THE SURPRISING EXPRESSIVITY OF A SINGLE NEURON WITH AUTOREGRESSIVE CHAIN-OF-THOUGHT

Anonymous authors
Paper under double-blind review

ABSTRACT

The success of autoregressive language models trained with Chain-of-Thought (CoT) supervision raises a fundamental question: How simple can a next-token predictor be while still performing general computation through its autoregressively generated CoT? Surprisingly, we show that a single neuron suffices. For a broad family of activations, including threshold and ReLU, a single autoregressive neuron can simulate any feedforward circuit with the same activation. Specifically, a size-s circuit on n inputs can be simulated by a neuron of dimension $O(ns + s^2)$ using $O(s^2)$ generation steps, with both bounds independent of circuit depth and tight under natural assumptions. For threshold activations, this improves upon the simulation of Joshi et al. (COLT 2025), whose bounds scale exponentially with circuit depth. Consequently, autoregressive threshold neurons can simulate polynomial-size circuits and hence polynomial-time Turing machines, while remaining efficiently learnable from CoT supervision. Our main expressivity result relies on a novel connection to Sidon sets from additive combinatorics.

1 INTRODUCTION

Autoregressive (AR) generation has emerged as one of the most successful paradigms in modern machine learning and underlies contemporary large language models (LLMs). At each generation step, the same parameterized next-token predictor is applied to the current context. This parameter sharing makes the model *time-invariant*, while repeated application allows a fixed predictor to perform computations of varying length. On multi-step reasoning tasks, the generated sequence can be viewed as a Chain-of-Thought (CoT) that guides the LLM toward the correct final answer (Wei et al., 2022c; Kojima et al., 2022). From a theoretical perspective, this raises a basic expressivity question:

Question 1 (Expressivity)**.** How simple can a class of next-token predictors be while still enabling efficient autoregressive simulation of general computation,[1] such as that performed by Turing machines or deep neural networks?

Chains of thought are relevant not only at inference time but also during training. Pretraining corpora often contain worked solutions, mathematical derivations, and other step-by-step explanations. During post-training, and particularly during supervised fine-tuning (SFT), reasoning traces are explicitly incorporated. These traces may consist of human-written step-by-step solutions or synthetic reasoning generated by a more capable teacher model. Thus, reasoning trajectories constitute a natural form of supervision: each trajectory provides a sequence of prefix-next-token examples that are used to learn a single next-token predictor. This raises the following learnability question:

Question 2 (Learnability)**.** Can repeated application of a simple predictor simulate general computation, while remaining efficiently learnable from CoT supervision?

Expressivity limits cannot be overcome by better supervision, while expressivity alone says little about whether the model can be learned efficiently from data.

Prior work and the gap addressed by this work. In practice, the most widely used class of next-token predictor is that of transformers (Vaswani et al., 2017). Consequently, a growing line

[1]Here, "efficient" means that both the number of parameters of the next-token predictor and the number of generation steps are polynomial in the input length and in the size (or runtime) of the simulated computation.

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.

of work has established strong expressivity results for transformers augmented with AR chain-of-thought generation (Li et al., 2024; Merrill & Sabharwal, 2024; Brösamle & Eckstein, 2026). These results, however, do not identify the simplest next-token operations that suffice to simulate general computation. Moreover, we focus on classes that admit provable polynomial-time learning guarantees for the next-token prediction problem.

Malach (2023) argued that much of the power of autoregressive models comes from CoT computation rather than the specific architecture. He supported this empirically on simple benchmarks and theoretically for AR linear-threshold models,[2] with one significant caveat: his construction allows a *different* linear-threshold predictor (neuron) for each generation step. This departs from practical autoregressive architectures and makes the simulation straightforward by allowing parameters to be assigned separately to each step.

Joshi et al. (2025) subsequently formalized the *time-invariant* setting, in which a single predictor is reused across all generation steps, emphasizing its naturality and the statistical benefits of parameter sharing. These benefits include sample-complexity bounds independent of the generation length (Hanneke et al., 2026; Li, 2026). However, parameter sharing complicates even basic expressivity questions. Indeed, the construction of Joshi et al. (2025) for simulating threshold circuits with an AR linear-threshold predictor incurs an exponential dependence on circuit depth, requiring both dimension and generation length $O(\mathrm{size}^{\mathrm{depth}})$. They explicitly leave open whether their simulation can be made more efficient. More broadly, it remains open whether a simple class of next-token predictors can efficiently simulate general computation while admitting efficient learning guarantees in the time-invariant setting.

Our results. We answer both questions above by showing that repeated application of a *single neuron* suffices. We state our main result using σ-circuits, which provide a common language for discrete and real-valued computation. A σ-circuit is simply a feedforward computation graph whose nodes apply an activation σ to affine combinations of earlier values. For the threshold activation on Boolean inputs, these circuits (namely, *threshold circuits*) capture Boolean computation and can simulate any bounded-time Turing-machine computation. More generally, σ-circuits are feedforward neural networks with activation function σ. Our main theorem applies to a broad family of activations:

Theorem 1 (Informal version of Theorems 2 and 3, Section 3)**.** *Under mild assumptions on the activation σ (satisfied by threshold, ReLU, GELU, and sigmoid, among others), any size-s σ-circuit on inputs of dimension n can be simulated by autoregressively applying a single fixed neuron $\boldsymbol{x} \mapsto \sigma(\langle \boldsymbol{w}, \boldsymbol{x} \rangle)$,[3] with $\boldsymbol{w} \in \mathbb{R}^d$. The dimension of the AR neuron is $d = O(ns + s^2)$ and the number of generation steps is $T = O(s^2)$, both completely independent of the circuit depth.*

The simulation implies the following expressivity and learning guarantees. We also study the optimality of the simulation bounds and provide an experimental illustration:

- **Expressivity (Section 3).** For threshold activations, our simulation improves on Joshi et al. (2025) by replacing their exponential dependence on circuit depth with depth-independent polynomial bounds. It implies that a single time-invariant AR neuron can simulate any Turing machine, with neuron dimension and generation length polynomial in the machine's running time and input length (Corollary 1). Our result also captures real-valued computation for a broad class of nonlinear activations: a single shared neuron can efficiently simulate any size-s feedforward network with the same activation on $[0, 1]^n$. The simulation is exact for ReLU and arbitrarily accurate for activations such as sigmoid and GeLU (Theorem 3). Together, these simulations answer Question 1 for both Boolean and real-valued computation.
- **Learnability (Section 4).** Combining our simulation with known learning guarantees for AR linear thresholds shows that a single autoregressive threshold neuron can be efficiently learned from complete CoT trajectories to predict the outputs of any polynomial-time Turing machine (Corollary 2). For ReLU, combining our simulation with known generalization bounds for neural networks yields analogous learning guarantees for feedforward ReLU networks (Corollary 3). Together, these results answer Question 2. Our result also implies that

[2] The paper formulates these models using argmax decoding over linear scores, which can equivalently be represented by a single linear-threshold gate.

[3] For vectors of different dimensions, the inner product is defined by padding the shorter vector with zeros on the left.

learning from final answers alone is computationally hard under the standard cryptographic assumption that one-way functions exist (Corollary 4). This assumption is weaker than those required in previous work (Joshi et al., 2025; 2026), strengthening the computational advantage of learning with CoT.

- **Optimality of the simulation bounds (Section 5).** The dimension bound $d = O(ns + s^2)$ matches the worst-case number of circuit weights. For generation length, we introduce the notion of *direct edge alignment* (Definition 2), which formalizes a natural structural property shared by our construction and prior simulations. Under this restriction, we prove a matching worst-case lower bound of $T = \Omega(s^2)$ (Theorem 4).
- **Experiments.** We use a CNN teacher with nontrivial test accuracy on CIFAR-10 (Krizhevsky, 2009) and train a single shared autoregressive ReLU neuron on its activation traces, without access to the teacher's weights. The learned AR neuron agrees with the teacher's predictions on approximately 99% of test examples.

The proof of Theorem 1 is based on a novel connection between autoregressive circuit simulation and Sidon sets from additive combinatorics. Using the classical construction of Erdos & Turán (1941), we position the circuit weights according to a Sidon set within the parameter vector of a single neuron. This careful placement prevents weights from colliding during the autoregressive process, despite parameter sharing. The resulting collision-free encoding is the key new ingredient, yielding a depth-independent simulation, and offering a simpler yet substantially more powerful alternative to the recursive construction of Joshi et al. (2025).

