Irie, Jürgen Schmidhuber, Christopher Potts, and Christopher D. Manning. MoEUT: Mixture-of-experts universal transformers. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/321387ba926b8e58d3591c0aeb52ffc2-Abstract-Conference.html.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989. doi: 10.1007/BF02551274.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1807.03819.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. doi: 10.1207/S15516709COG1402_1.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1380. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/3b01972cf31e6fa0fe29e4b8b5c2a0a1-Abstract-Conference.html.

Borjan Geshkovski, Cyril Letrouit, Yury Polyanskiy, and Philippe Rigollet. A mathematical perspective on Transformers. *Bulletin of the American Mathematical Society*, 62(3):427–479, 2025. doi: 10.1090/bull/1863. URL https://arxiv.org/abs/2312.10794.

Angeliki Giannou, Shashank Rajput, Jy-Yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped Transformers as programmable computers. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11398–11442, 2023. URL https://proceedings.mlr.press/v202/giannou23a.html.

Paul W. Goldberg and Mark R. Jerrum. Bounding the vapnik–chervonenkis dimension of concept classes parameterized by real numbers. *Machine Learning*, 18(2–3):131–148, 1995. doi: 10.1007/BF00993408. URL https://www.cs.ox.ac.uk/people/paul.goldberg/papers/GJ-ML95.pdf.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016. URL https://arxiv.org/abs/1603.08983.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 399–406, 2010. URL https://icml.cc/2010/papers/449.pdf.

Ingo Gühring and Mones Raslan. Approximation rates for neural networks with encodable weights in smoothness spaces. *Neural Networks*, 134:107–130, 2021. doi: 10.1016/j.neunet.2020.11.010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016. URL https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. doi: 10.1016/0893-6080(91)90009-T.

Geonho Hwang, Wonyeol Lee, Yeachan Park, Sejun Park, and Feras Saad. Floating-point neural networks are provably robust universal approximators. In *Computer Aided Verification*, volume 15932 of *Lecture Notes in Computer Science*, pp. 301–326, 2025. doi: 10.1007/978-3-031-98679-6_14. URL https://arxiv.org/abs/2506.16065.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. LoopFormer: Elastic-depth looped Transformers for latent reasoning via shortcut modulation. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RzYXb5YWBs.

Yuling Jiao, Yang Wang, and Bokai Yan. Approximation bounds for recurrent neural networks with application to regression. *arXiv preprint arXiv:2409.05577*, 2024. URL https://arxiv.org/abs/2409.05577.

Pascal Koiran and Eduardo D. Sontag. Vapnik–Chervonenkis dimension of recurrent neural networks. *Discrete Applied Mathematics*, 86(1):63–79, 1998. doi: 10.1016/S0166-218X(98)00014-6.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1909.11942.

Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993. doi: 10.1016/S0893-6080(05)80131-5.

Jialin Liu, Lisang Ding, Stanley Osher, and Wotao Yin. Expressive power of implicit models: Rich equilibria and test-time scaling. *arXiv preprint arXiv:2510.03638*, 2026. URL https://arxiv.org/abs/2510.03638v4. Version 4.

Jianfeng Lu, Zuowei Shen, Haizhao Yang, and Shijun Zhang. Deep network approximation for smooth functions. *SIAM Journal on Mathematical Analysis*, 53(5):5465–5506, 2021. doi: 10.1137/20M134695X. URL https://arxiv.org/abs/2001.03040.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision Transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023. doi: 10.1162/tacl_a_00562. URL https://aclanthology.org/2023.tacl-1.31/.

John Miller and Moritz Hardt. Stable recurrent models. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1805.10369.

Vishal Monga, Yuelong Li, and Yonina C. Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2):18–44, 2021. doi: 10.1109/MSP.2020.3016905. URL https://arxiv.org/abs/1912.10557.

Weigutian Ou, Philipp Schenkel, and Helmut Bölcskei. Three quantization regimes for ReLU networks. *arXiv preprint arXiv:2405.01952*, 2024. URL https://arxiv.org/abs/2405.01952v1. Version 1.

Yeachan Park, Geonho Hwang, Wonyeol Lee, and Sejun Park. Expressive power of ReLU and step networks under floating-point operations. *Neural Networks*, 175:106297, 2024. doi: 10.1016/j.neunet.2024.106297. URL https://arxiv.org/abs/2401.15121.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is Turing-complete. *Journal of Machine Learning Research*, 22(75):1–35, 2021. URL https://jmlr.org/papers/v22/20-302.html.

Aleksandar Petrov, Tom A. Lamb, Alasdair Paren, Philip H. S. Torr, and Adel Bibi. Universal in-context approximation by prompting fully recurrent models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/848784373188ddf641079524e89e0ac9-Abstract-Conference.html.

Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Lukas Gruber, Markus Holzleitner, Thomas Adler, David Kreil, Michael K. Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield networks is all you need. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=tL89RnzIiCd.

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

Anton Maximilian Schäfer and Hans-Georg Zimmermann. Recurrent neural networks are universal approximators. *International Journal of Neural Systems*, 17(4):253–263, 2007. doi: 10.1142/S0129065707001111.

Kristian Schwethelm, Daniel Rueckert, and Georgios Kaissis. How much is one recurrence worth? iso-depth scaling laws for looped language models. *arXiv preprint arXiv:2604.21106*, 2026. URL https://arxiv.org/abs/2604.21106v1. Version 1.

Zuowei Shen, Haizhao Yang, and Shijun Zhang. Deep network approximation in terms of intrinsic parameters. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 19909–19934. PMLR, 2022. URL https://proceedings.mlr.press/v162/shen22g.html.

Jonathan W. Siegel. Optimal approximation rates for deep ReLU neural networks on Sobolev and Besov spaces. *Journal of Machine Learning Research*, 24(357):1–52, 2023. URL https://jmlr.org/papers/v24/23-0025.html.

Ayhan Suleymanzade, Chanhyuk Lee, Floor Eijkelboom, Nicholas M. Boffi, İsmail İlkan Ceylan, and Jinwoo Kim. Thinking with looped flows. *arXiv preprint arXiv:2609.11801*, 2026. URL https://arxiv.org/abs/2609.11801v1. Version 1.

Ezra Winston and J. Zico Kolter. Monotone operator equilibrium networks. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://arxiv.org/abs/2006.08591.

Kevin Xu and Issei Sato. On expressive power of looped transformers: Theoretical analysis and enhancement via timestep encoding. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 69613–69646, 2025. URL https://proceedings.mlr.press/v267/xu25x.html.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2311.12424.

Dmitry Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94: 103–114, 2017. doi: 10.1016/j.neunet.2017.07.002. URL https://arxiv.org/abs/1610.01145.

Dmitry Yarotsky. Optimal approximation of continuous functions by very deep ReLU networks. In *Proceedings of the 31st Conference on Learning Theory*, volume 75 of *Proceedings of Machine Learning Research*, pp. 639–649. PMLR, 2018. URL https://proceedings.mlr.press/v75/yarotsky18a.html.

Dmitry Yarotsky and Anton Zhevnerchuk. The phase diagram of approximation rates for deep neural networks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 13005–13015. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/979a3f14bae523dc5101c52120c535e9-Paper.pdf.

Shijun Zhang. Sharp approximation rates for neural networks with affine latent parameterizations. *arXiv preprint arXiv:2608.31157*, 2026. URL https://arxiv.org/abs/2608.31157v1. Version 1.

Shijun Zhang, Jianfeng Lu, and Hongkai Zhao. On enhancing expressive power via compositions of single fixed-size ReLU network. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 41452–41487. PMLR, 2023. URL https://proceedings.mlr.press/v202/zhang23ad.html.

APPENDIX CONTENTS

A Approximation background and comparison with prior work . . . 14
A.1 Classical universal approximation . . . 14
A.2 What an optimal rate means . . . 14
A.3 From approximation rates to shared computation . . . 15
A.4 Dense parameters and repeated calls . . . 15
A.5 Affine latent budgets and dimension-dependent constants . . . 15
B Approximation bounds and constructions . . . 16
B.1 Spatial encoding and uniform accuracy . . . 17
B.2 Logarithmic-depth modules . . . 21
B.3 Completing the stage and call budgets . . . 26
B.4 Capacity and the approximation lower bound . . . 27
B.5 Constant-depth modules for finite L^p error . . . 28
B.6 Uniform approximation of Hölder functions . . . 30
C Proofs for prescribed call sequences . . . 31
C.1 Loading once and decoding after later visits . . . 32
C.2 Capacity along a fixed call sequence . . . 33
C.3 Frontier retention and the same-count comparison . . . 34
C.4 Consequences for contiguous-stage allocations . . . 36
D Finite-description bounds . . . 38
D.1 A total description budget . . . 38
D.2 Componentwise description budgets . . . 39

A APPROXIMATION BACKGROUND AND COMPARISON WITH PRIOR WORK

This section supplies background for Section 2.2: what universal approximation guarantees, how a rate strengthens that guarantee, and which resource choices matter when comparing rates.

A.1 CLASSICAL UNIVERSAL APPROXIMATION

Consider a single-hidden-layer network with activation function σ:

$$g_N(x) = \sum_{i=1}^{N} a_i \sigma(w_i^\top x + b_i), \qquad a_i, b_i \in \mathbb{R}, \quad w_i \in \mathbb{R}^d.$$

For any continuous sigmoidal activation, these networks are dense in $\mathcal{C}_d$ under the uniform norm: for every $f \in \mathcal{C}_d$ and $\varepsilon > 0$, some finite width N and weight assignment give $\|f - g_N\|_\infty < \varepsilon$ (Cybenko, 1989, Theorem 2). The width is chosen for each target and requested accuracy.

Uniform density on compact sets also holds for continuous, bounded, nonconstant activations; related results establish density in finite $L^p(\mu)$ norms for finite input measures (Hornik, 1991, Theorems 1–2). For continuous activations with biases, nonpolynomiality is the necessary and sufficient condition for this single-hidden-layer uniform density property. This includes ReLU, which is continuous and is not a polynomial on $\mathbb{R}$ (Leshno et al., 1993, Theorem 1). These statements concern the existence of an approximant. They do not provide a sharp relation between accuracy and network size, or guarantee that a particular training method finds the required weights.

A.2 WHAT AN OPTIMAL RATE MEANS

A quantitative comparison fixes a bounded target class, an error norm and a network family with a specified resource budget. For a parameter budget P, write $E(P)$ for the best worst-case error in that model. A sharp polynomial rate has the form

$$cP^{-\gamma} \le E(P) \le CP^{-\gamma}, \qquad \gamma > 0,$$

where $c, C > 0$ do not depend on P. The upper bound supplies approximants for every target; the lower bound prevents any admissible design from improving the worst-case order. The constants

depend only on the fixed dimension and function class. Matching polynomial exponents with an unresolved logarithmic factor is weaker than this two-sided comparison.

Optimality therefore refers to approximation order within the specified model. It does not identify an exact finite-budget minimizer, require every target to be equally difficult, or give an optimization algorithm. In this paper, the analogous budget is (J, T) and the design is chosen before the target, as specified in (2). Theorem 1 gives matching bounds while permitting the call allocation to be chosen; Theorem 2 asks what remains possible when the call sequence is prescribed.

A.3 From approximation rates to shared computation

Early quantitative bounds already showed that the target class matters. For functions with a bounded first Fourier moment, a network with N sigmoidal units achieves integrated squared error $O(N^{-1})$, hence L^2 error $O(N^{-1/2})$. The constant depends on the Fourier bound and domain, and the result is not a uniform-error guarantee for the whole Lipschitz ball (Barron, 1993, Theorem 1).

For smoothness classes, constructive ReLU approximation relates accuracy to both depth and parameter count. On the unit ball of $W^{s,\infty}([0,1]^d)$, for integer $s \geq 1$, uniform accuracy $\varepsilon \in (0, 1/2)$ is attainable with depth $O_{d,s}(\log(1/\varepsilon))$ and $O_{d,s}(\varepsilon^{-d/s}\log(1/\varepsilon))$ weights and computation units in the source's computation-graph model (Yarotsky, 2017, Theorem 1). Allowing much greater depth and discontinuous target-to-weight assignments leads to the sharp $P^{-2/d}$ uniform rate for bounded Lipschitz targets. Continuous weight selection instead limits the exponent to $1/d$; thus parameter count alone does not describe every approximation model (Yarotsky, 2018, Theorems 1–2).

Fixed-module composition and affine-latent approximation make reuse an explicit resource choice (Zhang et al., 2023; Zhang, 2026). Our model separates stored modules from their calls, as discussed in Section 2.2.

We next detail the resource counts and assumptions behind the comparisons in Table 1. The entries retain their original resource variables; matching rate exponents alone do not imply that one theorem covers another. Our model's resource convention is given in Section 3, and (18) gives the complete costs of our construction.

A.4 Dense parameters and repeated calls

A dense network with input size a, output size b, and $h \geq 1$ hidden layers of width w has

$$w(a+1) + (h-1)w(w+1) + b(w+1)$$

weights and biases, including zero entries. Its scalar operation count has the same order. Thus each added hidden layer in the width-$2d+10$ untied construction costs order d^2 parameters. This gives the table's $h = O(P/d^2)$ conversion when the budget supports the architecture; P is the source's total weight budget, denoted W there (Yarotsky, 2018, Theorem 2(b)).

For the uniform repeated-composition construction, the state dimension is $s_\infty = 3^d(5d+4) - 1$. Together with the width and depth in Table 1, this gives a sufficient dense module parameter count $q_\infty = O(d^3 16^d)$. Including the affine input and output maps preserves this storage order. The $n_\infty = 3r + 2d - 1$ calls execute $O(n_\infty q_\infty)$ scalar operations, and their unrolling has at most $n_\infty(2d+3)$ hidden layers. Repetition reuses the same stored module; the dimension-dependent module cost remains part of the comparison (Zhang et al., 2023).

A.5 Affine latent budgets and dimension-dependent constants

The affine latent model fixes the architecture and the generator $\theta_f = A\xi_f + a$ before choosing the target. Its budgets bound the latent dimension by M and the deployed network's dense parameter count P_Φ by P. A dense representation of (A, a) has $P_\Phi M + P_\Phi$ entries, outside the budget M. Evaluating this dense generator takes $O(P_\Phi M)$ operations once per materialized network; inference then takes $O(P_\Phi)$ operations per input. These costs describe the dense representation and need not apply to a structured or implicit generator (Zhang, 2026, Section 2.4).

The width and depth symbols in Table 1 are defined by

$$N_d = 3^d(\max\{20d, 96\} + 4),$$
$$B_d = (48 + 2d)N_d(N_d + 1) + N_d(d + 2) + 1,$$
$$L_{d,P} = 18\lfloor P/B_d \rfloor + 30 + 2d.$$

The sufficient upper-bound construction assumes $\min\{M, P\} \geq 4$ and $P \geq P_d^\star = 2^{2d+2}B_d$. For unit-Lipschitz targets its error bound is

$$(d + 2)C_d[P \min\{M, P\}]^{-1/d}, \qquad C_d = 4\,8^{1/d}\sqrt{d}\,B_d^{2/d}.$$

These dimension-dependent quantities specify a sufficient construction, not necessary costs for all affine latent networks (Zhang, 2026, Theorem 3.1 and Eqs. (7)–(10)).