2 PRELIMINARIES

Notation. We use boldface letters for vectors and write $[n] := \{1, 2, \ldots, n\}$. For a vector $\boldsymbol{v} \in \mathbb{R}^m$ and $i \in [m]$, we write $\boldsymbol{v}[i]$ for its i-th coordinate and $\boldsymbol{v}[-i] := \boldsymbol{v}[m - i + 1]$ for its i-th coordinate from the right. For real numbers a, b, let $a \vee b := \max\{a, b\}$ and $a \wedge b := \min\{a, b\}$. We use standard asymptotic notation, with $\Theta(\cdot)$, $\Omega(\cdot)$, and $O(\cdot)$ having universal implicit constants, and $\tilde{\Theta}(\cdot)$, $\tilde{\Omega}(\cdot)$, and $\tilde{O}(\cdot)$ additionally suppress factors that are polylogarithmic in the problem parameters. We write $\mathbb{1}[P]$ for the indicator of a predicate P.

Next, we define the autoregressive single-neuron model, the circuits it simulates, and the learning setting with complete CoT supervision.

Time-invariant Autoregressive Generation. Let Σ be a token alphabet and let $\mathcal{F} \subseteq \Sigma^{\Sigma^*}$ be a class of next-token predictors, called the *base class*. We use $\Sigma = \{0, 1\}$ for Boolean computation and $\Sigma = \mathbb{R}$ for real-valued computation. For an input $\boldsymbol{x} \in \Sigma^n$, starting from the initial context $\boldsymbol{x}^{(0)} = \boldsymbol{x}$, autoregressive generation repeatedly applies the same predictor $f \in \mathcal{F}$ and appends its output to the context. We write $f^{\mathsf{e2e\text{-}}t}(\boldsymbol{x})$ for the output token generated at the t-th autoregressive step:

$$f^{\mathsf{e2e\text{-}}t}(\boldsymbol{x}) = f(\boldsymbol{x}^{(t-1)}), \qquad \boldsymbol{x}^{(t)} = (\boldsymbol{x}^{(t-1)}, f^{\mathsf{e2e\text{-}}t}(\boldsymbol{x})).$$

We denote the resulting chain of thought by $f^{\mathsf{CoT}\text{-}T}(\boldsymbol{x}) = (f^{\mathsf{e2e\text{-}}1}(\boldsymbol{x}), \ldots, f^{\mathsf{e2e\text{-}}T}(\boldsymbol{x})) \in \Sigma^T$, consisting of the T generated tokens. Reusing the same predictor at every step makes the model *time-invariant*.

Single-neuron predictors. Our base class consists of single neurons with a fixed activation $\sigma : \mathbb{R} \to \Sigma$, where $\Sigma \subseteq \mathbb{R}$. For $\boldsymbol{x} \in \Sigma^n$ and $\boldsymbol{w} \in \mathbb{R}^d$, we use the right-aligned inner product $\langle \boldsymbol{w}, \boldsymbol{x} \rangle := \sum_{i=1}^{d \wedge n} \boldsymbol{w}[-i]\boldsymbol{x}[-i]$, equivalently padding the shorter vector with zeros on the left. This convention aligns each new CoT value with the shared parameter vector. For $b \in \mathbb{R}$, define the next-token predictor $f_{\boldsymbol{w},b,\sigma} : \Sigma^* \to \Sigma$ by $f_{\boldsymbol{w},b,\sigma}(\boldsymbol{x}) := \sigma(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)$. The resulting dimension-d base class is

$$\mathcal{F}_{d,\mathrm{lin},\sigma} := \left\{ f_{\boldsymbol{w},b,\sigma} : \boldsymbol{w} \in \mathbb{R}^d,\ b \in \mathbb{R} \right\}.$$

Thus, $\mathcal{F}_{d,\mathrm{lin},\sigma}$ is a class of single neurons with activation σ, or equivalently generalized linear models with link function σ. When $b = 0$, we write $f_{\boldsymbol{w},\sigma} := f_{\boldsymbol{w},0,\sigma}$. When σ is clear from context, we omit it from the notation and write $\mathcal{F}_{d,\mathrm{lin}}$, $f_{\boldsymbol{w},b}$, and $f_{\boldsymbol{w}}$.

Efficient Learning with CoT. We study learning these AR predictors from complete CoT supervision (Joshi et al., 2025). Let $\mathcal{F}_d$ be a base class of next-token predictors, where d measures predictor size. Given target generator $f_\star \in \mathcal{F}_d$, each training example reveals the input $\boldsymbol{x}$ and the entire trajectory $f_\star^{\mathsf{CoT}\text{-}T}(\boldsymbol{x})$. The learner is evaluated on the final output $f_\star^{\mathsf{e2e}\text{-}T}(\boldsymbol{x})$.[4] Here, we define CoT learnability in the Boolean setting $\Sigma = \{0,1\}$. Learning setting for real-valued models are given in Section D.1.

Definition 1 (Realizable CoT learnability)**.** *We say that $\mathcal{F}_d^{\mathsf{e2e}\text{-}T}$ is CoT-learnable over $\{0,1\}^n$ with generation length T and sample complexity $m(\varepsilon,\delta)$ if there is a learning rule A such that the following holds. For every distribution $\mathcal{D}$ over $\{0,1\}^n$, target generator $f_\star \in \mathcal{F}_d$, and $\varepsilon,\delta \in (0,1)$, let*

$$S_{\mathrm{CoT}} = \big((\boldsymbol{x}_1, f_\star^{\mathsf{CoT}\text{-}T}(\boldsymbol{x}_1)), \ldots, (\boldsymbol{x}_m, f_\star^{\mathsf{CoT}\text{-}T}(\boldsymbol{x}_m))\big), \qquad \boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \overset{\text{i.i.d.}}{\sim} \mathcal{D},$$

where $m = m(\varepsilon,\delta)$. Then, with probability at least $1-\delta$ over the sample and the randomness of A, the predictor $h = A(S_{\mathrm{CoT}}) : \{0,1\}^n \to \{0,1\}$ satisfies

$$\Pr_{\boldsymbol{x}\sim\mathcal{D}}\big[h(\boldsymbol{x}) \neq f_\star^{\mathsf{e2e}\text{-}T}(\boldsymbol{x})\big] \leq \varepsilon.$$

We call it efficient if its sample complexity and running time are polynomial in $n, d, T, 1/\varepsilon$, and $\log(1/\delta)$.

σ-circuits. We next define the circuits that serve as simulation targets. A size-s, depth-L σ-circuit $C : \Sigma^n \to \Sigma$ is a weighted directed acyclic graph $G = (V, E)$ with input nodes $V_{\text{in}} = \{v_{\text{in},1}, \ldots, v_{\text{in},n}\}$ at depth 0 and non-input nodes $\{v_1, \ldots, v_s\}$. The non-input nodes are partitioned into layers $V^{(1)}, \ldots, V^{(L-1)}, \{v_s\}$, where $V^{(\ell)}$ contains the hidden nodes at depth ℓ and v_s is the output node at depth L. Edges point from smaller to larger depth, with weights $w : E \to \mathbb{R}$.

For input $\boldsymbol{x} = (x_1, \ldots, x_n) \in \Sigma^n$, each node $v \in V$ has a value $\mathrm{val}(v) \in \Sigma$. Input nodes have values $\mathrm{val}(v_{\text{in},k}) = x_k$, and each non-input node v_i has a bias $\theta_i \in \mathbb{R}$ and computes

$$\mathrm{val}(v_i) = \sigma\left(\theta_i + \sum_{(u,v_i)\in E} w(u,v_i)\,\mathrm{val}(u)\right).$$

The output is $C(\boldsymbol{x}) = \mathrm{val}(v_s)$, and we write $C_i : \Sigma^n \to \Sigma$ for the function $C_i(\boldsymbol{x}) := \mathrm{val}(v_i)$ computed at node v_i. The circuit's *size* is its number of non-input nodes, $|V \setminus V_{\text{in}}| = s$. We call the circuit *layered* if every edge connects consecutive layers.

For threshold activation $\sigma(z) = \mathbf{1}\{z \geq 0\}$ and $\Sigma = \{0,1\}$, these are *threshold circuits* (Arora et al., 2009). For real-valued activations, these circuits are feedforward neural networks with activation σ, including MLPs and CNNs with or without skip connections. In this setting, we also consider inputs restricted to $[0,1]^n$, while retaining node and output values in $\mathbb{R}$.