B Approximation bounds and constructions

We prove Theorem 1 in four steps. Spatial encoding and shifted grids give a uniform approximation at a chosen resolution. We then realize the decoder with logarithmic-depth modules, fit its grid and stage counts to the budgets, and establish the matching capacity lower bound. Two supplementary results then give the constant-depth construction for finite L^p error and the uniform Hölder extension (Proposition 11 and Corollary 12). Figure 4 summarizes the proof dependencies, including the results for prescribed sequences and finite descriptions. Figure 5 illustrates the encoding and repeated decoding on one grid.

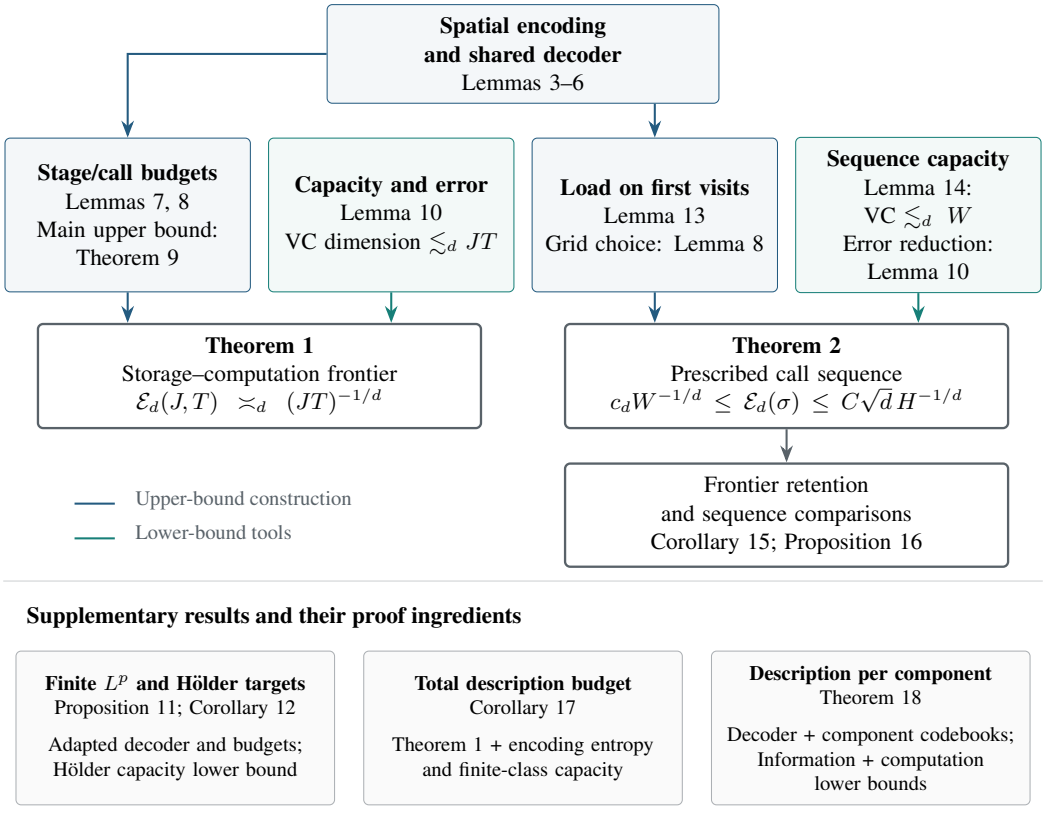


Figure 4: Proof roadmap. Arrows show the principal dependencies for the two main theorems, with supporting lemmas grouped. The lower boxes list the constructions and additional tools used for the supplementary results.

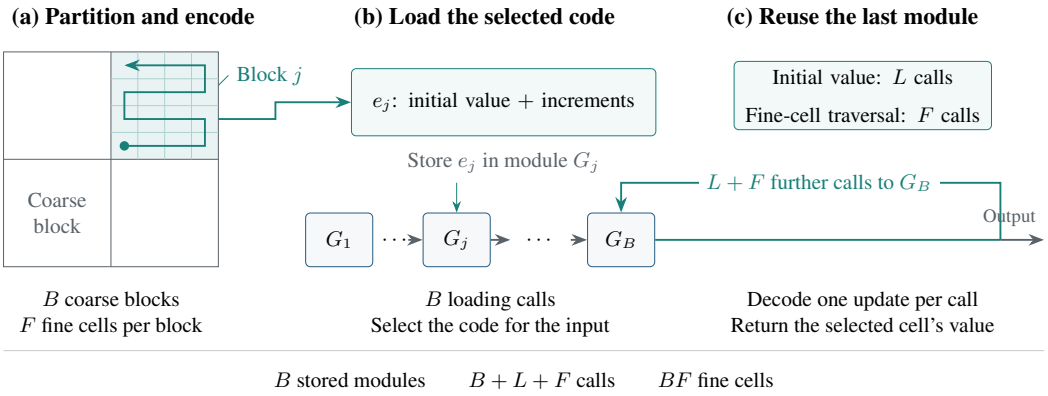


Figure 5: Spatial encoding and looped decoding on one grid. Each of the B stored modules carries a block code. After B loading calls select the input's code, the last module uses L calls to recover the initial value and F calls to traverse the fine cells. Shifted copies and input/output maps are omitted.

B.1 Spatial encoding and uniform accuracy

The upper bound rests on a spatial decoder that loads a block code and then reads its cell values. We first establish correctness of a decoder on one grid, then combine shifted grids to obtain uniform accuracy. Section B.2 realizes these shifted copies with shared spatial features and the module dimensions in the main theorem.

B.1.1 Encoding a block of grid values

A reflected-coordinate snake visits every cell once, moving to a neighboring cell at each step. Quantized Lipschitz values along this path have increments in $\{-1, 0, 1\}$. The first value and these increments can therefore be encoded as a real number whose digits are read by a fixed extraction rule. Encoding neighboring quantized values by finite-alphabet increments and recovering them sequentially follows established deep ReLU constructions (Yarotsky, 2018; Yarotsky & Zhevnerchuk, 2020). Fixed-module iteration can implement the digit extraction itself (Zhang et al., 2023, Proposition 3.2 and Appendix D). The lemma below gives the version used here: it loads a code from one of several stored modules, traverses the selected block, and maintains a bounded state even in transition regions.

Lemma 3 (Fixed-cost sequential spatial decoder)**.** *Let $n, m \geq 1$ be integers, $B = n^d$, $F = m^d$, $M = nm$, and $L = \lceil \log_3(2M+1) \rceil$. In every coarse block, order the fine cells by the same canonical reflected-coordinate snake: start at the zero counter with all directions positive, take coordinate zero as the fastest axis, and use the successor in Appendix B.1.3. For each block j, let $q_{j,0}, \ldots, q_{j,F-1}$ be integers in this order satisfying*

$$-M \leq q_{j,k} \leq M, \qquad q_{j,k} - q_{j,k-1} \in \{-1, 0, 1\} \quad (k \geq 1).$$

Encode $q_{j,0} + M$ in exactly L ternary digits, followed by the $F - 1$ digits $q_{j,k} - q_{j,k-1} + 1$, and let

$$e_j = \sum_{s=1}^{L+F-1} d_{j,s} 4^{-s}, \qquad d_{j,s} \in \{0, 1, 2\}.$$

We call e_j the block code; its base-four digits encode the initial value and increments. For every $0 < \tau < (2M)^{-1}$, there is a recurrent ReLU architecture chosen independently of the target with B stage modules, state dimension $O_d(1)$, and a sparse computation graph of size $O_d(1)$ per call, counting its affine coefficients and ReLU operations. It uses exactly $B + L + F$ calls and has the following properties. Modules $1, \ldots, B - 1$ are called once and the last module is called $1 + L + F$ times. Outside the width-τ transition set of the global M-grid, the output is exactly $q_{j,k}/M$ for the unique fine cell (j, k) containing the input. The output lies in $[-1, 1]$ everywhere, and the transition set has Lebesgue measure at most $C_d M \tau$. The topology and schedule are independent of the digits; only the B affine coefficients e_j are target dependent.

The transition set consists of strips near grid faces. The decoder is exact outside these strips and bounded everywhere. We first give the selector and digit-extraction primitives, then the complete recurrent transition, and finally verify the lemma. In the formulas below, a superscript $+$ denotes a state variable's value after the next call; update right-hand sides use the current state.

B.1.2 Selectors and digit extraction

Write $\rho(u) = \max\{u, 0\}$ and define the plateau function

$$\eta_{[a,b],\tau}(u) := \tau^{-1}\{\rho(u-a) - \rho(u-a-\tau) - \rho(u-b+\tau) + \rho(u-b)\}.$$

It takes values in $[0,1]$, is one on $[a+\tau, b-\tau]$, and is zero outside $[a,b]$. Taking the minimum of the d coordinate plateaux gives a box selector; the minimum is a fixed $O(d)$ ReLU circuit because $\min\{u,v\} = u - \rho(u-v)$. Selectors for adjacent half-open grid cells have disjoint interiors. Away from the grid transition set, exactly one relevant selector is one and all others are zero. Including the exterior faces, a union bound over the $O_d(M)$ coordinate hyperplanes gives transition-set measure at most $C_d M\tau$.

For digit extraction, set

$$\psi_{\mathrm{dig}}(w) = 3\rho(w-2/3) - 3\rho(w-1) + 3\rho(w-5/3) - 3\rho(w-2), \qquad D_4(z) = \psi_{\mathrm{dig}}(4z).$$

The four-ReLU map D_4 returns the leading digit of every legal base-four code with digits in $\{0,1,2\}$. A direct check on its affine pieces gives

$$0 \le z \le 2/3 \quad \Longrightarrow \quad 0 \le 4z - D_4(z) \le 2/3. \tag{13}$$

Thus $z^+ = 4z - D_4(z)$ shifts a legal code and keeps every bounded intermediate code in the prescribed interval.

B.1.3 Complete recurrent realization of the spatial decoder

This appendix supplies one affine–ReLU transition for all phases of Lemma 3. The prescribed schedule calls modules $1, \dots, B$ once, then module B another $L + F$ times. Let $\zeta_j = \mathbf{1}_{\{j=B\}}$ be a target-independent coefficient in module j. Its only target-dependent coefficient is e_j. All formulas below are evaluated at every call; binary ReLU gates select the updates.

State and gates. The $4d + 10$ state variables are the unchanged input $x \in [0,1]^d$, the code $z \in [0, 2/3]$, origin $o \in [0,1]^d$, initial value $a \in [0,6]$, value and output $q, y \in [-1,1]$, cell counters $c \in \{0, \dots, m-1\}^d$, direction bits $b \in \{0,1\}^d$, initial-value and fine-cell counters $N_A \in \{0, \dots, L\}$ and $N_F \in \{0, \dots, F\}$, and one-hot phase flags (p_L, p_A, p_F, p_D). Initially all state variables except x, the direction bits, and p_L are zero; $b_i = p_L = 1$. For $R > 0$, use

$$G_R(s,v) = \rho(v) - \rho(v - Rs) - \rho(-v) + \rho(-v - Rs).$$

When $s \in \{0,1\}$ and $|v| \le R$, this equals sv. Also set

$$\mathrm{And}(s,t) = \rho(s+t-1), \qquad E_k(v) = \rho(v-k+1) - 2\rho(v-k) + \rho(v-k-1).$$

The first is Boolean conjunction on bits, and the second is the indicator $v = k$ on integer inputs. Clipping is implemented by $\mathrm{clip}_{[a,b]}(v) = a + \rho(v-a) - \rho(v-b)$.

One-hot phase transitions. Using the old state, compute

$$e_A = \mathrm{And}(p_A, E_{L-1}(N_A)), \qquad e_F = \mathrm{And}(p_F, E_{F-1}(N_F)),$$
$$\iota_0 = E_0(N_F), \qquad s_F = \mathrm{And}(p_F, 1 - \iota_0).$$

The counter and phase updates are

$$N_A^+ = N_A + p_A, \qquad N_F^+ = N_F + p_F,$$
$$p_L^+ = (1-\zeta_j)p_L, \qquad p_A^+ = p_A - e_A + \zeta_j p_L,$$
$$p_F^+ = p_F - e_F + e_A, \qquad p_D^+ = p_D + e_F.$$

Induction along the prescribed schedule gives exactly B loading calls, L calls to decode the initial value, and F fine calls. The flags and counters are input- and target-independent, so this induction also holds on all spatial transition regions. The done state is stationary.

Loading and decoding. Let $\theta_j(x)$ be the coarse selector and v_j its fixed origin. For $B = 1$, replace the selector by one and the origin by zero using a fixed binary coefficient; the same padded circuit is used. Compute bounded candidate values

$$z_L = \mathrm{clip}_{[0,2/3]}(z + \theta_j(x)e_j), \qquad o_L = \mathrm{clip}_{[0,1]^d}(o + \theta_j(x)v_j),$$

and, with $r = D_4(z)$ from Equation (13), put

$$z_S = 4z - r, \qquad a_A = \mathrm{clip}_{[0,6]}(3a + r/M).$$

The actual code, origin, and initial value updates are

$$\begin{aligned} z^+ &= z + G_{2/3}(p_L, z_L - z) + G_{2/3}(p_A, z_S - z) + G_{2/3}(s_F, z_S - z), \\ o_i^+ &= o_i + G_1(p_L, o_{L,i} - o_i), \\ a^+ &= a + G_6(p_A, a_A - a). \end{aligned}$$

The safe-shift identity ensures all code differences fit their gate radius. A fine call shifts the code only after the first fine call, because that first call uses the already decoded initial value. The normalized current value is formed by

$$v_A = \mathrm{clip}_{[-1,1]}(a - 1), \qquad v_\Delta = \mathrm{clip}_{[-1,1]}(q + (r-1)/M), \qquad v = v_\Delta + G_2(\iota_0, v_A - v_\Delta).$$

For the current fine-cell counter c, define

$$\vartheta_c(x, o) = \min_{0 \le i < d} \eta_{[c_i/M, (c_i+1)/M], \tau}(x_i - o_i). \tag{14}$$

Then

$$y_F = \mathrm{clip}_{[-1,1]}\big(y + G_2(\vartheta_c(x,o), v - y)\big), \qquad q^+ = q + G_2(p_F, v - q), \quad y^+ = y + G_2(p_F, y_F - y).$$

Thus the current decoded value is stored during the same call. No product of two state variables has been used: products with M^{-1}, fixed origins, or e_j are affine coefficients, while Boolean and bounded selections use the displayed ReLU gates.