3 Expressivity of Autoregressive Single Neuron

In this section, we present our main result: for a broad class of activations σ, a single autoregressive neuron can simulate a σ-circuit using the same activation. For size-s circuits on n inputs, our simulations use dimension $d = O(ns + s^2)$ and generation length $T = O(s^2)$, after applying an input feature map that depends only on n and s (see Remark 1). We begin in Section 3.1 with the discrete token space $\Sigma = \{0,1\}$ and threshold activations, showing how to simulate boolean threshold circuits and Turing machines. In Section 3.2, we consider general activations with real-valued computations, showing that a single neuron can simulate general feed-forward neural networks with the same activation. Finally, Section 3.3 outlines how Sidon sets enable efficient serialization of circuit computations.

[4] Our positive results also hold under the stronger evaluation criterion that requires the learner to match the entire CoT trajectory.

3.1 AUTOREGRESSIVE LINEAR THRESHOLDS SIMULATE BOOLEAN THRESHOLD CIRCUITS

We begin with the threshold activation $\sigma(z) = \mathbf{1}\{z \geq 0\}$, which yields the autoregressive linear-threshold class studied in prior work (Joshi et al., 2025; 2026; Altabaa et al., 2025; Hanneke et al., 2026; Doron-Arad et al., 2026b). This is arguably the simplest autoregressive model over the binary token space $\Sigma = \{0, 1\}$: taking the logits to be the linear scores $(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)/2$ and $-(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)/2$ for the two tokens 1 and 0, respectively, yields a logistic next-token distribution after applying Softmax. Then, greedy decoding is exactly the linear threshold rule $\mathbf{1}\{\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b \geq 0\}$. Below is our main expressivity result, stated for the binary token space $\Sigma = \{0, 1\}$:

Theorem 2. *Consider $\sigma(z) = \mathbf{1}\{z \geq 0\}$, and the class $\mathcal{F}_{d,\mathrm{lin},\sigma}$ over the token space $\Sigma = \{0, 1\}$. For every input size n and circuit size s, there is a map $\phi_{n,s} : \{0,1\}^n \to \{0,1\}^{n'}$ for $n' = O(ns + s^2)$, with the following property. Every size-s Boolean threshold circuit $C : \{0,1\}^n \to \{0,1\}$ can be simulated by a single autoregressive linear threshold neuron $f_{\boldsymbol{w},\sigma}$ with $\boldsymbol{w} \in \mathbb{R}^d$, where $d = O(ns + s^2)$, in $T = O(s^2)$ generation steps:*

$$f^{\text{e2e-}T}_{\boldsymbol{w},\sigma}(\phi_{n,s}(\boldsymbol{x})) = C(\boldsymbol{x}) \qquad \textit{for every } \boldsymbol{x} \in \{0,1\}^n.$$

Moreover, $\phi_{n,s}$ depends only on n and s and is computable in time polynomial in n and s.

Remark 1 (A common feature map)**.** For any fixed input size n and circuit-size bound s, the map $\phi_{n,s}$ applies simultaneously to all threshold circuits of size at most s; only the parameter vector $\boldsymbol{w}$ depends on the circuit C. Indeed, smaller circuits can be padded to size s by inserting dummy gates that output zero before the output gate, without changing the computed function. Such feature maps also appear in prior simulation results for specific autoregressive neurons (Joshi et al., 2025; 2026; Doron-Arad et al., 2026a).

Combining Theorem 2 with the standard simulation of polynomial-time Turing machines by polynomial-size threshold circuits (Pippenger & Fischer, 1979; Arora et al., 2009), yields the following universality result:

Corollary 1 (Turing machine simulation)**.** *Let $\tau(n) = \mathrm{poly}(n)$ be a running-time bound. There exists a family of $\mathrm{poly}(n)$-time computable feature maps $\phi_n : \{0,1\}^n \to \{0,1\}^{n'}$, where $n' = \tilde{O}(n\tau(n) + \tau(n)^2)$, such that the following holds. For every deterministic Turing machine M running in time at most $\tau(n)$ and every sufficiently large n,[5] there exists a linear threshold $f_{\boldsymbol{w}}$ with $\boldsymbol{w} \in \mathbb{R}^d$ such that*

$$f^{\text{e2e-}T}_{\boldsymbol{w}}(\phi(\boldsymbol{x})) = M(\boldsymbol{x}) \qquad \textit{for every } \boldsymbol{x} \in \{0,1\}^n,$$

where $d = \tilde{O}(n\tau(n) + \tau(n)^2)$ and $T = \tilde{O}(\tau(n)^2)$.

Moreover, Theorem 2 replaces the bounds $d, T = O(s^L n)$ of Joshi et al. (2025, Lemma 4.5), where L is the circuit depth, with $d = O(ns + s^2)$ and $T = O(s^2)$. This removes the exponential dependence on circuit depth, enabling efficient simulation of polynomial-time Turing machines.

3.2 NEURAL NETWORKS CAN BE SIMULATED BY A SINGLE AUTOREGRESSIVE NEURON

More broadly, our expressivity result is not specific to threshold activations: it applies to a broad class of standard nonlinear activations. Here we take $\Sigma = \mathbb{R}$. In this case, σ-circuits correspond to standard feedforward neural networks. To state the results formally, we first specify the assumptions on the activation σ.

Assumption 1. *There exists $c_\sigma \in \mathbb{R}$ such that $\sigma(z) = 0$ for every $z \leq c_\sigma$ (the zero-region condition). Moreover, σ is bounded on every bounded interval.*

The zero-region condition is satisfied by activations such as threshold and ReLU. Many other common activations, including sigmoid, GELU, and SiLU, do not become exactly zero below any finite threshold, but instead converge to zero as their input tends to $-\infty$. For such activations, we obtain an approximate version of our expressivity result. In particular, the zero-region condition in Assumption 1 can be replaced by a decay to zero at $-\infty$, together with a mild regularity condition.

[5] The required lower bound on n may depend on M, but the feature maps depend only on n and τ. An extra logarithmic factor in the common circuit-size bound absorbs machine-dependent simulation constants; see Remark 1 for the padding argument.

Assumption 2. *The activation $\sigma : \mathbb{R} \to \mathbb{R}$ is locally Lipschitz (Definition 3) and satisfies $\lim_{z\to-\infty} \sigma(z) = 0$.*

Below, we state our exact and approximate expressivity result, under Assumptions 1 and 2 respectively, for simulating general feed-forward neural networks with the same activation and over a bounded domain.

Theorem 3. *For every input dimension n and circuit size s, there exist a feature map $\phi_{n,s} : [0,1]^n \to [0,1]^{n'}$ and integers d, T, with $n', d = O(ns + s^2)$ and $T = O(s^2)$, such that the following holds for every size-s σ-circuit $C : [0,1]^n \to \mathbb{R}$:*

1. ***Exact simulation.*** *If σ satisfies Assumption 1, there exists $\boldsymbol{w} \in \mathbb{R}^d$ such that*
$$f^{\text{e2e-}T}_{\boldsymbol{w},\sigma}(\phi_{n,s}(\boldsymbol{x})) = C(\boldsymbol{x}) \qquad \text{for every } \boldsymbol{x} \in [0,1]^n.$$
2. ***Approximate simulation.*** *If σ satisfies Assumption 2, then for every $\varepsilon > 0$, there exists $\boldsymbol{w} \in \mathbb{R}^d$ such that*
$$\left| f^{\text{e2e-}T}_{\boldsymbol{w},\sigma}(\phi_{n,s}(\boldsymbol{x})) - C(\boldsymbol{x}) \right| \le \varepsilon \qquad \text{for every } \boldsymbol{x} \in [0,1]^n.$$

Moreover, $\phi_{n,s}$ depends only on n and s and is computable in time polynomial in n and s.[6]

Our exact simulations also preserve the circuit's intermediate gate values in the generated trajectory, as formalized by the following property.