Reflected-coordinate successor. Coordinate zero is the fastest axis. Set $\kappa_0 = 1$. For $i = 0, \dots, d-1$, form

$$\begin{aligned} l_i &= \rho(1 - c_i) - \rho(-c_i), & h_i &= \rho(c_i - m + 2) - \rho(c_i - m + 1), \\ t_i &= \mathrm{And}(b_i, h_i) + \mathrm{And}(1 - b_i, l_i), & f_i &= \mathrm{And}(\kappa_i, t_i), \\ v_i &= \mathrm{And}(\kappa_i, 1 - t_i), & \kappa_{i+1} &= f_i, \\ \Delta c_i &= \mathrm{And}(v_i, b_i) - \mathrm{And}(v_i, 1 - b_i), & \Delta b_i &= f_i - 2\,\mathrm{And}(b_i, f_i). \end{aligned}$$

Here l_i, h_i are endpoint bits, t_i indicates that the current direction is blocked, f_i flips that direction and carries to the next coordinate, and v_i permits a move. Write $c_i^+ = c_i + G_1(p_F, \Delta c_i)$ and $b_i^+ = b_i + G_1(p_F, \Delta b_i)$. The selector above reads c, before this update. Induction on the number of coordinates shows that successive slices traverse the lower-dimensional path in alternating directions. Thus all m^d cells are visited once, successive cells share a face, and the terminal carry makes no out-of-grid move. For $m = 1$, both endpoint bits are one; each coordinate flips without moving, which covers the singleton case.

State bounds and fixed cost. One-hot gating, the safe shift, and clipping preserve the displayed state bounds, including where selectors take intermediate values. The loading and decoding induction is given in Section B.1.4.

The direct circuit uses $35d + 56$ ReLU evaluations and $O_d(1)$ affine operations per call. These counts do not depend on n, m, B, L, F, the code, or the phase. Target-independent coefficients such as τ^{-1} and M^{-1} and the parameter entries for them are counted in the common resource bounds. To use a budget-independent compact state cube, conjugate the transition by the diagonal normalization $a \mapsto a/6$, $N_A \mapsto N_A/L$, $N_F \mapsto N_F/F$, $c_i \mapsto c_i/\max\{1, m-1\}$, together with affine normalization of z, q, y. Coordinatewise clipping of the normalized output makes every module a map of the full cube into itself and is inactive on reachable states. This adds only $O_d(1)$ operations and parameters; the explicit $35d + 56$ count refers to the unnormalized transition before this affine normalization and coordinatewise clipping.

B.1.4 CORRECTNESS OF THE SEQUENTIAL SPATIAL DECODER

The selectors and digit reader are given in Section B.1.2, and the complete state transition is given in Section B.1.3. It remains to verify the loading and decoding invariants and the stated resource bounds.

Proof of Lemma 3. Loading the coarse code. The phase flags in Section B.1.3 select exactly B loading calls, L initial-value calls, and F fine calls. For an input outside the transition set, exactly one coarse selector is one. Thus after loading, $z = e_j$ and $o = v_j$ for the matching block j. For $B = 1$, the constant selector loads its code and the origin is zero.

Decoding and storing. After the next L calls, initial-value induction gives $a = (q_{j,0} + M)/M$, and z contains precisely the remaining increment digits. The current-value state q is still zero; the first fine call sets it to $a - 1 = q_{j,0}/M$ without consuming an increment digit. The counter c and loaded origin o determine the fine-cell selector in (14). Before each later fine call k, where $1 \le k < F$, the counter points to cell k, $q = q_{j,k-1}/M$, and the leading unread digit is $q_{j,k} - q_{j,k-1} + 1$. The update reads that digit, shifts the code, and sets the current value to $q_{j,k}/M$. On every fine call, including the first, the output is assigned this value exactly when the input matches the current cell; otherwise it is unchanged. Only then is the counter advanced. The reflected-coordinate successor in Section B.1.3 uses $O(d)$ gates and visits all m^d cells once, in alternating reflected slices. Induction over the calls therefore gives the value of the unique matching fine cell.

Bounds and resource counts. The one-hot gates, safe digit shift (13), and candidate clips preserve the state bounds on every input, including the transition strips. On a legal selected code, clipping leaves every active decoding update unchanged, so it does not alter the preceding induction. In particular, $y \in [-1, 1]$ everywhere. The selector construction in Section B.1.2 gives transition-set measure at most $C_d M\tau$.

All phase flags, counters, clips, minima, digit shifts, and gates use a fixed number of state coordinates and $O_d(1)$ sparse affine–ReLU operations per call. Their fixed coefficients depend only on d, n, m, τ and the stage index, while the code coefficient appears only in its stage's affine loading branch. The transition in Section B.1.3 supplies the stated state, call, and target-dependence counts. □

B.1.5 UNIFORM ACCURACY FROM SHIFTED GRIDS

For the Lipschitz class, a strict majority of $2d + 1$ shifted copies will be accurate at every input. Their median gives the required uniform error. The use of shifts and medians to remove grid transition regions is an existing approximation tool; a coordinatewise median-of-three version is given in earlier work (Lu et al., 2021, Lemma 3.4). The following lemma instead arranges $2d + 1$ simultaneous shifts. This form allows all copies to use the same coordinate features in our recurrent implementation.

Lemma 4 (Shifted grids for Lipschitz approximation)**.** *Let $n, m \ge 1$ be integers, $M = nm \ge 2$ and $Q = 2d + 1$. Write $\mathbf{1}$ for the all-ones vector in $\mathbb{R}^d$. Put*

$$\delta_r = \frac{r - d}{QM}, \qquad u_r(x) = \frac{x + \mathbf{1}}{3} + \delta_r \mathbf{1}, \qquad \tau = \frac{1}{4QM}, \qquad 0 \le r < Q.$$

For $f \in \mathcal{F}_d^{\mathrm{Lip}}$, define

$$g_r(v) = \tfrac{1}{3} f\Big(\mathrm{clip}_{[0,1]^d}(3v - \mathbf{1} - 3\delta_r \mathbf{1})\Big).$$

Quantizing g_r at the fine-cell centers by $q = \lfloor M g_r \rfloor$ gives valid block codes for Lemma 3. If $y_r(x)$ equals the decoded value q/M whenever $u_r(x)$ lies outside the width-τ grid transition set, then

$$\|f - 3\,\mathrm{med}(y_0, \dots, y_{Q-1})\|_\infty \le \frac{3(1 + \sqrt{d}/2)}{M}. \tag{15}$$

No restriction is imposed on the remaining copies' real-valued outputs.

Proof. Each g_r is bounded by one and is unit-Lipschitz. Adjacent cell centers are distance $1/M$ apart, so the quantized integers belong to $[-M, M]$ and their successive differences lie in $\{-1, 0, 1\}$.

The shifted inputs lie inside the cube, at distance greater than τ from its exterior faces: $1/3 - d/(QM) > 1/12$ whereas $\tau \le 1/24$. For each coordinate, the residues of $M u_{r,i}(x)$ modulo one are

spaced by $1/Q$. A transition interval has circular length $2M\tau = 1/(2Q)$, so at most one shift is bad for that coordinate. Thus at most d copies are bad, and at least $d+1$ are good at every input.

For a good copy, the selected cell center is at distance at most $\sqrt{d}/(2M)$ from $u_r(x)$, and quantization costs at most $1/M$. Since $3g_r(u_r(x)) = f(x)$,

$$|3y_r(x) - f(x)| \le \frac{3(1+\sqrt{d}/2)}{M}.$$

An interval containing a strict majority of the Q values also contains their median, proving (15). □

B.2 Logarithmic-depth modules

The single-grid decoder establishes how to recover the encoded values. To combine shifted grids with the module dimensions in Theorem 1, we share their coordinate features, compute traversal carries in parallel, and clip the normalized state at the end of each call. These changes preserve the valid decoding trajectories; on the remaining inputs, clipping keeps the state bounded. Each call still completes one decoding update. Only codes and decoded values need separate registers for each shifted copy. An affine input map initializes the state, and the output network takes the median once. The budget conversion follows in Section B.3.

B.2.1 One decoding update in a fixed-size module

We first give the grid-level guarantee, then construct its transition. The proof builds shared coordinate features, uses them to update each copy, and finally counts the state, width and depth.

Write $C_{01}(t) = \rho(t) - \rho(t-1)$ for clipping to $[0,1]$ and, for $s \in [0,1]$, define the ReLU gate

$$G_R(s,v) = v - \rho(v - Rs) + \rho(-v - Rs).$$

This lies between zero and v. If $|v| \le R$, it equals sv when s is binary; also $G_R(0,v) = 0$ for every real v, without a size restriction. The identity $\rho(v) - \rho(-v) = v$ shows that this is the gate in Section B.1.3, written with two ReLU nodes. Put $Q = 2d+1$.

Lemma 5 (A complete decoding update in one call)**.** *For integers $d, n \ge 1$ and $m \in 4\mathbb{N}$, let $M = nm$, $B = n^d$, $F = m^d$ and $L = \lceil \log_3(2M+1) \rceil$. There is a dense module architecture chosen independently of the target with*

$$s_g = 13d + 10, \qquad w_g = 108d + 42, \qquad h_g = 8 + \lceil \log_2 d \rceil,$$

where every hidden layer has width w_g. Using B stored modules, an affine input map and a ReLU median output map, every $f \in \mathcal{F}_d^{\mathrm{Lip}}$ admits an assignment with $B + L + F$ module calls and

$$\|f - \Phi\|_\infty \le \frac{3(1+\sqrt{d}/2)}{M}. \tag{16}$$

The first $B-1$ modules are called once and the last is repeated. The budget argument in Section B.3.2 shows that restricting m to a multiple of four does not change the main rate. The state stays in its normalized cube, and further calls to the last module leave the decoder output unchanged.

Proof. Step 1. Set up shifted grids and block codes. Set

$$u = \frac{x + \mathbf{1}}{3}, \qquad \delta_r = \frac{r-d}{QM}, \quad 0 \le r < Q, \qquad \tau = \frac{1}{4QM}.$$

Use the quantized block values for g_r from Lemma 4. Each code contains the L ternary digits of $q_{\mathrm{first}} + M$ and the $F-1$ successive increments shifted into $\{0,1,2\}$. These digits are stored in base four, so every code belongs to $[0, 2/3]$ and the safe digit shift (13) applies. By Lemma 4, preserving the spatial decoder on good copies suffices for the median to give (16). We implement these updates while keeping the entire state bounded.

The state separates shared spatial information from the four registers needed by each shifted copy. The following table summarizes all coordinates; the shared features A, D are constructed in Step 2.

State variables	Role	Coordinates
u, A, D	Shared input coordinates and spatial features	$3d$
c, b	Shared cell counters and directions	$2d$
N_A, N_F	Shared decoding counters	2
p_L, p_A, p_F, p_D	Shared loading, initial-value, fine-cell, done phases	4
$(z_r, a_r, q_r, y_r)_{r=0}^{Q-1}$	Code, initial value, current value, output per copy	$4Q$
Total, with $Q = 2d + 1$		$13d + 10$

Each copy retains four registers: the unread code z_r, the initial-value accumulator a_r, the current value q_r, and the selected output y_r. They start at zero. All copies share the fine-cell counters, direction bits and four phase flags of Section B.1.3. We denote its two scalar counters by N_A, N_F, counting decoded initial-value digits and visited fine cells. Initially $N_A = N_F = 0$, all cell counters are zero, all directions are positive, and only the loading flag p_L is one. The phase transitions use the thresholds L, F and are independent of the copy registers.

Step 2. Build two shared features per coordinate. The copies need not store separate coarse-cell origins. Write $t_i = nu_i$ and $\mathrm{tri}(t) = 2\,\mathrm{dist}(t, \mathbb{Z})$. Retain two feature arrays $A, D \in \mathbb{R}^d$, initialized affinely by $A_i = 2t_i$ and $D_i = 1/2 - 2t_i$. When the loading stage has origin $o = \alpha/n$, let $R = n^{d-1}$ and add

$$\Delta A_i = \frac{-4\rho(t_i - \alpha_i - 1/2) + 4\rho(t_i - \alpha_i - 1)}{R},$$
$$\Delta D_i = \frac{4\rho(t_i - \alpha_i - 1/4) - 4\rho(t_i - \alpha_i - 3/4)}{R},$$

using the bounded gate $G_2(p_L, \cdot)$, where p_L is the loading flag. Each coordinate index occurs R times in the list of B origins. Summing these hinge differences gives, at the end of loading,

$$A_i = \mathrm{tri}(t_i), \qquad D_i = \mathrm{tri}(t_i - 1/4).$$

For example, the slope of the first expression changes from 2 to -2 at each half-integer and back to 2 at each integer. The second has its changes at quarter-integers. Their initial values fix the corresponding triangular functions. Throughout loading, A_i decreases from $2t_i$ to its final value and D_i increases from $1/2 - 2t_i$; the bounds $0 \le A_i \le 2n$ and $-2n \le D_i \le 1$ therefore hold. The features remain fixed during decoding.

Step 3. Recover the coordinates for each shifted copy. Taking a fractional part creates jumps at integers. We use two coordinate representations, with jumps at integers and half-integers respectively, and choose the one whose jump is farther from the current cell center. The shared triangular features recover this coordinate away from the jump; a mask suppresses the transition region. Put $\eta = n\tau = 1/(4Qm)$ and $\kappa = 1/(2\eta)$. For one coordinate, let $\xi = C_{01}(A)$. After loading, $\xi = \mathrm{tri}(t)$ and $D = \mathrm{tri}(t - 1/4)$. For a fine-cell counter $c \in \{0, \dots, m-1\}$, write $k = (c + 1/2)/m$, $h = 1/m$ and $\epsilon = n\delta_r$. We need a coordinate offset Δ and a mask V with two properties on these completed features and legal counters:

- If a good shifted copy matches the cell, so that $\mathrm{frac}(t + \epsilon) = k + w$ with $|w| < h/2 - \eta$, then $\Delta = w - \epsilon$ and $V = 1$.
- Whenever $V > 0$, the offset satisfies $\Delta \equiv t - k \pmod 1$.

The first property ensures that the matching cell is selected. The second will exclude every other cell. We construct the offset and mask as follows. Define two coordinate recovery functions

$$F_0(\xi, D) = 1 - \xi/2 - \rho(1 - \xi - \kappa\rho(D - 1/2)),$$
$$F_H(\xi, D) = (1 + \xi)/2 - \rho(\xi - \kappa\rho(1/2 - D)).$$

For $0 \le \xi \le 1$ and arbitrary D,

$$\xi/2 \le F_0 \le 1 - \xi/2, \qquad (1 - \xi)/2 \le F_H \le (1 + \xi)/2.$$

Let $s = \mathrm{frac}(t)$. For $s \in [\eta, 1/2]$, $D \le 1/2$ and $\xi = 2s$, so $F_0 = s$. For $s \in [1/2, 3/4]$, $D - 1/2 = 1 - \xi$ and $\kappa \ge 1$; for $s \in [3/4, 1 - \eta]$, $D - 1/2 = \xi$ and $\kappa\xi \ge 1$. The outer ReLU in F_0 therefore vanishes in both latter intervals, giving $F_0 = s$ whenever t is at distance at least η from an integer. Since $\mathrm{tri}(t+1/2) = 1-\xi$ and $\mathrm{tri}(t+1/4) = 1-D$, the identity $F_H(\xi, D) = F_0(1-\xi, 1-D)$ gives $F_H = \mathrm{frac}(t + 1/2)$ away from the half-integer jumps.

Since m is a multiple of four, the representation selectors have the simple form

$$U_c = C_{01}(c - 3m/4 + 1), \qquad q_c = C_{01}(c - m/4 + 1) - U_c.$$

They belong to $[0, 1]$ even at noninteger counters. At integer counters, the selection is

Cell counter	(q_c, U_c)	Recovered coordinate
$0 \le c < m/4$	$(0, 0)$	F_H
$m/4 \le c < 3m/4$	$(1, 0)$	F_0
$3m/4 \le c < m$	$(0, 1)$	F_H

For the half-shifted representation, subtracting $1/2$ and adding U_c selects the coordinate representative near the current cell center k: the lower quarter uses $F_H - 1/2$, and the upper quarter uses $F_H + 1/2$. Set

$$\begin{aligned} \Delta_0 &= F_0 - k, & \Delta_H &= F_H - k - 1/2 + U_c, \\ V_0 &= C_{01}((\xi - 2\eta)/(2\eta)), & V_H &= C_{01}((1 - \xi - 2\eta)/(2\eta)), \\ \Delta &= \Delta_H + G_2(q_c, \Delta_0 - \Delta_H), & V &= V_H + G_1(q_c, V_0 - V_H). \end{aligned}$$

The difference of the two coordinate offsets has absolute value at most $3/2$, and both masks belong to $[0, 1]$. Consequently these selections are bounded also before the features reach their triangular values.

The selected jump is at least $1/4 \ge 1/m$ from the chosen cell center. Thus one distance estimate covers every grid used here.

We now verify the two required properties. Since $|\epsilon| \le h/2 - 2\eta$, if a good shifted copy matches the current cell, write $\mathrm{frac}(t + \epsilon) = k + w$ with $|w| < h/2 - \eta$. The desired coordinate offset is $\Delta_* = w - \epsilon$, with $|\Delta_*| < h - 3\eta$. It remains on the selected coordinate branch, more than 3η from its jump. The formulas therefore give $\Delta = \Delta_*$ and $V = 1$. Conversely, $\xi = 2\,\mathrm{dist}(t, \mathbb{Z})$, so $V_0 > 0$ implies that t is more than η from every integer. Likewise $1 - \xi = 2\,\mathrm{dist}(t, \mathbb{Z} + 1/2)$, so $V_H > 0$ implies that t is more than η from every half-integer. On legal integer counters, q_c selects one of these exact representations; the integer U_c does not change its residue. Hence $V > 0$ implies $\Delta \equiv t - k \pmod 1$.

Step 4. Select cells and update the state variables. Five balanced min/max reductions give

$$\begin{aligned} l_c &= \max_i(o_i - u_i), & r_c &= \min_i(o_i + 1/n - u_i), \\ l_f &= -1/(2m) - \min_i \Delta_i, & r_f &= 1/(2m) - \max_i \Delta_i, & V_* &= \min_i V_i. \end{aligned}$$

The coarse selector for copy r is

$$\theta_{L,r} = \rho\big(\min\{1, (\delta_r - l_c)/\tau, (r_c - \delta_r)/\tau\}\big).$$

When $B = 1$, this same formula is identically one on actual inputs: $u_i \in [1/3, 2/3]$ and $|\delta_r| + \tau < 1/(2m) \le 1/8$ keep the point inside the coarse-cell plateau. The effective fine selector is

$$\theta_{F,r} = \rho\big(\min\{p_F, V_*, (n\delta_r - l_f)/\eta, (r_f - n\delta_r)/\eta\}\big).$$

The minimum of p_F and V_* is computed once and shared by all copies. These formulas satisfy $0 \le \theta_{F,r} \le p_F$. A good matching copy has $\theta_{F,r} = 1$. If the selector is positive, every coordinate mask is positive and $-h/2 < \Delta_i + n\delta_r < h/2$; the congruence just proved places the shifted point inside the current cell. Thus a good nonmatching copy has selector zero.

Each stage stores Q block codes w_r, its coarse-block origin and the last-loading-stage flag. Here are all four register updates, evaluated from the old state. Let $\omega_r = D_4(z_r)$ be the next digit, and set

$$\begin{aligned} \iota_0 &= E_0(N_F), & s_F &= \rho(p_F - \iota_0), \\ v_{\Delta,r} &= q_r + (\omega_r - 1)/M, & v_r &= v_{\Delta,r} + G_2(\iota_0, a_r - 1 - v_{\Delta,r}), \end{aligned}$$

where E_0 is the integer equality test in Section B.1.3. Thus v_r uses the decoded initial value on the first fine call and an increment on subsequent calls. The updates are

$$\begin{aligned} z_r^+ &= z_r + G_{2/3}(p_L, \theta_{L,r} w_r) + G_{2/3}(p_A, 3z_r - \omega_r) + G_{2/3}(s_F, 3z_r - \omega_r), \\ a_r^+ &= a_r + G_6(p_A, 2a_r + \omega_r/M), \\ q_r^+ &= q_r + G_2(p_F, v_r - q_r), \qquad y_r^+ = y_r + G_2(\theta_{F,r}, v_r - y_r). \end{aligned} \tag{17}$$

The product $\theta_{L,r} w_r$ is affine in the selector because w_r is a stored coefficient. The code shifts on each initial-value call and on every fine call after the first. No intermediate candidate is clipped; the complete state is clipped once at the end of the call.

The common reflected counter needs no serial chain of carries. To distinguish its blocked bits from the coordinates $t_i = nu_i$, write these bits as β_i. The carries, indexed as in Section B.1.3, are

$$\kappa_0 = 1, \qquad \kappa_{i+1} = \rho\left(\sum_{j=0}^{i} \beta_j - i\right), \qquad 0 \le i < d.$$

On the valid Boolean trajectory these are prefix conjunctions, so the flip and move flags can be written

$$f_i = \kappa_{i+1}, \qquad v_i = \kappa_i - \kappa_{i+1}.$$

Indeed, $\kappa_{i+1} = \kappa_i \beta_i$ on that trajectory, giving $v_i = \kappa_i(1 - \beta_i)$. This replaces the separate ReLU conjunction for v_i by an affine difference. The remaining counter increments and one-hot phase equations are unchanged. Equality with the original counter is needed only on its valid trajectory; final clipping bounds the full state map.

Step 5. Bound the state and preserve decoding accuracy. The state listed in Step 1 has $s_g = 13d + 10$ coordinates. Use raw intervals $[0, 2/3]$ for z_r, $[0, 6]$ for a_r, $[-1, 1]$ for q_r, y_r, $[0, L]$ for N_A and $[0, F]$ for N_F. Together with the feature bounds and counter ranges, these give a bounding box with positive side lengths; constant state variables can be assigned any positive scale. Normalize it affinely to $[0, 1]^{s_g}$ and apply the clipping map $C_{01}(v) = \rho(v) - \rho(v - 1)$ coordinatewise at the end of every call. This defines a map of the whole cube without requiring intermediate candidates to stay in their state intervals.

Clipping preserves the decoded values because good copies stay within these intervals. The common features, counters and phases update independently of all copy state variables, and each copy reads only its own state variables and this common state. On a good copy, loading selects exactly one valid code. Each prefix of the initial-value encoding divided by M lies in $[0, 2]$.

On the first fine call, the decoded initial value candidate lies in $[-1, 1]$; the unused increment candidate lies in $[-1/M, 1/M]$ because the current value is still zero. Thus the radius-two gate selects the initial value exactly. Each later fine call reads a valid increment and produces the next quantized value in $[-1, 1]$. All active writes therefore fit their gate radii and reproduce the original decoder. Inactive writes vanish for arbitrary candidate values, including after decoding ends. Inductively, the common state and every good copy already lie in their prescribed intervals, so final clipping leaves them unchanged. Other copies are bounded by clipping and cannot affect the good copies.

Step 6. Realize the dense module and count its width and depth. The following table bounds the ReLU nodes before final normalization and clipping, and the maximum number of successive ReLU evaluations needed for each group. Put $\ell = \lceil \log_2 d \rceil$. We count repeated hinges as separate nodes.

Operation	ReLU nodes (at most)	Maximum depth
Coordinate offsets and masks	$20d$	3
Five balanced reductions	$5(d-1)$	$3 + \ell$
Coarse and effective fine windows	$6Q + 1$	$6 + \ell$
First-fine and fine-shift flags	4	2
Code, initial value, value and output updates	$18Q$	$7 + \ell$
Shared features, counters and phases	$22d + 8$	5
Before final clipping	$95d + 32$	$7 + \ell$

The offset and mask circuit uses $18d$ hinges when $\xi = C_{01}(A)$ is shared; the table's $20d$ bound also allows recomputing ξ. Per copy, the 18 hinges split into four for digit reading, two for value selection, six for code writes, and two each for the initial-value, current-value and output writes. Shared terms use $8d$ feature hinges, $14d$ counter hinges (four endpoint, two blocked-bit, one prefix, two move, one direction and four gated-update hinges per coordinate), and eight phase-test hinges; the affine move flag costs none. Coordinate offsets and masks reach depth 3, followed by at most ℓ reduction layers. The minimum of the two fine-window terms and $\min\{p_F, V_*\}$ are computed in parallel; combining them and applying the final ReLU gives selector depth $6 + \ell$ and output-write depth $7 + \ell$. Other paths are shorter.

A layered realization carries the s_g normalized inputs and all previously computed nonnegative ReLU values. At most $N = 95d + 32$ such values are needed, so $s_g + N = 108d + 42$ channels suffice at every raw layer. Each new affine row reads the inputs and its predecessor values; identity rows carry the others. The final hidden layer computes the two hinges for each coordinate's normalized clipping, bringing the maximum hidden depth to $8 + \ell$. It also carries the original normalized input if needed. Its $3s_g$ channels fit the same width. The output affine map takes the clipping differences and adds no hidden layer. Thus the realization has no skip connections. Unused channels are padded by zeros, and every dense matrix entry and bias is counted. Identity maps on the cube fit the same topology.

The majority argument therefore proves (16). At the end of decoding all write phases vanish, so further calls preserve the output. □

The extension to prescribed call sequences will reuse earlier modules after loading. The following lemma gives the properties needed to let these modules decode on later visits.

Lemma 6 (Common decoding after loading)**.** *The modules constructed in Lemma 5 have the following properties: each maps the normalized state cube into itself; all loading modules agree on the entire slice $p_L = 0$; and their layered realizations retain the original normalized input coordinates through the last hidden layer.*

Proof. Normalization and final clipping give the first property. For the second, the block codes enter through $G_{2/3}(p_L, \theta_{L,r} w_r)$ in (17), and their origin-dependent feature increments through $G_2(p_L, \cdot)$ in Section B.2.1. These gates vanish at $p_L = 0$ for every candidate value. The last-load flag enters only through $(1 - \zeta_j)p_L$ and $\zeta_j p_L$ in the phase equations (Section B.1.3); fine decoding uses shared features, not the stored origin. Common normalization and clipping preserve this equality, including away from valid trajectories. Third, the layered realization retains the original normalized input coordinates through its last hidden layer, alongside the values needed to form its output. □

B.2.2 Counting the input and median output maps

The affine input map has $s_g(d + 1)$ parameters. The output map extracts the Q normalized output state variables, sorts them, and returns three times their median in the original coordinates. Its cost is paid once, separately from the recurrent calls.

For an explicit dense bound, put

$$k = \lceil \log_2 Q \rceil, \qquad p = 2^k, \qquad w_{\text{out}} = 2p, \qquad h_{\text{out}} = k(k + 1).$$

Pad the normalized outputs by ones to length p; the $(d + 1)$st order statistic is still the median of the original Q values. We use the classical bitonic sorting network. Its recursive merge adds k rounds at size 2^k, so $D_k = D_{k-1} + k$, $D_0 = 0$, and hence $D_k = k(k + 1)/2$ parallel compare-exchange rounds (Batcher, 1968).

To realize these sorting rounds as a dense ReLU network, for nonnegative v, w one round computes $\rho(v - w)$ alongside the input values, then forms $\min(v, w) = v - \rho(v - w)$ and $\max(v, w) = w + \rho(v - w)$ in its next hidden layer. At most $3p/2$ channels are needed in the first layer and p in the second. Padding both to $2p$ gives the stated width and depth; extraction and padding are part of the first affine map, and median selection and rescaling are part of the final one. Hence sufficient complete dense counts are

$$\begin{aligned} q_g &= w_g(s_g + 1) + (h_g - 1)w_g(w_g + 1) + s_g(w_g + 1), \\ b_g &= s_g(d + 1) + w_{\text{out}}(s_g + 1) + (h_{\text{out}} - 1)w_{\text{out}}(w_{\text{out}} + 1) + w_{\text{out}} + 1. \end{aligned} \tag{18}$$

In particular, $q_g = O(d^2 \log(d+1))$ and $b_g = O(d^2 \log^2(d+1))$. Fixed coefficients and zero entries are included. The parameter count and scalar-operation count of one evaluation of each component have the same order. Storing at most J modules and executing at most T calls therefore gives the total resource bounds, including the input and output maps,

$$P = O\big(Jd^2 \log(d+1) + d^2 \log^2(d+1)\big), \quad U = O\big(Td^2 \log(d+1) + d^2 \log^2(d+1)\big). \quad (19)$$

B.3 Completing the stage and call budgets

The construction above specifies grid sizes and the calls they require. We now choose those sizes within (J, T) and obtain exactly J stages without changing the computed function.

B.3.1 Exactly J separately stored modules

Lemma 7 (Stage splitting and padding)**.** *Fix $1 \le J \le T$ and a common module topology that realizes the identity on the state space. A stagewise architecture chosen independently of the target using $k \le J$ modules and $n \le T$ calls can be realized with exactly J separately stored modules and at most T calls, with the same output and module-size bounds.*

Proof. If $n \ge J$, split the original runs of repeated calls into exactly J nonempty pieces, in their original order, and store a separate copy of the corresponding module for each piece. Such a split exists because $k \le J \le n$: each split increases the number of pieces by one, up to the n individual calls. If $n < J$, first split all n calls and append $J - n$ identity modules. The resulting call count is $\max\{n, J\} \le T$ and the composition is unchanged. The splitting and padding rule is independent of the target. Each copy stores its own parameters. The construction assigns equal values to these copies, but their parameters are counted separately.

This construction needs no additional channels or layers. On a normalized cube, a dense module of width at least the state dimension realizes identity by carrying every coordinate through its hidden layers.

□

The normalized decoder in Lemma 5 already has this identity realization. Its dimension bounds therefore apply after splitting and padding. Lower bounds for at-most-J designs also hold for the exactly-J subclass. At $J = T$, positivity and $\sum_j r_j \le T$ force all $r_j = 1$; this is an untied composition of the fixed-size modules, not a claim about unrestricted feedforward architectures.