Property 1 (CoT preservation)**.** *The exact simulation in Theorem 3 can be chosen so that there exist designated steps $1 \le t_1 < \cdots < t_s = T$, depending only on n, s, such that*

$$f^{\text{e2e-}t}_{\boldsymbol{w},\sigma}(\phi_{n,s}(\boldsymbol{x})) = \begin{cases} C_i(\boldsymbol{x}), & \text{if } t = t_i \text{ for some } i \in [s], \\ 0, & \text{otherwise}, \end{cases} \qquad \text{for every } \boldsymbol{x} \in [0,1]^n \text{ and } t \in [T].$$

The designated steps are computable in time polynomial in n, s. The same property holds for the Boolean simulation in Theorem 2, with inputs restricted to $\{0,1\}^n$.

3.3 Proof technique: serialization via Sidon sets

The main technical contribution is a new simulation approach: we serialize the circuit computation into an autoregressive sequence and use an additive-combinatorial structure to prevent different circuit weights from colliding within a single shared parameter vector.

Each non-input circuit node v_i, for $i \in [s]$, is assigned a generation step $t_i \in [T]$, at which the value $C_i(\boldsymbol{x})$ is produced, i.e. $f^{\text{e2e-}t_i}_{\boldsymbol{w},\sigma}(\phi_{n,s}(\boldsymbol{x})) = C_i(\boldsymbol{x})$. At any later step t_j, where $j > i$, this value is the $(t_j - t_i)$-th most recent value in the autoregressive context and is therefore aligned with the coordinate $\boldsymbol{w}[-(t_j - t_i)]$, where $\boldsymbol{w}$ is the shared parameter vector. To simulate a circuit edge $(v_i, v_j) \in E$ with weight $w(v_i, v_j)$, we therefore set $\boldsymbol{w}[-(t_j - t_i)] = w(v_i, v_j)$, so that $C_i(\boldsymbol{x})$ contributes $w(v_i, v_j)C_i(\boldsymbol{x})$ to the preactivation at step t_j, matching the corresponding term in the definition of a σ-circuit.

The difficulty is that each coordinate of $\boldsymbol{w}$ is reused at every generation step. Consequently, if $t_j - t_i = t_\ell - t_k$, for two distinct pairs (i, j) and (k, ℓ), then two potentially different circuit weights would have to be encoded by the same coordinate of $\boldsymbol{w}$. Similarly, aligning input-to-gate weights requires distinct sums $t_i + t'_q$, where t'_q denotes the initial position assigned to the corresponding input value, counted from the right. Sidon sets provide the required collision-free assignment. Formally, a set $S \subseteq \mathbb{N}$ is a *Sidon set* if

$$a + b = c + d \quad \Longrightarrow \quad \{a, b\} = \{c, d\} \qquad \text{for all } a, b, c, d \in S.$$

Equivalently, its positive pairwise *differences* are also distinct (Observation 3). Using the classical construction of Erdos & Turán (1941), we obtain such a set of size s whose largest element is $O(s^2)$. The precise formulation we use is stated and proved in Claim 1. Then we use this set to choose the generation steps $t_1, \ldots, t_s$ and to construct the feature map $\phi_{n,s}$. The formal proof completes this picture by verifying that the values generated at undesignated time steps do not interfere with the simulation. The main novelty lies in viewing the proof through this collision-free serialization framework.

[6] Here, we use the real-RAM model, in which reading, writing, and copying a real number take unit time.

4 IMPLICATIONS FOR LEARNING

Efficient Learning with CoT. We have shown that a single AR neuron can efficiently simulate general computations such as Turing machines and feedforward neural networks. The CoT in our simulations is the neuron's complete autoregressive trajectory: the intermediate tokens perform the simulated computation (by Property 1), and the final token is the output of the simulated machine or network. Suppose that these complete trajectories are provided during training. For a fixed activation σ, each trajectory generated by a target neuron $f_\star \in \mathcal{F}_{d,\mathrm{lin},\sigma}$ provides labeled prefix–next-token examples for the same shared predictor. Consequently, it suffices to find a neuron in $\mathcal{F}_{d,\mathrm{lin},\sigma}$ that predicts the labeled next token for each prefix exactly or approximately. For some activation functions, such a neuron can be found efficiently. Combined with an appropriate generalization bound, this computational guarantee yields efficient learnability from CoT supervision (Definition 1).

For the threshold activation, linear programming can efficiently find parameters consistent with all observed prefix–next-token pairs. Combining Corollary 1 with known generalization bounds for autoregressive linear thresholds (e.g.Corollary 5) yields the following:

Corollary 2. *Let M be a polynomial-time Turing machine, and let $\mathcal{D}$ be any distribution over $\{0,1\}^n$. Given a sufficiently large polynomial number of complete CoT trajectories of length $T = \mathrm{poly}(n)$ encoding the computations of M on i.i.d. inputs drawn from $\mathcal{D}$, one can efficiently train a single linear-threshold neuron $f_{\boldsymbol{w}}$ to predict $M(\boldsymbol{x})$ with high probability over a fresh $\boldsymbol{x} \sim \mathcal{D}$.*

Similarly, a single AR ReLU neuron can simulate an entire, potentially large and deep, feedforward ReLU network (Theorem 3). With complete activation supervision, fitting this shared neuron reduces to linear programming as well.[7] Combined with generalization bounds for ReLU Neural Networks (Bartlett et al., 2019) yields the following:

Corollary 3 (Informal version of Corollary 6 and remark 3)**.** *For any distribution over $[0,1]^n$, complete activation supervision from a size-s ReLU network C suffices to train a single autoregressive ReLU neuron to predict $\mathbb{1}[C(\boldsymbol{x}) > 0]$ (or approximate $C(\boldsymbol{x})$ under a suitable bounded regression loss) with high probability on a fresh input. The neuron's dimension, generation length, sample complexity, and training time are polynomial in the relevant parameters.*

In both cases, gradient-based methods offer an alternative to linear programming for fitting the observed prefix–next-token pairs. Penalizing squared violations of the linear feasibility constraints yields a convex, smooth loss whose minimum value is zero (under realizability); for ReLU, this gives the per-example loss in Equation (4) used in our experiments. Gradient descent with a suitable step size can therefore make this loss arbitrarily small.

Hardness of E2E Learning. We return to linear-threshold models with $\Sigma = \{0,1\}$ and consider learning without access to the intermediate trajectory. Under End-to-End (E2E) supervision, a training example reveals only the input $\boldsymbol{x}$ and the final output $f_\star^{\mathsf{e2e}\text{-}T}(\boldsymbol{x})$ of a target generator $f_\star \in \mathcal{F}_{d,\mathrm{lin}}$. The learner is evaluated on the same final output, but, unlike under CoT supervision, all intermediate values remain latent during training. Formal definitions are given in Section A.1.

Joshi et al. (2025) showed that E2E learning of autoregressive linear thresholds is computationally hard under a hardness assumption on learning constant-depth threshold circuits. Their reduction simulates a size-s, depth-L threshold circuit using dimension and generation length of order s^L. Thus, the reduction has polynomial overhead only for constant L, which limits the computational assumptions from which it can derive hardness.

Our simulation removes this limitation. Every polynomial-time computation is simulated with polynomial dimension and generation length. Combining this simulation with the existence of one-way functions yields E2E hardness under one of the most common cryptographic assumptions.

Corollary 4 (Informal version of Corollary 7 and remark 3)**.** *If one-way functions exist, then no polynomial-time algorithm learns autoregressive linear-threshold with E2E supervision.*

Indeed, the existence of one-way functions can be viewed as a minimal assumption for computational cryptography, already sufficient for fundamental primitives such as pseudorandom generators, pseudorandom functions, private-key encryption and digital signatures (Goldreich, 2007).

[7] For a fixed prefix $\boldsymbol{z}$ and target activation y, consistency here requires $\langle \boldsymbol{w}, \boldsymbol{z} \rangle + b = y$ when $y > 0$ and $\langle \boldsymbol{w}, \boldsymbol{z} \rangle + b \leq 0$ when $y = 0$.

5 Optimality of the simulation bounds on T and d

In this section, we discuss the optimality of the bounds of the generation length T and dimension d in Theorem 3. We show that the bounds are tight up to constant factors, under some reasonable assumptions on the construction. Joshi et al. (2026) showed simulation of depth-2 threshold circuit with $T = O(s)$ and $d = O(ns)$. This motivates the question of whether we can improve the bounds when the circuit has additional structure, such as bounded depth or width. We show that even then, we cannot expect to improve those bounds in some cases.

Dimension d. Since the simulation should be able to represent every circuit, it is reasonable to expect that all the weights of the circuit should be encoded in the autoregressive model, that is in the parameter vector $\boldsymbol{w}$.[8] A size-s circuit has at most ns edges from input nodes and at most $\binom{s}{2}$ edges between non-input nodes. Therefore, the bound of $d = O(ns + s^2)$ matches the number of circuit weights in the worst case. Even for constant depth circuits, the number of edges between non-input nodes can be $\Theta(s^2)$. Indeed, a depth-3 circuit can have two consecutive layers of $\Theta(s)$ non-input nodes with all edges between them, and hence $\Theta(s^2)$ internal edges.

Generation length T. We next show that a quadratic generation length is necessary under a natural restriction on how the model simulates the circuit: Each non-input node value $C_i(\boldsymbol{x})$ must be generated at a designated step (as in Property 1), and for every edge between non-input nodes, the model must use the generated value of the source node directly when computing the target node, with an autoregressive coefficient equal to the edge weight. We formalize this restriction below.

Definition 2 (Direct edge alignment)**.** *Fix $\Sigma = \mathbb{R}$ and an activation $\sigma : \mathbb{R} \to \mathbb{R}$. Consider a simulation scheme that simulates every σ-circuit C by an autoregressive neuron $f_{\boldsymbol{w},\sigma}$ using any feature map ϕ. The scheme is* directly edge-aligned *if, for every circuit with non-input nodes $v_1, \ldots, v_s$, there are designated steps $1 \le t_1 < \cdots < t_s \le T$ such that:*

1. *$f^{\text{e2e-}t_i}_{\boldsymbol{w},\sigma}(\phi(\boldsymbol{x})) = C_i(\boldsymbol{x})$ for every $i \in [s]$ and $\boldsymbol{x} \in [0,1]^n$.*
2. *For every edge (v_j, v_i) between non-input nodes, the value generated at step t_j, namely $C_j(\boldsymbol{x})$, is used at step t_i with coefficient $w(v_j, v_i)$.*[9]

For threshold circuits on Boolean inputs, the same definition applies with $\Sigma = \{0,1\}$ and $\boldsymbol{x} \in \{0,1\}^n$. The construction in the proof of Theorem 3 is directly edge-aligned, as well as all the previous constructions (Malach, 2023; Joshi et al., 2025; 2026). Direct edge alignment is a natural restriction, since it allows the autoregressive model to simulate the circuit in a straightforward manner. Under this restriction, the quadratic dependence on the circuit depth is necessary, even for width-1 circuits:

Theorem 4. *For every $L \ge 1$, any directly edge-aligned simulation (Definition 2) for arbitrary σ-circuits has generation length $T = \Omega(L^2)$ on depth-L circuits, even when restricted to width-1 circuits.*

In the worst case, a circuit with s nodes may have depth proportional to s, leading to a generation length of $T = \Omega(s^2)$. This holds even for circuits whose non-input nodes form a single directed path.

6 Experiments

We test whether a single shared neuron trained efficiently from complete activation traces generalizes on a nontrivial vision task. We first train a four-hidden-layer ReLU CNN teacher C on CIFAR-10 (Krizhevsky, 2009), obtaining nontrivial test accuracy. Given an image $\boldsymbol{x}$, we record its intermediate activations $C_1(\boldsymbol{x}), C_2(\boldsymbol{x}), \ldots$ and ten class logits. We serialize these values using the construction in Theorem 3 with the affine-output extension in Remark 2: the student uses shared

[8] This assumption does not yield an unconditional lower bound, since edge counting alone does not preclude the existence of a more compact simulation. However, any such simulation must apply to arbitrary circuits, including those that are already "compressed".

[9] Formally, the value $C_j(\boldsymbol{x})$ generated at step t_j is the $(t_i - t_j)$-th most recent value in the autoregressive input. Thus, being used with coefficient $w(v_j, v_i)$ means that $\boldsymbol{w}[-(t_i - t_j)] = w(v_j, v_i)$, so this value contributes $w(v_j, v_i)C_j(\boldsymbol{x})$ to the preactivation at step t_i.

weights $\boldsymbol{w}$, ReLU at hidden and undesignated steps, and identity activation at ten designated output steps. For efficiency, we represent the AR neuron (i.e., $\boldsymbol{w}$) using a sparse implementation which enables very efficient training and inference. We independently initialize the trainable parameters underlying $\boldsymbol{w}$ and optimize them using Adam. Importantly, the student is trained only on activation traces and has no access to the teacher's weights. Training the AR neuron on $4{,}000$ teacher activation trajectories takes roughly 5 minutes on an NVIDIA L40S GPU.

At inference time, we classify images by evaluating the AR neuron only at designated steps. The obtained test accuracy on CIFAR-10 is almost identical to the teacher's; Table 1 summarizes the results. This experiment illustrates the mechanism behind the theory: complete CoT traces transform a general computation into supervised examples, enabling efficient training of a single neuron. See Section F for further details.

Table 1: CIFAR-10 test results. Student results show mean $\pm$ standard deviation.

Model	Supervision	Test accuracy	Teacher agreement
CNN teacher	45,000 images	72.150%	$--$
AR student	4,000 traces	$(72.106 \pm 0.054)\%$	$(99.484 \pm 0.070)\%$

7 Conclusion and Future Directions

We show that, perhaps surprisingly, a single autoregressive neuron can efficiently simulate feedforward circuits, with bounds independent of circuit depth. For the threshold activation, this yields efficient simulation of any polynomial-time Turing machine together with efficient learning from complete CoT supervision. For ReLU, our construction enables the efficient training of a single autoregressive neuron to simulate a large, nontrivial neural networks, as demonstrated by our CIFAR-10 experiment. Together, these results show that a simple next-token predictor can express complex computation, while learning tractability can depend critically on exposing high-quality intermediate computation.

Our results suggest that the quality of chain-of-thought supervision should be understood relative to the next-token architecture: intermediate steps should expose computations that the predictor can represent and learn. Our construction demonstrates an extreme case of this principle, where appropriately structured computational traces enable a single autoregressive neuron to efficiently learn to simulate complex computations. This suggests a tradeoff between the structure of intermediate supervision and the architectural complexity required of the next-token predictor. Although traces in existing text corpora are difficult to control, teacher-generated traces offer an opportunity to shape this supervision. This connects our results to the success of reasoning distillation, in which relatively simple students acquire reasoning capabilities through supervision from more complex teachers (Ho et al., 2023; Guo et al., 2025). While our results do not establish parameter compression, they provide an idealized theoretical basis for this practice: exposing suitable intermediate computations can make a complex task learnable by a much simpler next-token predictor. This perspective motivates jointly designing teacher-generated traces and the student's architecture.

Several theoretical questions remain open for the current architecture. Motivated by Section 5, can simulations without direct edge alignment improve the worst-case s^2 terms in generation length? Can structured or sparse circuits instead admit non-trivial bounds which are governed by their edge count alone? Beyond the current architecture, is there a natural and simple extension of the autoregressive neuron that achieves the optimal linear generation length $T = \Theta(s)$ while preserving time invariance and the CoT-preserving property? Could a stronger but still simple architecture achieve $T = o(s)$? Such an improvement would necessarily move beyond gate-by-gate generation.

AI use statement

We used OpenAI models to edit the wording and presentation of the paper, including the presentation of proofs, identify relevant literature, and assist with writing code for the experiments. These models did not contribute to the logical reasoning underlying the proofs. The authors reviewed the AI-assisted edits, verified the cited references, and tested the experimental code. We take responsibility for the final content of this work.

REFERENCES

Awni Altabaa, Omar Montasser, and John Lafferty. Cot information: Improved sample complexity under chain-of-thought supervision. *arXiv preprint arXiv:2505.15927*, 2025.

Sanjeev Arora, Boaz Barak, et al. *Computational complexity: a modern approach*, volume 1. Cambridge University Press Cambridge, 2009.

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.

Moritz Brösamle and Stephan Eckstein. The expressive power of low precision softmax transformers with (summarized) chain-of-thought. *arXiv preprint arXiv:2605.18079*, 2026.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Ilan Doron-Arad, Idan Mehalel, and Elchanan Mossel. Stochastic autoregressive learning. *arXiv preprint arXiv:2608.07224*, 2026a.

Ilan Doron-Arad, Idan Mehalel, and Elchanan Mossel. A theory of online learning with autoregressive chain-of-thought reasoning. *arXiv preprint arXiv:2605.06819*, 2026b.