B.3.2 Choosing a grid within the available budget

The next lemma separates grid rounding from the network construction. It will be used both when the call allocation can be chosen and when it is prescribed. Restricting the number of fine cells per coordinate to a multiple of four simplifies the coordinate selectors in Section B.2.

Lemma 8 (Grid sizes within a storage and call budget)**.** *Let d, K, S be positive integers with $1 \le K \le S$ and $S \ge 8 \cdot 4^d$. There are integers $n \ge 1$ and $m \in 4\mathbb{N}$ such that, with $B = n^d$, $F = m^d$, $M = nm$ and $L = \lceil \log_3(2M+1) \rceil$,*

$$B \le K, \qquad B + L + F \le S, \qquad BF \ge \frac{KS}{64 \cdot 4^d}.$$

The choices depend only on d, K, S.

Proof. Set $u = S/8$ and choose

$$n = \left\lfloor \min\{K, u\}^{1/d} \right\rfloor, \qquad m = 4 \left\lfloor u^{1/d}/4 \right\rfloor.$$

Since $u^{1/d} \ge 4$, these satisfy $B \le K$ and $B + F \le 2u$. The elementary inequality $3^k \ge 2k + 1$ for integers $k \ge 1$ gives

$$3^{n+m} \ge (2n+1)(2m+1) \ge 2nm + 1, \qquad L \le n + m \le B + F.$$

Consequently, $B + L + F \le 2(B + F) \le 4u = S/2 \le S$. Using $\lfloor v \rfloor \ge v/2$ for $v \ge 1$ gives $n \ge \min\{K, u\}^{1/d}/2$ and $m \ge u^{1/d}/2$. Thus

$$BF \ge 4^{-d} \min\{K, S/8\}(S/8) \ge \frac{KS}{64 \cdot 4^d},$$

where the last inequality uses $K \le S$. □

B.3.3 The upper bound for every budget

Theorem 9 (The main upper bound with logarithmic-depth modules)**.** *For every $d \ge 1$, $T \ge 1$ and $1 \le J \le T$, a network design with the module dimensions of Lemma 5, exactly J stages and at most T calls satisfies*

$$P \le Jq_g + b_g, \qquad U = O(Tq_g + b_g), \qquad \|f - \Phi\|_\infty \le C\sqrt{d}\,(JT)^{-1/d}$$

for every $f \in \mathcal{F}_d^{\mathrm{Lip}}$ after a suitable parameter assignment. Here q_g, b_g are as in (18) *and C is absolute.*

Proof. For $T \ge 8 \cdot 4^d$, apply Lemma 8 with $(K, S) = (J, T)$ and then Lemma 5. The resulting error is at most

$$\frac{3(1 + \sqrt{d}/2)}{(BF)^{1/d}} \le A_d (JT)^{-1/d}, \qquad A_d = 12(1 + \sqrt{d}/2)64^{1/d} \le C\sqrt{d},$$

where C is absolute. Identity on the normalized cube fits the same dense topology, so Lemma 7 gives exactly J stages without changing its state, width or depth, or exceeding T calls.

For $T < 8 \cdot 4^d$, use J single-call identity modules and set the output map to zero. Since $|f| \le 1$ and $J \le T$,

$$(JT)^{1/d} \le T^{2/d} < 16 \cdot 64^{1/d} \le A_d,$$

so the same error bound holds. Both branches use the same padded dense topology. The module and input/output costs therefore give $P \le Jq_g + b_g$ and $U = O(Tq_g + b_g)$. □

This proves the upper half of Theorem 1. The matching capacity lower bound is proved next.

B.4 Capacity and the approximation lower bound

This argument supplies the lower half of Theorem 1. It applies to every fixed network design under the module-size bounds, independently of the decoder used for the upper bound. The proof follows the standard reduction from uniform approximation to shattering signs on a grid (Yarotsky, 2017, Theorem 4). We give an elementary Lipschitz extension of the grid data, then apply the parameter–runtime theorem with stored weights as parameters and all recurrent calls included in the runtime (Goldberg & Jerrum, 1995, Theorem 2.3).

Lemma 10 (From sign capacity to uniform error)**.** *Let $\mathcal{H}$ be a class of functions on $[0, 1]^d$ whose threshold class $\{\mathbf{1}[\Phi > 0] : \Phi \in \mathcal{H}\}$ has finite VC dimension V. Then*

$$\sup_{f \in \mathcal{F}_d^{\mathrm{Lip}}} \inf_{\Phi \in \mathcal{H}} \|f - \Phi\|_\infty \ge \tfrac{1}{8}(V + 1)^{-1/d}.$$

If $\mathcal{H}$ is the output class of an admissible network design with at most P stored parameters and U scalar operations, then its threshold class has $V \le C(P + 1)(U + 1)$ for an absolute constant C. Consequently, the module-size bounds imply $\mathcal{E}_d(J, T) \ge c_d(JT)^{-1/d}$ for every $1 \le J \le T$.

Proof. Take the n^d centers of grid cubes of side $1/n$ and assign arbitrary values $v_i \in \{-1/(4n), 1/(4n)\}$. These data satisfy the unit Euclidean-Lipschitz condition. The function

$$x \longmapsto \min_i \{v_i + \|x - x_i\|_2\},$$

clipped to $[-1/(4n), 1/(4n)]$, extends the data to a member of $\mathcal{F}_d^{\mathrm{Lip}}$. Uniform error strictly below $1/(4n)$ for every such target would realize all binary labelings on the grid. Choose $n = \lfloor V^{1/d} \rfloor + 1$. Then $n^d > V$ and $n \le 2(V + 1)^{1/d}$, proving the first assertion, including $V = 0$.

For the capacity bound, treat each target-dependent parameter as a real input to the evaluator; there are at most P such inputs. Target-independent coefficients are fixed, although they are included in P. Affine operations, each ReLU comparison and the final threshold test use $O(U+1)$ exact arithmetic and comparison operations. The parameter–runtime theorem therefore gives $V \le C(P+1)(U+1)$ (Goldberg & Jerrum, 1995, Theorem 2.3). Reusing a stored parameter does not create another independent real input.

Under the module-size bounds, $P \le q_d(J+2)$ and $U \le u_d(T+2)$. Thus $V+1 \le C_d JT$ for positive J, T. The constants are uniform over fixed graphs, coefficients and schedules. Taking the infimum over these designs after the worst-case error bound proves the last assertion. □

The same witness for Hölder targets. For $0 < \alpha \le 1$, replace the grid values above by $v_i \in \{-a, a\}$ with $a = 1/(4n^\alpha)$ and use the distance $\|x-y\|_2^\alpha$. This is a metric because $(s+t)^\alpha \le s^\alpha + t^\alpha$. Distinct grid centers are at least $1/n$ apart, so the assigned values satisfy the unit Hölder condition. The same minimum extension and clipping, with this metric and range $[-a, a]$, preserve that condition and the assigned values. Thus sign capacity V forces worst-case uniform error at least

$$a = \frac{1}{4n^\alpha} \ge \tfrac{1}{8}(V+1)^{-\alpha/d}, \qquad n = \lfloor V^{1/d} \rfloor + 1,$$

for the unit Hölder class with range $[-1, 1]$. The constant $1/8$ uses $\alpha \le 1$.

B.5 Constant-depth modules for finite L^p error

For finite p, bounded errors on sufficiently thin grid transition regions contribute arbitrarily little to the L^p error (Zhang et al., 2023, Appendix A.2). A single grid therefore suffices, allowing six-layer modules to attain the joint J, T upper rate.

Proposition 11 (Finite L^p approximation)**.** *For every integer $d \ge 1$, $1 \le p < \infty$ and integers $1 \le J \le T$, there is a multi-stage architecture chosen independently of the target with exactly J modules and at most T calls such that every $f \in \mathcal{F}_d^{\mathrm{Lip}}$ admits an assignment with*

$$\|f - \Phi\|_{L^p([0,1]^d)} \le C\sqrt{d}\,(JT)^{-1/d}, \tag{20}$$

where C is absolute. The modules have state dimension $4d+10$, width $24d+42$ in every hidden layer, and six hidden layers. In this construction, both the input map A_{in} and the readout $R_{\mathrm{out}} = A_{\mathrm{out}}$ are affine. The full costs satisfy

$$P = O((J+1)d^2), \qquad U = O((T+1)d^2).$$

The module topology depends only on d. For each p, J, T, the fixed numerical coefficients are chosen independently of f; the counted parameter assignment is then selected for f. Coefficient magnitude and precision are unrestricted.

We reuse the block codes, initialization and reflected-coordinate traversal of Lemma 3 and Section B.1.3. The changes are a two-layer box selector, parallel traversal carries and coordinatewise clipping at the end of each call. The latter two changes also appear in Section B.2.

Proof of Proposition 11. Step 1. Select a box in two layers. For a box with origin v and side length b, take $0 < 2\tau < b$ and $0 \le \lambda \le 1$. Define

$$S_\lambda(x; v, b) = \rho\left(\lambda - \sum_{i=0}^{d-1}\left[\rho\left(\frac{v_i + \tau - x_i}{\tau}\right) + \rho\left(\frac{x_i - v_i - b + \tau}{\tau}\right)\right]\right). \tag{21}$$

Every term in the sum is nonnegative. All terms vanish inside the box with a margin τ removed, whereas at least one term is ≥ 1 outside the open box. Hence S_λ lies in $[0, \lambda]$, equals λ on the smaller box, and is zero outside the open box. This uses $2d+1$ ReLUs in two layers. In particular, S_0 vanishes everywhere. Use the two-hinge ReLU gate G_R from Section B.2; its identity $G_R(0, z) = 0$ holds even for unbounded unused candidates.

Step 2. Realize one complete decoding update. Let $n, m \ge 1$ be integers, and put

$$B = n^d, \qquad F = m^d, \qquad M = nm, \qquad L = \lceil \log_3(2M+1) \rceil.$$

Each block code contains L digits for the initial value and $F-1$ increments. Write a_0 for the initial-value accumulator of Section B.1.3. Keep the $s = 4d + 10$ state variables

$$(x, z, o, a_0, q, y, c, b, N_A, N_F, p_L, p_A, p_F, p_D).$$

The four vectors x, o, c, b have dimension d and all other state variables are scalar. Initially, the input is x, all direction bits and p_L equal one, and all other state variables equal zero. A module stores its coarse origin v_j, last-loading flag ζ_j, and target code e_j. For $0 < \tau < (2M)^{-1}$, use

$$\theta_L = \beta + (1-\beta)S_1(x; v_j, 1/n), \qquad \theta_F = S_{p_F}(x - o; c/M, 1/M), \qquad \beta = \mathbf{1}_{\{B=1\}}.$$

Thus the fine selector already includes its phase flag. The singleton coarse grid uses the same graph with a constant loading selector.

Retain And and E_k from Section B.1.3. With the digit reader $r = D_4(z)$ of (13), put

$$z_S = 4z - r, \quad \iota = E_0(N_F), \quad s_F = \mathrm{And}(p_F, 1-\iota),$$
$$v_\Delta = q + (r-1)/M, \quad v = v_\Delta + G_2(\iota, a_0 - 1 - v_\Delta).$$

Before final clipping, update the code, origin, initial-value accumulator, current value and selected output as follows:

$$z^+ = z + G_{2/3}(p_L, \theta_L e_j) + G_{2/3}(p_A, z_S - z) + G_{2/3}(s_F, z_S - z),$$
$$o_i^+ = o_i + G_1(p_L, \theta_L v_{j,i}), \qquad a_0^+ = a_0 + G_6(p_A, 2a_0 + r/M),$$
$$q^+ = q + G_2(p_F, v - q), \qquad y^+ = y + G_2(\theta_F, v - y).$$

All products with codes, origins or M^{-1} are affine coefficients; there is no product of two state variables.

For the reflected traversal, form the endpoint and blocked bits t_i as in Section B.1.3, then compute all carries in parallel:

$$\kappa_0 = 1, \qquad \kappa_{i+1} = \rho\left(\sum_{j=0}^{i} t_j - i\right), \qquad f_i = \kappa_{i+1}, \qquad v_i = \kappa_i - \kappa_{i+1}.$$

On Boolean inputs each carry is the conjunction of the preceding blocked bits. The signed moves, direction changes, counters and one-hot phase equations are otherwise unchanged. Their affine sums are computed in parallel and their $O(d^2)$ coefficients and scalar operations are counted. The fine selector reads the old cell before the traversal advances. This includes $m = 1$, when every coordinate is blocked and no coordinate moves.

Normalize the raw state intervals affinely to $[0,1]^s$, and clip to that cube only at the end of the call. The raw intervals are $[0,1]$ for x, o, b and the flags, $[0, 2/3]$ for z, $[0,6]$ for a_0, $[-1,1]$ for q, y, $[0, \max\{1, m-1\}]$ for each c_i, $[0, L]$ for N_A, and $[0, F]$ for N_F. All scales are positive.

The counters, directions and phases follow the valid trajectory of Section B.1.3 independently of the decoded registers. Outside the grid face strips, exactly one coarse selector loads $(z, o) = (e_j, v_j)$. Initial-value calls preserve a legal code suffix and decoded prefix.

On the first fine call, $q = 0$, $v_\Delta \in [-1/M, 1/M]$, and $a_0 - 1 \in [-1, 1]$, so the radius-two gate selects the initial value. Later calls read valid increments and write y only at the matching cell. Hence the register bounds hold, clipping is inactive, and after $B + L + F$ calls y is the encoded cell value. Clipping bounds the output on the remaining inputs, and the done phase makes further calls to the last module preserve it.

Step 3. Count six hidden layers. The two selectors use $4d + 2$ ReLUs. The traversal and its gated updates use $14d$, and the origin writes use $2d$. Scalar operations use another 30: four digit-reader hinges, two for selecting the current value, six code-write hinges, two each for the initial value, value and output writes, four for ι and s_F, and eight for the phase-completion tests. The total is therefore $N = 20d + 32$. The longest path is the reflected traversal: endpoint hinges occupy the first level, followed by blocked bits, parallel prefix carries, signed moves, and gated counter or direction updates at level five. The box selectors and digit, value and output updates also fit within these levels: each state update uses the old state and intermediate expressions, never another updated state variable.

As in Section B.2, carry the s normalized nonnegative inputs and the ReLU values already computed. Width $w = s + N = 24d + 42$ therefore suffices. The sixth hidden layer clips the updated state. It uses $2s$ clipping hinges and retains the s original inputs, with $3s \le w$. The final affine map returns the clipping differences, or the retained input in identity mode. Every hidden layer is padded to the same width w; there are no skip connections. The complete dense counts are

$$q_{\mathrm{lp}} = w(s+1) + 5w(w+1) + s(w+1),$$
$$b_{\mathrm{lp}} = s(d+1) + (s+1) = s(d+2) + 1. \tag{22}$$

Here b_{lp} counts affine initialization and scalar readout. Both counts are $O(d^2)$ and include every fixed coefficient and zero entry. The exact scalar-operation counts have the same order.