Paul Erdos and Pál Turán. On a problem of sidon in additive number theory, and on some related problems. *J. London Math. Soc*, 16(4):212–215, 1941.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in neural information processing systems*, 36:70757–70798, 2023.

Oded Goldreich. *Foundations of Cryptography: A Two-Volume Set*. Cambridge University Press, Cambridge, UK, 2007. ISBN 9780521873734.

Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Johan Haastad, Russell Impagliazzo, Leonid A Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

Steve Hanneke, Idan Mehalel, and Shay Moran. Sample complexity of autoregressive reasoning: Chain-of-thought vs. end-to-end. *arXiv preprint arXiv:2604.12013*, 2026.

Namgyu Ho, Laura Schmid, and Se-Young Yun. Large language models are reasoning teachers. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 14852–14882, 2023.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Nirmit Joshi, Gal Vardi, Adam Block, Surbhi Goel, Zhiyuan Li, Theodor Misiakiewicz, and Nathan Srebro. A theory of learning with autoregressive chain of thought. In *The Thirty Eighth Annual Conference on Learning Theory*, pp. 3161–3212. PMLR, 2025.

Nirmit Joshi, Roey Magen, Nathan Srebro, Nikolaos Tsilivis, and Gal Vardi. Learning to think from multiple thinkers. *arXiv preprint arXiv:2604.24737*, 2026.

Michael J. Kearns and Leslie G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, 1994. doi: 10.1145/174644.174647.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009. URL `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

Zhiyuan Li. The optimal sample complexity of learning autoregressive chain-of-thought. *arXiv preprint arXiv:2607.07423*, 2026.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In *The Twelfth International Conference on Learning Representations*, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Eran Malach. Auto-regressive next-token predictors are universal learners. *arXiv preprint arXiv:2309.06979*, 2023.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. In *International Conference on Learning Representations*, volume 2024, pp. 7690–7706, 2024.

Nicholas Pippenger and Michael J Fischer. Relations among complexity measures. *Journal of the ACM (JACM)*, 26(2):361–381, 1979.

Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Colin Wei, Yining Chen, and Tengyu Ma. Statistically meaningful approximation: a case study on approximating turing machines with transformers. *Advances in Neural Information Processing Systems*, 35:12071–12083, 2022a.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, 2022b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022c.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

CONTENTS

1 Introduction **1**

2 Preliminaries **3**

3 Expressivity of Autoregressive Single Neuron **4**

3.1 Autoregressive Linear Thresholds Simulate Boolean Threshold Circuits 5

3.2 Neural Networks can be simulated by a Single Autoregressive Neuron 5

3.3 Proof technique: serialization via Sidon sets 6

4 Implications for Learning **7**

5 Optimality of the simulation bounds on T and d **8**

6 Experiments **8**

7 Conclusion and Future Directions **9**

A Technical Preliminary **13**

A.1 Supervised Learning with CoT. 13

B Related Work **13**

C Expressivity **14**

C.1 Exact simulation 14

C.2 Approximate simulation 17

C.3 Additional Claims 18

C.4 Proof of Corollary 1 19

D Learning **20**

D.1 Learning Neural Networks with CoT 20

D.2 Hardness of Learning with E2E supervision 21

E Proofs from Section 5 **22**

F Experiments **22**

A TECHNICAL PRELIMINARY

A function $\sigma : \mathbb{R} \to \mathbb{R}$ is *locally Lipschitz* if around every point in its domain, it behaves like a Lipschitz function.

Definition 3 (Locally Lipschitz function)**.** *A function $\sigma : \mathbb{R} \to \mathbb{R}$ is* locally Lipschitz *if, for every $z \in \mathbb{R}$, there exist $\delta > 0$ and $L < \infty$ such that $|\sigma(u) - \sigma(v)| \le L|u - v|$ for all $u, v \in (z - \delta, z + \delta)$.*

A.1 SUPERVISED LEARNING WITH COT.

We recall the learning framework of Joshi et al. (2025). Fix $\Sigma = \{0, 1\}$, an input length n, a generation length T, and a base class $\mathcal{F}_d \subseteq \Sigma^{\Sigma^*}$. For a distribution $\mathcal{D}$ over $\{0, 1\}^n$ and a target generator $f_\star \in \mathcal{F}_d$, both learning settings aim to predict the final output $f_\star^{\mathsf{e2e}\text{-}T}(\boldsymbol{x})$. They differ only in the information available during training. An E2E example is $\big(\boldsymbol{x}, f_\star^{\mathsf{e2e}\text{-}T}(\boldsymbol{x})\big)$, whereas a CoT example is $\big(\boldsymbol{x}, f_\star^{\mathsf{CoT}\text{-}T}(\boldsymbol{x})\big)$. Thus, CoT supervision reveals all intermediate outputs, while E2E supervision treats them as latent variables.

Definition 4 (Realizable E2E learnability)**.** *We say that $\mathcal{F}_d^{\mathsf{e2e}\text{-}T}$ is E2E-learnable over $\{0, 1\}^n$ with generation length T and sample complexity $m(\varepsilon, \delta)$ if there is a learning rule A such that the following holds. For every distribution $\mathcal{D}$ over $\{0, 1\}^n$, target generator $f_\star \in \mathcal{F}_d$, and $\varepsilon, \delta \in (0, 1)$, let*

$$S_{\mathrm{E2E}} = \big((\boldsymbol{x}_1, f_\star^{\mathsf{e2e}\text{-}T}(\boldsymbol{x}_1)), \ldots, (\boldsymbol{x}_m, f_\star^{\mathsf{e2e}\text{-}T}(\boldsymbol{x}_m))\big), \qquad \boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \overset{\text{i.i.d.}}{\sim} \mathcal{D},$$

where $m = m(\varepsilon, \delta)$. Then, with probability at least $1 - \delta$ over the sample and the randomness of A, the predictor $h = A(S_{\mathrm{E2E}}) : \{0, 1\}^n \to \{0, 1\}$ satisfies

$$\Pr_{\boldsymbol{x} \sim \mathcal{D}} \big[h(\boldsymbol{x}) \neq f_\star^{\mathsf{e2e}\text{-}T}(\boldsymbol{x})\big] \le \varepsilon.$$

A CoT learner (Definition 1) receives examples $\big(\boldsymbol{x}_i, f_\star^{\mathsf{CoT}\text{-}T}(\boldsymbol{x}_i)\big)$ but is evaluated on the same final output.

We allow both learners to be improper.

The Computational Aspect. We call a learner efficient if its sample complexity and running time are polynomial in $n, d, T, 1/\varepsilon$, and $\log(1/\delta)$.

The main benefit of CoT supervision is computational. Let $v = \mathrm{VCdim}(\mathcal{F}_d)$ denote the VC dimension of the base class. Joshi et al. (2025) showed that the CONSISTENT rule learns the autoregressive model under CoT supervision with sample complexity

$$m_{\mathrm{CoT}}(\varepsilon, \delta) = O\left(\frac{v \log 2T \log(2/\varepsilon) + \log(1/\delta)}{\varepsilon}\right).$$

Linear Threshold. For $\sigma(z) = \mathbb{1}[z \ge 0]$ and $\mathcal{F}_d = \mathcal{F}_{d,\mathrm{lin}}$, the predictors are linear thresholds on binary contexts with real weights and biases. Their VC dimension is $v = d + 1$, so this bound is nearly linear in the number of parameters and depends only logarithmically on the generation length.[10] Moreover, the CONSISTENT rule is computationally efficient in this setting: the observed CoT trajectories provide T labeled prefix–next-token examples, and a linear threshold consistent with all these examples can be found by linear programming in polynomial time.

Consequently, Corollary 3.7 and Corollary 4.3 of Joshi et al. (2025) give the following result.

Corollary 5. *Autoregressive linear-threshold models (i.e. $\mathcal{F}_{d,\mathrm{lin}}^{\mathsf{e2e}\text{-}T}$) are efficiently learnable with CoT supervision (Definition 1).*

B RELATED WORK

Chain-of-Thought (CoT) was initially introduced as a prompting approach that encourage a series of intermediate reasoning steps before producing the final answer (Wei et al., 2022c; Kojima et al., 2022).

[10] Li (2026) subsequently removed the dependence on T, obtaining the optimal rate $O((d + \log(1/\delta))/\varepsilon)$.

Subsequent work demonstrated that supervising the correctness of these intermediate reasoning steps can further improve reasoning performance, highlighting the benefits of training with CoT supervision (Lightman et al., 2024; Ho et al., 2023; Zelikman et al., 2022). More recently, reinforcement learning (RL) techniques applied in LLM training have encouraged these models to generate more and more tokens during inference, which can be interpreted as a form of CoT reasoning (Jaech et al., 2024; Guo et al., 2025). The Chain-of-Thought supervision we analyze in this work is closely related to the practice of supervised fine-tuning Brown et al. (2020); Wei et al. (2022b); Grattafiori et al. (2024) of LLMs.

Expressivity with CoT. Since Transformers (Vaswani et al., 2017) are the dominant architecture underlying LLMs, growing attention has been devoted to characterizing their expressivity (Wei et al., 2022a), particularly when augmented with CoT (Merrill & Sabharwal, 2024; Brösamle & Eckstein, 2026; Feng et al., 2023). For example,Merrill & Sabharwal (2024) showed that the number of autoregressive steps acts as a computational resource for Transformers and, in particular, that polynomially many steps suffice to recognize every language in P. Our results establish an analogous phenomenon for autoregressive linear-threshold models in the nonuniform setting: additional steps enable more complex computations, and polynomially many steps allow even a single linear-threshold neuron to capture $\mathsf{P/poly}$, the nonuniform analogue of P.

Malach (2023) showed that *time-dependent* AR linear-threshold models can simulate threshold circuits, and hence Turing-machine computations, using a number of autoregressive steps polynomial in the circuit size. In related work, Joshi et al. (2025) established an analogous result for time-invariant AR linear-threshold models, albeit with exponentially many autoregressive steps. More recently, Joshi et al. (2026) showed that these models can efficiently simulate depth-2 threshold circuits and used this simulation to establish hardness results for learning from CoT traces generated by multiple distinct generators.

C Expressivity

Both main-text simulation theorems (Theorems 2 and 3) follow from a common construction for σ-circuits. We first give the exact construction under Assumption 1, proving the exact part of Theorem 3 together with CoT preservation, efficient construction of the feature map and designated steps, and preservation of binary inputs (Proposition 1). Restricting this construction to threshold activation and Boolean inputs also proves Theorem 2. We then establish the approximate part of Theorem 3 by controlling the errors introduced when undesignated steps produce small nonzero values (see Theorem 5).

C.1 Exact simulation

We first state the additional properties of the exact simulation, then prove them jointly with the exact part of Theorem 3.

Proposition 1 (Additional properties of the exact simulation)**.** *Under Assumption 1, the exact simulation in Theorem 3 can be chosen to satisfy the following properties.*

1. ***CoT preservation.*** *There exist designated steps $1 \le t_1 < \cdots < t_s = T$, depending only on n, s, such that, for every size-s σ-circuit C, the simulating neuron satisfies*

$$f^{\text{e2e-}t}_{\boldsymbol{w},\sigma}(\phi_{n,s}(\boldsymbol{x})) = \begin{cases} C_i(\boldsymbol{x}), & \text{if } t = t_i \text{ for some } i \in [s], \\ 0, & \text{otherwise}, \end{cases} \qquad \forall \boldsymbol{x} \in [0,1]^n, t \in [T].$$

2. ***Efficiency.*** *The feature map $\phi_{n,s}$ and the designated steps depend only on n, s and can be constructed in time polynomial in n, s. Evaluating $\phi_{n,s}(\boldsymbol{x})$ takes $O(n')$ real-RAM operations.*

3. ***Binary inputs.*** *The feature map satisfies $\phi_{n,s}(\{0,1\}^n) \subseteq \{0,1\}^{n'}$.*

Proof of the exact part of Theorem 3 and Proposition 1. Order the non-input gates as $v_1, \ldots, v_s$ such that every gate appears after all of its non-input predecessors with v_s the output gate. For

every $i \in [s]$, let θ_i be the bias of v_i, let $\boldsymbol{w}^{\text{in}}[q, i]$ be the weight from input gate $v_{\text{in},q}$ to v_i, and let $\boldsymbol{w}[j, i]$ be the weight from v_j to v_i, assigning weight zero to missing edges.

The goal is to choose vector weights $\boldsymbol{w}$, a feature map $\phi_{n,s}$, and time steps $1 \le t_1 < \cdots < t_s \le T$ such that the autoregressive process satisfies $f^{\text{e2e-}t_i}_{\boldsymbol{w},\sigma}(\phi_{n,s}(\boldsymbol{x})) = C_i(\boldsymbol{x})$.

Next, partition the input values $\boldsymbol{x}[1], \ldots, \boldsymbol{x}[n]$ into $k = \lceil n/s \rceil = O(n/s + 1)$ blocks, each of size at most s. Each block is encoded in a separate block of $\phi_{n,s}(\boldsymbol{x})$. Formally, let $k = \lceil n/s \rceil$, and pad $\boldsymbol{x}$ by setting $\boldsymbol{x}[q] = 0$ for $n < q \le ks$. For every $r \in [k]$ and $j \in [s]$, set $\boldsymbol{x}_r[j] := \boldsymbol{x}[(r-1)s + j]$ to be the j-th element of the r-th block. Let $\boldsymbol{w}^{\text{in}}_r[j, i]$ be the weight from $\boldsymbol{x}_r[j]$ to v_i, that is, $\boldsymbol{w}^{\text{in}}_r[j, i] := \boldsymbol{w}^{\text{in}}[(r-1)s + j, i]$, where weights from padded coordinates are zero. Define the feature map $\phi_{n,s} : [0,1]^n \to [0,1]^{n'}$ and the weight vector $\boldsymbol{w} \in \mathbb{R}^d$ by

$$\boldsymbol{w} = [\ \underbrace{\boldsymbol{v}}_{T \text{ coordinates}}\ ,\ \underbrace{\boldsymbol{z}_k}_{T' \text{ coordinates}}\ , \ldots ,\ \underbrace{\boldsymbol{z}_1}_{T' \text{ coordinates}}\ ,\ \underbrace{\boldsymbol{z}}_{T \text{ coordinates}}\ ]$$

$$\phi_{n,s}(\boldsymbol{x}) = [\ \underbrace{0^{T-1}, 1}_{T \text{ coordinates}}\ ,\ \underbrace{\boldsymbol{u}_k}_{T' \text{ coordinates}}\ , \ldots ,\ \underbrace{\boldsymbol{u}_1}_{T' \text{ coordinates}}\ ,\ \underbrace{0^T}_{T \text{ coordinates}}\ ],$$

for some T, T' and d that will be specified later. The blocks $\boldsymbol{u}_r \in [0,1]^{T'}$ and $\boldsymbol{z}_r \in \mathbb{R}^{T'}$ for $r \in [k]$ are responsible for the contributions of the input values, while $\boldsymbol{z}$ propagates previously computed gate values. We now describe formally the role and construction of each block:

Construction of $\boldsymbol{v}$. The vector $\boldsymbol{v}$ encodes the biases $\theta_1, \ldots, \theta_s$. During the first T autoregressive steps, the 1 in $\phi_{n,s}(\boldsymbol{x})$ shifts across the coordinates of $\boldsymbol{v}$. We set the coordinate aligned with this 1 at time t_i to θ_i. Thus, at time t_i, the bias contribution is θ_i. In other time steps $t \ne t_i$, we set the aligned coordinate to a sufficiently negative value to ensure that $f^{\text{e2e-}t}_{\boldsymbol{w},\sigma}(\phi_{n,s}(\boldsymbol{x})) = 0$. Formally, define

$$M_C := \sup\{|C_i(\boldsymbol{x})| : \boldsymbol{x} \in [0,1]^n,\ i \in [s]\} < \infty,$$

which is well-defined by Observation 2. Set

$$A := \sum_{r=1}^{k} \|\boldsymbol{z}_r\|_1 + M_C \|\boldsymbol{z}\|_1.$$

At every undesignated time, we set the aligned coordinate to $c_\sigma - A - 1$, where c_σ is from Assumption 1. This suppresses every contribution from the input and previously generated gate values. Formally, for every $t \in [T]$, define

$$\boldsymbol{v}[-t] = \begin{cases} \theta_i, & t = t_i \text{ for some } i \in [s], \\ c_\sigma - A - 1, & \text{otherwise}, \end{cases} \tag{1}$$

where here and throughout the proof, negative indices count coordinates from the right endpoint of a block. Note that after t steps, the 1 in $\phi_{n,s}(\boldsymbol{x})$ is aligned with the t^{th} coordinate of $\boldsymbol{v}$ from the right.