Step 4. Bound the integral error and fit the budgets. At each fine-cell center $x_{j,k}$, encode $q_{j,k} = \lfloor Mf(x_{j,k}) \rfloor$. Neighboring centers are $1/M$ apart, so their quantized values have increments in $\{-1, 0, 1\}$. The good-region error is at most $(1 + \sqrt{d}/2)/M$. The union of the fine-grid face strips has measure at most $2d(M+1)\tau \le 4dM\tau$ and contains the coarse-grid boundaries. Since the target and output both lie in $[-1, 1]$,

$$\|f - \Phi\|_p \le \frac{1 + \sqrt{d}/2}{M} + 2(4dM\tau)^{1/p}.$$

Taking $\tau = (2^{p+2}dM^{p+1})^{-1}$ makes the second term $1/M$ and satisfies $0 < 2\tau < 1/M$. Thus the error is at most $(2 + \sqrt{d}/2)/M$.

For $T \ge 8 \cdot 4^d$, use the grids of Lemma 8; the decoder above accepts these choices of n, m. Since

$$\frac{2 + \sqrt{d}/2}{M} \le \frac{3(1 + \sqrt{d}/2)}{M},$$

the budget calculation in Theorem 9 gives (20). Below that threshold, use zero output and the same small-budget estimate, since its L^p error is at most one. The identity mode and Lemma 7 give exactly J stages and at most T calls in every case, with no change of local topology. Copied modules occupy separate parameter vectors. Hence $P \le Jq_{\mathrm{lp}} + b_{\mathrm{lp}}$ and $U = O(Tq_{\mathrm{lp}} + b_{\mathrm{lp}})$, as claimed. □

The norm-dependent choice is the strip width and the associated weights. In particular, $\tau^{-1} = 2^{p+2}dM^{p+1}$ grows with p and resolution. The argument applies to each finite p and does not give a uniform-error guarantee for one fixed numerical network by sending p to infinity.

B.6 Uniform approximation of Hölder functions

The Lipschitz condition controls how much neighboring grid values can differ. Changing the quantization scale allows the same decoder to handle Hölder functions. Choosing the quantization scale from target regularity is already part of the finite-alphabet coding approach (Yarotsky & Zhevnerchuk, 2020). The result below transfers our joint storage–call bounds to this class; it does not require a new decoder. For $0 < \alpha \le 1$, define

$$\mathcal{H}_d^\alpha = \{f : [0,1]^d \to [-1,1] : |f(x) - f(y)| \le \|x - y\|_2^\alpha \text{ for all } x, y\}.$$

Let $\mathcal{E}_{d,\alpha}(J,T)$ denote (2) with $\mathcal{F}_d^{\mathrm{Lip}}$ replaced by $\mathcal{H}_d^\alpha$, under the same module-size bounds. In particular, $\mathcal{E}_{d,1} = \mathcal{E}_d$.

Corollary 12 (Hölder uniform approximation frontier)**.** *For every $d \ge 1$ and $0 < \alpha \le 1$, there are $c_{d,\alpha} > 0$ and an absolute constant $C > 0$ such that, for all integers $1 \le J \le T$,*

$$c_{d,\alpha}(JT)^{-\alpha/d} \le \mathcal{E}_{d,\alpha}(J,T) \le Cd^{\alpha/2}(JT)^{-\alpha/d}. \tag{23}$$

The upper bound uses the module dimensions of Lemma 5 and the same full resource counts (19)*. The topology is independent of α, J, T. For each α, J, T, the fixed numerical coefficients are chosen independently of the target, with unrestricted magnitude and precision.*

Proof. Upper bound. Use the grids and shifts in Lemma 4. At resolution $M = nm$, replace its normalized targets by

$$\lambda_M = 3^\alpha M^{1-\alpha}, \qquad g_r(v) = \lambda_M^{-1} f\Big(\mathrm{clip}_{[0,1]^d}(3v - \mathbf{1} - 3\delta_r \mathbf{1})\Big).$$

Since $\lambda_M \geq 1$, we have $|g_r| \leq 1$. Clipping is nonexpansive, so neighboring fine-cell centers v, w satisfy

$$|g_r(v) - g_r(w)| \leq \lambda_M^{-1}(3/M)^\alpha = M^{-1}.$$

Thus $q = \lfloor Mg_r \rfloor$ lies in $[-M, M]$ and has successive increments in $\{-1, 0, 1\}$. These are precisely the code conditions of Lemma 3; global Lipschitz regularity of g_r is unnecessary.

We reuse the circuit constructed in the proof of Lemma 5, rather than its Lipschitz-only approximation statement. The circuit reads any block codes satisfying the conditions of Lemma 3, as the Hölder codes above do. Its spatial features, state bounds, gates and $B + L + F$ calls are unchanged. Replace the final output scaling by λ_M, so that the output is $\lambda_M \operatorname{med}_r y_r$. For a good shifted copy, $y_r = q/M$ at a center within distance $\sqrt{d}/(2M)$ of $u_r(x)$, and hence

$$|f(x) - \lambda_M y_r(x)| \leq \left(\frac{3\sqrt{d}}{2M}\right)^\alpha + \frac{\lambda_M}{M} = 3^\alpha\big[1 + (\sqrt{d}/2)^\alpha\big]M^{-\alpha}.$$

The shifted-grid argument still makes a strict majority of copies good at every input, so the median has the same bound. Scaling changes only the existing final affine coefficients of the median network; all recurrent state variables retain their original ranges.

For $t \geq 0$, $1 + t^\alpha \leq 2(1 + t)^\alpha$, so the grid error is at most

$$2\left[\frac{3(1 + \sqrt{d}/2)}{M}\right]^\alpha.$$

Use the grid choice, stage splitting and padding from Theorem 9. With $A_d \leq C_0\sqrt{d}$ as in that proof, the resulting error is at most

$$2A_d^\alpha(JT)^{-\alpha/d} \leq Cd^{\alpha/2}(JT)^{-\alpha/d},$$

where C is absolute. In the zero-output branch of that proof, $1 \leq A_d(JT)^{-1/d}$; raising this inequality to α proves the same bound. Both branches retain the same topology and full parameter and operation counts.

Lower bound. Fix a network design and let V be the VC dimension of its threshold class. The Hölder version of the grid witness in Appendix B.4 gives worst-case uniform error at least $\frac{1}{8}(V + 1)^{-\alpha/d}$ on $\mathcal{H}_d^\alpha$. The parameter–runtime part of Lemma 10 is independent of the target class and gives $V + 1 \leq C_d JT$ uniformly over admissible network designs. Taking the infimum over designs proves the lower bound. □

For fixed d and α, uniform accuracy ε therefore requires JT of order $\varepsilon^{-d/\alpha}$. Less regular targets need a larger budget, while the same exchange between stored modules and calls remains available. This can inform representation budgets for spatial surrogates with a known Hölder bound. It is a worst-case approximation statement; the output scale grows with resolution when $\alpha < 1$, and no bound on numerical precision is imposed.

C Proofs for prescribed call sequences

We prove Theorem 2 using the spatial decoder from Lemma 5. The upper bound makes it load each selected module once and decode on subsequent calls, even when they return to an earlier module. The lower bound counts parameters in order of first access. We then derive the criterion for retaining the frontier and the same-count comparison, before specializing to contiguous-stage allocations.

Throughout, σ is fixed independently of the input and target, visits all J modules, and has length T. Modules are relabelled by first appearance. The quantities τ_j, H, W and the minimax class are those of Section 5. For the proof, write $R_j = T - \tau_j + 1$ for the calls from module j's first visit onward, including that visit. Then $H(\sigma) = \max_{1 \leq j \leq J} jR_j$. The common module-size bounds include the construction below. Input and output maps are evaluated once and counted.

C.1 Loading once and decoding after later visits

The normalized decoder of Lemma 5 uses $B = n^d$ coarse blocks, $F = m^d$ fine cells with $m \in 4\mathbb{N}$, $M = nm$, and $L = \lceil \log_3(2M+1) \rceil$. Its B loading updates followed by $L + F$ decoding updates give uniform error at most $3(1+\sqrt{d}/2)/M$. Write D_j for its loading module for block j and p_L for the loading flag. By Lemma 6, each D_j preserves the normalized state cube, all D_j agree on $p_L = 0$, and their layered realizations retain the original input coordinates. Thus an earlier loading module can perform later decoding, with its inputs available for the gate below.

Using an auxiliary state to switch between repeated maps is an established construction (Zhang et al., 2023, Proposition 3.3 and Appendix E). Here the added control tracks loading progress: it must load each selected module only once, then allow every later visit to perform the common update. The next lemma supplies this rule and its module dimensions.

Lemma 13 (Loading on first visits)**.** *If $B \le J$ and $1 + L + F \le R_B$, the decoder can follow the prescribed sequence using modules with*

$$s = 13d + 11, \qquad w = 108d + 45, \qquad h = 9 + \lceil \log_2 d \rceil. \tag{24}$$

Every hidden layer has width w. One call either completes one decoder update or preserves the state. The affine input and ReLU median output maps remain outside the recurrent modules. Each recurrent module has q dense parameters, and the input and output maps together have b, where

$$\begin{aligned} q &= w(s+1) + (h-1)w(w+1) + s(w+1) = O(d^2 \log(d+1)), \\ b &= s(d+1) + w_{\mathrm{out}}(s+1) + (h_{\mathrm{out}} - 1)w_{\mathrm{out}}(w_{\mathrm{out}} + 1) + w_{\mathrm{out}} + 1 = O(d^2 \log^2(d+1)), \end{aligned} \tag{25}$$

where $w_{\mathrm{out}}, h_{\mathrm{out}}$ are the median network dimensions in Section B.2. Thus $P \le Jq + b$ and $U = O(Tq + b)$.

Proof. Enable each load once. Assign the B block codes to the first B modules, with the last-load flag set only at $j = B$. Add a normalized progress coordinate $c \in [0, 1]$, initially zero. For $1 \le j \le B$, with $C_{01}(v) = \rho(v) - \rho(v-1)$, set

$$\begin{aligned} e_j(c) &= C_{01}(j - Bc), & a_j &= e_j(c) - \rho(e_j(c) - p_L), \\ \beta_j &= 1 - p_L + a_j, & c^+ &= c + a_j/B. \end{aligned} \tag{26}$$

For each normalized base-state coordinate, use

$$z_i^+ = \rho(z_i - \beta_j) + D_{j,i}(z) - \rho(D_{j,i}(z) - \beta_j). \tag{27}$$

For $j > B$, use separately stored copies of the entire controlled module at $j = B$, including its gate and counter update. All right-hand sides read the old state. No multiplication of two state variables is used.

On the whole cube, $a_j = \min\{e_j, p_L\}$ gives $0 \le a_j \le p_L$ and $0 \le \beta_j \le 1$. For every $1 \le j \le B$,

$$0 \le Bc^+ \le Bc + C_{01}(j - Bc) \le \max\{Bc, j\} \le B;$$

and the copied modules satisfy the same bound. The state update equals $\rho(z_i - \beta_j) + \min\{D_{j,i}(z), \beta_j\}$, which lies in $[0, 1]$. It equals z_i at $\beta_j = 0$ and $D_{j,i}(z)$ at $\beta_j = 1$. Thus the complete enlarged cube remains invariant.

Follow the prescribed sequence. Until B different modules have appeared, $p_L = 1$ and Bc counts those already loaded. A repeated label has $j \le Bc$, so $e_j = 0$ and the state waits. Relabelling by first appearance makes the next new label exactly $j = Bc + 1$; a larger unseen label cannot occur first. Thus $e_j = 1$ and this call performs its load. Every load increases Bc by one. Thus c always equals the number of completed loads divided by B; the Bth load gives $c = 1$ and $p_L = 0$. Thereafter $a_j = 0$, $\beta_j = 1$, and c stays fixed for every label. Labels above B first appear after loading has finished. The common-update property therefore makes all remaining calls decode, including returns to earlier modules. There are $R_B - 1 \ge L + F$ calls after the last load; further calls preserve the completed output. This also covers $B = 1$.

Realize the gate within the stated module size. The base realization carries its original normalized inputs through every hidden layer. Three additional channels suffice. In the first layer they are $c, u_+ = \rho(j - Bc), u_- = \rho(j - Bc - 1)$; in the second they become

$$c, \qquad e = \rho(u_+ - u_-), \qquad v = \rho(u_+ - u_- - p_L).$$

Carry c, e, v thereafter. The quantities $a_j = e - v$, β_j, and c^+ are affine in these channels and the original p_L, so the width is $w_g + 3 = 108d + 45$ throughout the original h_g layers. Append one hidden layer containing, for each base coordinate, $D_{j,i}(z)$ and the two gating hinges in (27), together with c^+. These nonnegative values need $3s_g + 1$ channels, which fit within the same width. The final affine layer forms the differences. Hence $s = s_g + 1$ and $h = h_g + 1$, giving (24). Padding unused channels and counting all matrix entries and biases gives (25). The output map ignores c through counted zero coefficients. Every scheduled call evaluates its selected module in full, including calls that preserve the state. □

Grid sizes for every budget. Choose an index a attaining $H = \max_j jR_j$, set $k = \lceil a/2 \rceil$, and write $R = R_k$. Since first visits are distinct calls, $\tau_a - \tau_k \geq a - k$, so $R_k \geq a - k + 1 \geq k$. Also $kR_k \geq aR_a/2 = H/2$. If $R \geq 8 \cdot 4^d$, apply Lemma 8 with $(K, S) = (k, R)$. It supplies $B \leq k$, $F = m^d$ with $m \in 4\mathbb{N}$, and $B + L + F \leq R$, with

$$BF \geq \frac{kR}{64 \cdot 4^d} \geq \frac{H}{128 \cdot 4^d}.$$

Because $B \geq 1$ and $R_B \geq R$, the condition $1 + L + F \leq R_B$ follows. The preceding lemma gives error at most $12(1 + \sqrt{d}/2)128^{1/d} H^{-1/d} \leq C\sqrt{d}\, H^{-1/d}$. If $R < 8 \cdot 4^d$, then $H \leq 2kR \leq 2R^2 < 128 \cdot 16^d$; zero output gives the same bound since $16 \leq 12(1 + \sqrt{d}/2)$. All grid choices and fixed coefficients depend only on d, σ, before the target. This proves the upper half of Theorem 2.

C.2 Capacity along a fixed call sequence

We use the standard activation-pattern partition of parameter space and polynomial sign counting (Bartlett et al., 2019, Section 4 and Lemma 17). For a prescribed sequence, the additional step is to track when each module's parameters first enter the computation. We group calls while the number of visited modules stays within a factor of two, which bounds the parameter variables available in each group. Repeated visits add operations without introducing new variables, so the total group cost is controlled by W.

Lemma 14 (Capacity of a prescribed call sequence)**.** *For every fixed admissible design with sequence σ, the class $\{\mathbf{1}[\Phi_{\mathcal{A},\theta} > 0] : \theta\}$ has VC dimension at most $C_d W(\sigma)$. The constant is uniform over designs and sequences under the module-size bounds.*

Proof. Fix $n \geq 1$ sample inputs. Let n_t count the distinct modules visited by call t. Because n_t is nondecreasing, the following groups of calls are contiguous:

$$I_i = \{t : 2^i \leq n_t < 2^{i+1}\}, \qquad 0 \leq i \leq \lfloor \log_2 J \rfloor.$$

Throughout I_i, fewer than 2^{i+1} modules have been reached, so parameters of later modules cannot yet enter the computed state. Include the input map in the first interval and the output map and final threshold test in the last. Let $g_i \geq 1$ bound the operations in interval i and $p_i \geq 1$ the parameter variables available by its end. The common module-size bounds give

$$g_i \leq C_d(|I_i| + 2), \qquad p_i \leq C_d 2^i.$$

In particular,

$$\sum_i p_i g_i \leq C_d W(\sigma), \qquad \sum_i p_i \leq C_d J. \tag{28}$$

Indeed, $W = \sum_{t=1}^{T} n_t$ and $2^i \leq n_t$ on I_i; the sum $\sum_i p_i g_i$ therefore charges each call at most a constant multiple of n_t. The bound $\sum_i 2^i \leq 2J$ absorbs the input and output costs because $W \geq J$.

Fix a region of parameter space on which all preceding activation decisions, for every sample, are specified. Incoming state coordinates then have fixed polynomial expressions in the preceding parameters. Within any subsequent branch, every preactivation is a polynomial of degree at most

$D = C_d(T+2)$. An affine layer increases parameter degree by at most one, and a fixed ReLU branch either preserves its input or returns zero. This uses the affine/ReLU model, without multiplication of two computed states.

For one sample, enumerate all branches within group i and all predicates on those branches. There are at most $(g_i+1)2^{g_i}$ predicates. Their union over the samples contains at most $M_i = n(g_i+1)2^{g_i}$ polynomials, each involving at most p_i variables and of degree at most D. The signs of this union determine the actual branch of every sample, hence their joint activation pattern. Unrealizable branches only enlarge the union. Polynomial sign counting bounds the number of refinements by $[CD(M_i+1)]^{p_i}$ for all $M_i, p_i \geq 1$; the trivial 2^{M_i} bound suffices when $M_i < p_i$ (Bartlett et al., 2019, Lemma 17). We use the strict threshold $\mathbf{1}[z > 0]$, which assigns zero to $z = 0$; polynomial zero sets are included in the sign count. The logarithm of this uniform bound is at most

$$C_d p_i \big[g_i + \log(n(T+2))\big],$$

where the constants depend only on the module-size bounds and d. Restricting sign vectors to a preceding region cannot increase their number. The bound is uniform over its polynomial expressions and does not require the region to be connected.

Let $\Pi(n)$ denote the largest number of binary output labelings realizable on any n inputs. Multiplying the refinement counts and using (28) therefore gives the growth-function estimate

$$\log \Pi(n) \leq C_d W(\sigma) + C_d J \log(n(T+2)). \tag{29}$$

If these n samples are shattered, $\log \Pi(n) = n \log 2$. Using $\log x \leq C\sqrt{x}$ separately for n and $T+2$ gives

$$n \leq C_d\big[W(\sigma) + J\sqrt{n} + J\sqrt{T}\big] \leq \tfrac{1}{2}n + C_d'\big[W(\sigma) + J^2 + T\big] \leq \tfrac{1}{2}n + C_d'' W(\sigma).$$

The middle inequality is Young's inequality; the last uses $W \geq T$ and $W \geq J(J+1)/2$. Thus every shattered sample has $n \leq C_d W(\sigma)$, uniformly over fixed designs, proving the claim. □

The grid argument in Lemma 10 now gives $c_d W(\sigma)^{-1/d}$ as a uniform lower bound on approximation error. Taking the infimum over designs proves the lower half of Theorem 2.

C.3 Frontier retention and the same-count comparison

The timing criterion follows from the elementary comparison

$$\frac{W^2}{4JT} \leq H \leq W \leq JT. \tag{30}$$

For the first inequality, set $k = \lceil W/(2T) \rceil \in \{1, \ldots, J\}$. Double counting gives $W = \sum_{b=1}^{J} R_b$. Since the remaining-call counts are nonincreasing,

$$W = \sum_{b=1}^{J} R_b \leq (k-1)T + JR_k < \frac{W}{2} + JR_k.$$

Thus $k \geq W/(2T)$ and $R_k > W/(2J)$, giving $H \geq kR_k \geq W^2/(4JT)$. For the remaining inequalities, every b satisfies $bR_b \leq \sum_{j=1}^{b} R_j \leq W$, and $R_b \leq T$.

A second comparison gives the logarithmic gap stated in Theorem 2. Since $H \geq T$ and $R_b \leq \min\{T, H/b\}$, monotonicity of the integrand gives

$$H \leq W \leq T + \int_1^J \min\{T, H/x\}\, dx = H[1 + \log(JT/H)]. \tag{31}$$

Thus the ratio of the upper to the lower bound is at most $C_d[1 + \log(JT/H)]^{1/d}$. They match up to constants when $H \asymp W$; otherwise a gap may remain. We next characterize exactly when they recover the main frontier.

Corollary 15 (Enough calls after enough first visits)**.** *Fix constants $a, b \in (0, 1]$. If some first visit τ_k satisfies*

$$k \geq aJ, \qquad R_k \geq bT, \tag{32}$$

then, under fixed module-size bounds large enough for the construction,

$$\mathcal{E}_d(\sigma) \asymp_{d,a,b} (JT)^{-1/d}.$$

Conversely, if $\mathcal{E}_d(\sigma) \leq K(JT)^{-1/d}$ for a fixed $K > 0$, there is $a > 0$, depending only on d, K and the module-size bounds, and an index k with $k \geq aJ$ and $R_k \geq aT$.

Proof. Condition (32) gives $H \geq abJT$ directly. The rate follows from Theorem 2 and $W \leq JT$. Conversely, the lower bound gives $W \geq cJT$ for some $0 < c \leq 1$ depending only on d, K and the module-size bounds. The index used to prove (30) then satisfies $k \geq W/(2T) \geq cJ/2$ and $R_k > W/(2J) \geq cT/2$. This gives the explicit first-visit condition with $a = c/2$. □

This proves the characterization in Section 5: along a sequence of budgets, attaining the frontier with a fixed error multiplier is equivalent to H (or W) remaining a fixed fraction of JT. The fractions depend only on that multiplier, d, and the module-size bounds, independently of the budgets.

Every sequence has $R_1 = T$ and $R_j \geq J - j + 1$, because modules not yet visited must still appear. Consequently,

$$H \geq \max\{T, J^2/4\}. \tag{33}$$

Repeated cycles and balanced grouped calls. For a sequence of r repetitions of $(1, \ldots, J)$, we have $T = Jr$ and $\tau_j = j$, so

$$W = JT - \frac{J(J-1)}{2}, \qquad H = \max_{j \leq J} j(T - j + 1) \geq JT/4.$$

The last inequality follows by taking $j = \lceil J/2 \rceil$. Any sequence beginning with an initial full pass through its modules has the same first-visit times. For balanced grouped calls, the rounded uniform allocation in Proposition 16 gives $H \geq JT/4$; the same order holds when no module receives more than a fixed multiple of T/J calls. Figure 6(c,d) illustrates the access curves for balanced grouped calls and repeated cycles.

Identical counts, different access times. For σ_A in (11), $R_1 = T$ and $R_j = J - j + 1$ for $j \geq 2$. Hence

$$W = T + J(J-1)/2, \qquad H \asymp T + J^2.$$

For σ_B, $\tau_j = j$ and the preceding calculation gives $H \asymp W \asymp JT$. Theorem 2 therefore proves (12), under the same local size bounds. Figure 6(a,b) shows how these identical call counts produce different access curves and values of H, W. Each sequence has its own design and target-dependent parameter assignment.

When an order separation is possible. If $J \leq J_0$, then $H \geq JT/J_0$; if $J \geq aT$, then $H \geq aJT/4$. In either regime every sequence retains the main order. For the same-count example the error ratio is of order $[JT/(T + J^2)]^{1/d}$, which diverges exactly when $J \to \infty$ and $J/T \to 0$, since $(T + J^2)/(JT) = 1/J + J/T$. Thus, for prescribed sequences as well, attaining $O(T^{-2/d})$ requires and is ensured by $J = \Omega(T)$ distinct visited modules.

Viewing a cycle as one larger block. Grouping one cycle into $\mathcal{B} = G_J \circ \cdots \circ G_1$ gives $\mathcal{B}^r$, but this larger block stores and evaluates all constituents. If each has parameter count q, depth h, and work u, the concatenated graph has $O(Jq)$ parameters, $O(Jh)$ depth, and $O(Ju)$ work per cycle. Its r repetitions remain $T = Jr$ constituent calls. It is not one module of size bounded independently of J.

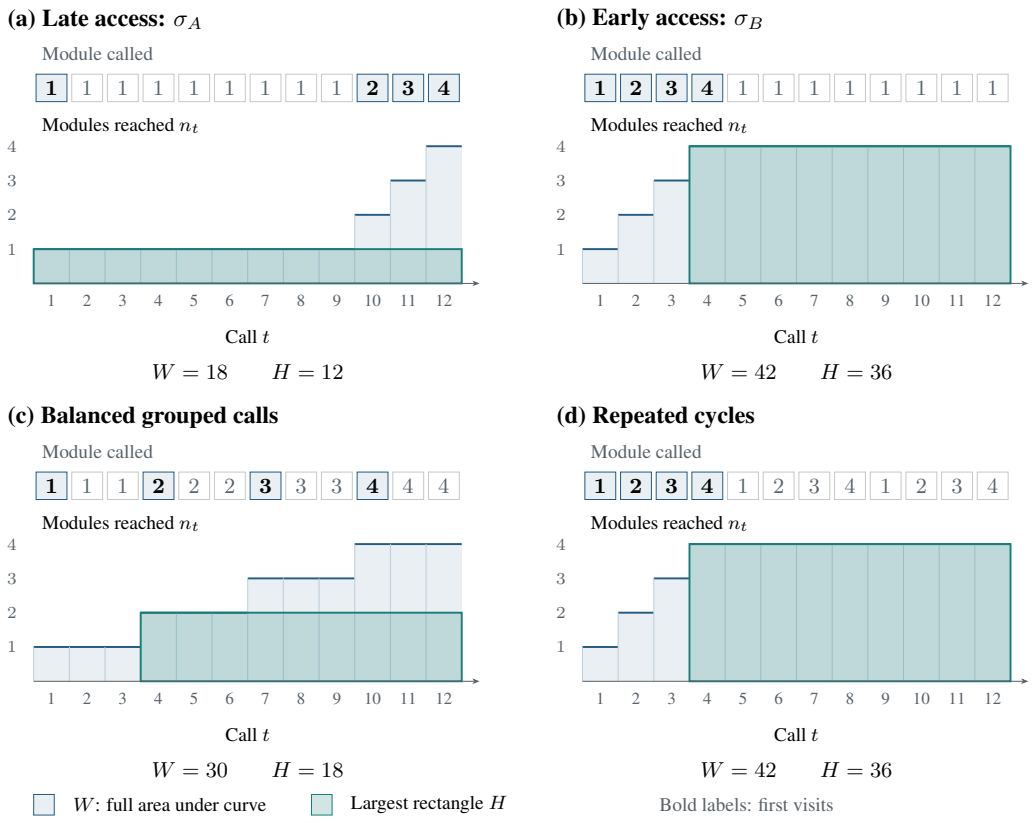


Figure 6: Call sequences and their access curves for $J = 4$ and $T = 12$. Each timeline entry is a module label, aligned with one call below. Panels (a,b) have identical per-module counts but delay or advance access to modules $2, 3, 4$. Panels (c,d) call each module three times, either consecutively or cyclically. The full area under each curve, including the green rectangle, gives W; the green rectangle attains H. Balanced grouped calls and repeated cycles both retain the $(JT)^{-1/d}$ approximation order. The displayed values describe the bounds, not exact approximation errors.

C.4 Consequences for contiguous-stage allocations

For an allocation vector $r = (r_1, \ldots, r_J)$ of positive integers with sum T, set

$$\sigma = \big(\underbrace{1, \ldots, 1}_{r_1 \text{ calls}}, \ldots, \underbrace{J, \ldots, J}_{r_J \text{ calls}}\big),$$

and write $\mathcal{E}_d(r) = \mathcal{E}_d(\sigma)$. Thus $\mathcal{E}_d(r)$ fixes a contiguous allocation with exactly T calls; $\mathcal{E}_d(J, T)$ in (2) also minimizes over stage lengths with at most T calls. Since $\tau_j = 1 + \sum_{i<j} r_i$, we have exactly

$$R_j = \sum_{i=j}^{J} r_i, \qquad H(r) = \max_j j R_j, \qquad W(r) = \sum_j j r_j.$$

All results below follow from the fixed-sequence theorem. We quantify concentration at one stage and then the robustness of uniform repetition.

C.4.1 Concentrating the additional calls at one stage

Now let $r_j = 1 + (T - J)\mathbf{1}[j = k]$ and put $E = T - J$. Then

$$R_b = J - b + 1 + E\mathbf{1}[b \le k], \qquad W = \frac{J(J+1)}{2} + kE.$$

Writing $S = J^2 + kT$, we show directly that

$$S/8 \le H \le W \le S.$$

The mandatory call at each stage gives $H \ge J^2/4$. Taking $b = \lceil k/2 \rceil$ gives $b \ge k/2$ and $R_b = T - b + 1 \ge T/2$, so $H \ge kT/4$ as well. Finally, $H \le W$, and the displayed formula gives $W \le J^2 + kT = S$. These inequalities and Theorem 2 give

$$\mathcal{E}_d(r) \asymp_d (J^2 + kT)^{-1/d}. \tag{34}$$

For $k \ge aJ$, this gives $\mathcal{E}_d(r) \asymp_{d,a} (JT)^{-1/d}$.

If $J \asymp T^\eta$ and $k \asymp T^\beta$, with $0 \le \beta \le \eta < 1$, the polynomial error exponent is $\max\{2\eta, 1+\beta\}/d$, with matching bounds up to constants. For $0 < \eta < 1$ and $\beta < \eta$, the exponent is smaller than $(1+\eta)/d$ by $\min\{1-\eta, \eta-\beta\}/d$. The two contributions cross at k of order J^2/T when this falls within the integer range.

For concentration at stage k in the regime $J = o(T)$, the formula $\mathcal{E}_d(r) \asymp_d (J^2 + kT)^{-1/d}$ shows that attaining the main frontier requires and is ensured by $k = \Omega(J)$. No logarithmic uncertainty remains for this family.

C.4.2 Uniform repetition and redistribution of calls

For example, allocations with $r_1 \le \cdots \le r_J$ have $R_k \ge T/2$ at $k = \lfloor J/2 \rfloor + 1$, so $H \ge JT/4$. More generally, if $r_j \le KT/J$ for fixed $K \ge 1$, take $k = \lfloor J/(2K) \rfloor + 1$. At most $T/2$ calls precede this stage, and $H \ge JT/(4K)$. The next result gives a sharper bound for a specific integer rounding of uniform allocation and shows that its approximation order survives large changes in the calls per stage.

Proposition 16 (Uniform allocation and redistribution of calls)**.** *Define the positive integer allocation*

$$r_j^{\mathrm{unif}} = \left\lfloor \frac{jT}{J} \right\rfloor - \left\lfloor \frac{(j-1)T}{J} \right\rfloor. \tag{35}$$

Then

$$H(r^{\mathrm{unif}}) \ge \frac{JT}{4} \ge \frac{1}{4} \max_r H(r).$$

More generally, for a positive integer allocation r with the same total, let

$$\Delta(r) = \frac{1}{2} \sum_{j=1}^{J} |r_j - r_j^{\mathrm{unif}}|. \tag{36}$$

If $\Delta(r) \le \delta T$ with $0 \le \delta < 1$, then

$$H(r) \ge \frac{(1-\delta)^2}{4} JT, \qquad \mathcal{E}_d(r) \asymp_{d,\delta} (JT)^{-1/d}.$$

More precisely, for every $0 \le \delta \le 1$,

$$\max_{\substack{r \in \mathbb{N}^J,\ \sum_j r_j = T \\ \Delta(r) \le \delta T}} \mathcal{E}_d(r) \asymp_d \left[T + J^2 + (1-\delta)^2 JT\right]^{-1/d}. \tag{37}$$

Here $\mathbb{N} = \{1, 2, \ldots\}$, and the constants are independent of δ, J, T. Each error $\mathcal{E}_d(r)$ allows its own admissible design.

Proof. Put $\varepsilon = 1 - \delta$ and $S = T + J^2 + \varepsilon^2 JT$. We first bound H for every allocation with $\Delta(r) \le \delta T$. Equal totals imply that $\Delta = \Delta(r)$ is both the sum of the positive differences and the absolute sum of the negative differences. Thus it counts the calls moved, and every suffix loses at most Δ calls. If $\varepsilon > 0$, take $b = \lceil \varepsilon J/2 \rceil \in \{1, \ldots, J\}$. Since $b - 1 < \varepsilon J/2$ and $\Delta \le (1-\varepsilon)T$, the telescoping definition of the uniform allocation gives

$$R_b \ge T - \left\lfloor \frac{(b-1)T}{J} \right\rfloor - \Delta \ge \frac{\varepsilon}{2} T.$$

Together with $b \geq \varepsilon J/2$, this gives $H \geq \varepsilon^2 JT/4$, which also holds at $\varepsilon = 0$. Every positive allocation also has $H \geq T$ and $H \geq J^2/4$ by (33). Combining the three inequalities gives $S \leq 9H$, so Theorem 2 proves the upper bound in (37), uniformly over $0 \leq \delta \leq 1$.

For a matching lower bound, write $u_j = r_j^{\mathrm{unif}}$, set $K = \max\{1, \lceil \varepsilon J \rceil\}$, and move all calls beyond the mandatory first call at stages $j > K$ to stage one:

$$L = \sum_{j>K} (u_j - 1), \qquad r_1 = u_1 + L, \quad r_j = u_j \ (2 \leq j \leq K), \quad r_j = 1 \ (j > K).$$

Since $u_j - 1 \leq T/J$ and $J - K \leq \delta J$, this allocation has $\Delta(r) = L \leq \delta T$. Moreover,

$$W(r) = \frac{J(J+1)}{2} + \sum_{j \leq K} j(u_j - 1) + L \leq J^2 + \frac{T}{J} K^2 + T \leq J^2 + 2\varepsilon^2 JT + 3T \leq 3S.$$

Here $K \leq \varepsilon J + 1$ and $J \geq 1$. The lower bound in Theorem 2 applied to this allocation completes (37).

Taking $\delta = 0$ in this bound on H gives $H(r^{\mathrm{unif}}) \geq JT/4$; $H \leq JT$ gives the comparison with $\max_r H(r)$. For any fixed $\delta < 1$, the same bound and $W \leq JT$ give the claimed rate for every admissible allocation by Theorem 2. □

Thus uniform allocation attains the best possible order without optimizing individual call counts. Maximizing H instead can improve the factor $H^{-1/d}$ in this construction guarantee by at most $4^{1/d}$. This is a comparison of guaranteed upper bounds, not an exact ratio of minimax errors. The matching redistribution bound also quantifies the deterioration when the movable fraction tends to one. It reaches the first-stage concentration scale once $(1-\delta)^2 JT$ is no larger than $T + J^2$.

Moving $\Delta \leq r_i - 1$ calls from stage i to stage $k > i$ increases R_b by Δ exactly for $i < b \leq k$. It increases W by $(k-i)\Delta$ and weakly increases H. A strict improvement in H requires one of the changed products to exceed the previous maximum. These facts about the bounds do not prove monotonicity of the minimax error or of training performance.

D FINITE-DESCRIPTION BOUNDS

The total-budget result formalizes the information bottleneck discussed in Section 7. We then ask the stronger question of whether the same information can be distributed across separate component codebooks. This componentwise refinement is not needed for the main storage–computation frontier.

D.1 A TOTAL DESCRIPTION BUDGET

A small number of real-valued parameters can contain a long code. The storage–computation frontier therefore leaves a further question: how much target information is needed to benefit from additional calls? Combining the main theorem with classical encoding entropy gives a direct answer (Yarotsky & Zhevnerchuk, 2020; Gühring & Raslan, 2021).

Suppose I bits specify the complete target-dependent parameter assignment, including the module, input/output and initialization values. For a fixed network design this permits at most 2^I assignments. A fixed map Γ selects the assignment from its bit string; its description and offline computation are unrestricted, and the selected parameters are evaluated exactly. A target-independent copying or padding rule can be incorporated into Γ without increasing I, even though each copied module still stores its own parameters. Write $\mathcal{E}_d^{\mathrm{FI}}(I, J, T)$ for the resulting minimax uniform error, with the graph and schedule chosen before the target; FI denotes finite information.

Corollary 17 (Total description budget)**.** *Under the same module-size bounds, for all integers $I \geq 1$ and $1 \leq J \leq T$,*

$$\mathcal{E}_d^{\mathrm{FI}}(I, J, T) \asymp_d [1 + \min\{I, JT\}]^{-1/d}. \tag{38}$$

Proof. Classical entropy bounds provide at most 2^I representative Lipschitz functions whose uniform distance to any target is at most $C_d(1+I)^{-1/d}$ (Yarotsky & Zhevnerchuk, 2020, Section 7). Apply

Theorem 1 to these representatives using one fixed graph and schedule, and let Γ return the parameters for the selected representative. The triangle inequality gives

$$\text{error} \le C_d(1+I)^{-1/d} + C_d(JT)^{-1/d} \le C'_d[1+\min\{I, JT\}]^{-1/d}.$$

For the lower bound, the at most 2^I assignments give a finite function class $\mathcal{H}$ with $|\mathcal{H}| \le 2^I$. Its sign VC dimension is at most I, since shattering N points requires 2^N distinct functions. Lemma 10 therefore gives error at least $\frac{1}{8}(1+I)^{-1/d}$. Combining this with the computation lower bound in Theorem 1 proves the result, including small budgets. □

When $I \ll JT$, additional calls cannot improve the best approximation order without more target information. When $JT \ll I$, storage or computation is the limiting resource. Figure 7 illustrates these two regimes.

This has a direct implication for parameter compression. If P counted parameters are each specified by at most b bits, the total description has at most bP bits, so

$$\text{worst-case uniform error } \ge c_d(1+bP)^{-1/d}. \tag{39}$$

To attain error $O_d((JT)^{-1/d})$, this inequality requires $bP \gtrsim_d JT$. Since the local size bounds give $P = O_d(J)$, a uniform per-parameter description limit must satisfy $b \gtrsim_d T$. This is a necessary bound on the allowed description length, not a claim that every weight carries that many bits. At $J \asymp T$ and $P \asymp_d J$, it recovers the requirement $b \gtrsim_d P$ for error $O_d(P^{-2/d})$.

The upper bound uses the classical encoding-entropy approach: Γ selects an exact parameter assignment from a finite codebook (Gühring & Raslan, 2021, Definition 3.1 and Observation 3.2). It therefore bounds the information needed to specify the network, but does not guarantee stability under finite-precision execution. A separate limit on each component requires distributing this information across modules; Section D.2 establishes the corresponding bounds.

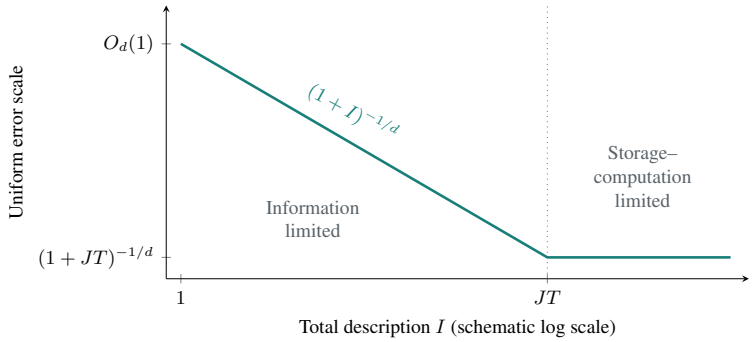


Figure 7: Description length limits the benefit of computation. At fixed J, T, the error scale decreases as $(1+I)^{-1/d}$ until the description budget reaches order JT, then stays at order $(1+JT)^{-1/d}$. The plot shows the two regimes of Corollary 17 on symbolic logarithmic axes, with unspecified multiplicative constants.

D.2 Componentwise description budgets

D.2.1 Component codebooks and approximation rate

Give every module, the input map including initialization, and the output map a fixed codebook of at most 2^ℓ entries. Each entry specifies the component's complete parameter vector. The codebooks are chosen for each budget before the target; no further target-dependent values or choices are allowed. Only the at most ℓ bits selecting each component's entry are charged; the fixed codebooks are not. Thus ℓ bounds target-dependent description length per component, not total implementation storage. Let $\mathcal{E}_d^{\mathrm{loc}}(\ell, J, T)$ denote the resulting minimax uniform error.

We use the decoder of Lemma 5 with $m \in 4\mathbb{N}$, choosing the grid sizes to fit each module's codebook. The logarithmic lower bound on ℓ below reserves enough bits for the initial values; the remaining bits encode fine-cell increments.

Theorem 18 (Description budget per component)**.** *There are constants $a_d, c_d, C_d, T_d > 0$ such that, for all integers $T \geq T_d$, $1 \leq J \leq T$, and $\ell \geq a_d \log_2(T+2)$,*

$$c_d[J \min\{\ell, T\}]^{-1/d} \leq \mathcal{E}_d^{\mathrm{loc}}(\ell, J, T) \leq C_d[J \min\{\ell, T\}]^{-1/d}. \tag{40}$$

The codebooks and designs obey the fixed module-size bounds of Section 3.

D.2.2 Lower bound: information and computation

Set $I = \ell(J+2)$. At most $J+2$ components select codes, so the resulting function class has at most 2^I members and sign VC dimension at most I. Lemma 10 gives error at least $\frac{1}{8}[1+\ell(J+2)]^{-1/d}$. For each fixed design and its codebooks, this finite output class is contained in the unrestricted parameter class of the same design. The parameter–runtime bound and grid reduction in Lemma 10 therefore give the additional lower bound $c_d(JT)^{-1/d}$, uniformly over the codebooks. Their maximum gives the lower half of (40). This argument remains valid for positive ℓ below the logarithmic threshold.

D.2.3 Upper bound: constructing the component codebooks

We choose each component's codebook before the target is known. Each loading module stores one code for each of the $Q = 2d+1$ shifted grids; the codebook will contain every valid tuple of such codes at the chosen resolution. Choose

$$n = \left\lfloor \min\{J, T/8\}^{1/d} \right\rfloor, \qquad m = 4 \left\lfloor \frac{[\min\{\ell, T\}/(16Q)]^{1/d}}{4} \right\rfloor. \tag{41}$$

Write $B = n^d$, $F = m^d$, $M = nm$ and $L = \lceil \log_3(2M+1) \rceil$. Choose the dimension-dependent constants a_d, T_d large enough that $\min\{\ell, T\}/(16Q) \geq 4^d$ throughout the stated range. Then $n \geq 1$ and $m \in 4\mathbb{N}$, so the main decoder applies directly. Each of the Q codes in a module uses L initial-value digits and $F-1$ increment digits. Every digit belongs to $\{0,1,2\}$ and can be represented by two bits. Since $M^d = BF \leq T^2$, we have $L = O_d(\log(T+2))$. Increasing a_d if necessary gives $2QL \leq \ell/2$; also $2QF \leq \ell/8$. Thus the complete description uses at most $2Q(L+F-1) \leq \ell$ bits.

As in the proof of Lemma 8, $L \leq n + m \leq B + F$. The calls therefore fit:

$$B + L + F \leq 2(B+F) \leq T/4 + T/(8Q) < T.$$

The inequality $\lfloor v \rfloor \geq v/2$ for $v \geq 1$ gives

$$BF \geq 4^{-d} \min\{J, T/8\} \frac{\min\{\ell, T\}}{16Q} \geq \frac{J \min\{\ell, T\}}{128Q\, 4^d}.$$

The chosen fixed codebooks therefore contain every valid tuple of codes at this resolution; the input and output maps need no target bits. The shared spatial features and counter coefficients are independent of the target, so they do not enlarge these codebooks. Applying Lemma 5 and the same-topology splitting and identity padding in Lemma 7 gives exactly J stages. A copied module receives the original codebook and an identity module a singleton codebook. This proves the upper bound under the same module-size bounds.

D.2.4 Range and interpretation

Below the logarithmic threshold, only the lower bound is asserted here. Within the matching regime, for a fixed error multiplier $A > 0$, error at most $A(JT)^{-1/d}$ forces $\ell \gtrsim_{d,A} T$ by the local information lower bound (for sufficiently large T depending on d, A). Conversely, $\ell \geq C_d T$ suffices for error $O_d((JT)^{-1/d})$ by the upper bound.

Within the theorem's range, increasing the call budget beyond the scale of ℓ leaves the order limited by $J\ell$; retaining the unrestricted-precision frontier requires longer component descriptions. The upper bound distributes the information while respecting each component's codebook. A global table of $2^{J\ell}$ complete networks need not meet those individual limits. Here ℓ describes a whole component, not each scalar weight. Nor does the formula prescribe $r_j \asymp \ell$ for each stage: the call allocation can be chosen, and loading and decoding can occur in different stages. Combining unequal component descriptions with prescribed loop allocations would require a further result. The codebook constraint by itself also supplies no guarantee for finite-precision arithmetic during execution.