Construction of $\boldsymbol{u}_r$ and $\boldsymbol{z}_r$. The vector $\boldsymbol{u}_r$ encodes the input values $\boldsymbol{x}_r[1], \ldots, \boldsymbol{x}_r[s]$ (the r^{th} block in the partition of $\boldsymbol{x}$) separated by zeros. Their locations are determined by indices $t'_1, \ldots, t'_s \in [O(s^2)]$ that will be specified later. For every $r \in [k]$ and $\ell \in [T']$, define $\boldsymbol{u}_r \in [0,1]^{T'}$ by

$$\boldsymbol{u}_r[-\ell] = \begin{cases} \boldsymbol{x}_r[j], & \ell = t'_j \text{ for some } j \in [s], \\ 0, & \text{otherwise}. \end{cases} . \tag{2}$$

That is, $\boldsymbol{x}_r[j]$ is located in the t'^{th}_j coordinate of $\boldsymbol{u}_r$ from the right. We can conclude that:

Observation 1. *At step t_i, the input value $\boldsymbol{x}_r[j]$ is aligned with $\boldsymbol{z}_r[-(t_i + t'_j - 1)]$ for every $r \in [k]$ and $j \in [s]$.*

This input value contributes to the preactivation at step t_i with coefficient $\boldsymbol{w}^{\text{in}}_r[j, i]$. Therefore, we define the weight blocks $\boldsymbol{z}_r$ as follows: For every $r \in [k]$ and $i, j \in [s]$, set

$$\boldsymbol{z}_r[-\ell] = \begin{cases} \boldsymbol{w}^{\text{in}}_r[j, i], & \ell = t_i + t'_j - 1, \\ 0, & \text{otherwise}. \end{cases}$$

To ensure that these assignments are consistent we need that the sums $t_i + t'_j$ are all distinct, which will be guaranteed by the choice of t_i and t'_j later in the proof.

Requirement 1. *The sums $t_i + t'_j$ for $i, j \in [s]$ are distinct.*

Construction of $\boldsymbol{z}$. The vector $\boldsymbol{z}$ is responsible for propagating the outputs of previously computed gates $C_1(\boldsymbol{x}), C_2(\boldsymbol{x}), \ldots, C_s(\boldsymbol{x})$. More specifically, it encodes the weights between non-input gates. If at time t_i the process outputs $f^{\text{e2e-}t_i}_{\boldsymbol{w},\sigma}(\phi_{n,s}(\boldsymbol{x})) = C_i(\boldsymbol{x})$, then at later times $t > t_i$, the value $C_i(\boldsymbol{x})$ shifts through the coordinates of $\boldsymbol{z}$, and is aligned with $\boldsymbol{z}[-(t - t_i)]$. When $t = t_j$ for some $j > i$, we want the previously generated values $C_i(\boldsymbol{x})$ to contribute with coefficients $\boldsymbol{w}[i, j]$. Formally, for every $i \in [s]$ and $\ell \in [T]$, define $\boldsymbol{z} \in \mathbb{R}^T$ by

$$\boldsymbol{z}[-\ell] = \begin{cases} \boldsymbol{w}[i, j], & \ell = t_j - t_i \text{ for some } j > i, \\ 0, & \text{otherwise.} \end{cases}$$

To ensure that these assignments are consistent we need that the differences $t_j - t_i, j > i$ are all distinct, which will be guaranteed by the choice of t_i later in the proof.

Requirement 2. *The differences $t_j - t_i$ over all $1 \le i < j \le s$ are distinct.*

Construction of T, T' and $t_1, \ldots, t_s, t'_1, \ldots, t'_s$. In this part, we choose the time steps $t_1, \ldots, t_s$, the initial input values locations $t'_1, \ldots, t'_s$, and T to satisfy Requirement 1 and Requirement 2. Fortunately, these requirements are almost exactly the same as those required for a Golomb ruler and Sidon set (See Definition 6 and Definition 5). Moreover, there is a well-known construction of Golomb rulers and Sidon sets by Erdos & Turán (1941). Apply Claim 1 with parameter $2s$. Write the resulting Golomb ruler (Sidon set) as $\{a_1 < \cdots < a_{2s}\}$, then $a_1, \ldots, a_{2s}$ are positive, $a_{2s} = O(s^2)$, and their positive pairwise differences and sums are distinct. For $i, j \in [s]$, set $t'_j = a_j$, $t_i = a_{s+i}$, and $T = t_s$. Hence $t_1 < \cdots < t_s = T = O(s^2)$, the differences $t_i - t_j$ for $j < i$ are distinct, and the sums $t_i + t'_j$ are distinct as well. Moreover, we can choose $T' = O(s^2)$ such that $t_i + t'_j \le T'$ for all $i, j \in [s]$. This choice of T' ensures that all non-zero cordinates of $\boldsymbol{u}_r$ are aligned with $\boldsymbol{z}_r$ throughout the autoregressive process, for any $r \in [k]$.

We prove by induction over the first T steps that the generated value is $C_i(\boldsymbol{x})$ at step t_i and zero at every other step. Assume this holds through step $t - 1$.

- If $t \notin \{t_1, \ldots, t_s\}$, then the leading 1 in $\phi_{n,s}(\boldsymbol{x})$ is aligned with $\boldsymbol{v}[-t] = c_\sigma - A - 1$. Since every input coordinate belongs to $[0, 1]$, the absolute input contribution is at most $\sum_{r=1}^{k} \|\boldsymbol{z}_r\|_1$. By the induction hypothesis, every previously generated value is either zero or some $C_j(\boldsymbol{x})$, so the absolute propagated contribution is at most $M_C \|\boldsymbol{z}\|_1$. Thus the preactivation is at most

$$c_\sigma - A - 1 + \sum_{r=1}^{k} \|\boldsymbol{z}_r\|_1 + M_C \|\boldsymbol{z}\|_1 = c_\sigma - 1 < c_\sigma.$$

  The zero-region condition in Assumption 1 gives $f^{\text{e2e-}t}_{\boldsymbol{w},\sigma}(\phi_{n,s}(\boldsymbol{x})) = 0$.

- Suppose that $t = t_i$ for some $i \in [s]$. The leading 1 in $\phi_{n,s}(\boldsymbol{x})$ is aligned with $\boldsymbol{v}[-t_i]$, so its contribution is θ_i. Moreover, for every $r \in [k]$ and $j \in [s]$, the value $\boldsymbol{x}_r[j]$ is aligned with $\boldsymbol{z}_r[-(t_i + t'_j - 1)]$ and contributes $\boldsymbol{w}^{\text{in}}_r[j, i]\boldsymbol{x}_r[j]$. A value generated at an earlier step $\tau < t_i$ is aligned with $\boldsymbol{z}[-(t_i - \tau)]$. By the induction hypothesis, it is zero unless $\tau = t_j$ for some $j < i$. In that case it equals $C_j(\boldsymbol{x})$, and $\boldsymbol{z}[-(t_i - t_j)] = \boldsymbol{w}[j, i]$. Consequently, the preactivation at step t_i is

$$\theta_i + \sum_{q=1}^{n} \boldsymbol{w}^{\text{in}}[q, i]\boldsymbol{x}[q] + \sum_{j=1}^{i-1} \boldsymbol{w}[j, i]C_j(\boldsymbol{x}).$$

  This is precisely the preactivation computed by v_i. Applying σ gives $f^{\text{e2e-}t_i}_{\boldsymbol{w},\sigma}(\phi_{n,s}(\boldsymbol{x})) = C_i(\boldsymbol{x})$, completing the induction.

Thus, at every step $t \in [T]$, the generated value is $C_i(\boldsymbol{x})$ if $t = t_i$ for some $i \in [s]$, and is zero otherwise. This proves the CoT-preserving property. Since $t_s = T$ and v_s is the output gate, the final generated value is $C(\boldsymbol{x})$.

Every coordinate of $\phi_{n,s}(\boldsymbol{x})$ is an input coordinate, zero, or one. Hence, $\phi_{n,s}(\{0,1\}^n) \subseteq \{0,1\}^{n'}$. For threshold activation, every generated token is also binary, so the simulation on Boolean inputs uses $\Sigma = \{0,1\}$.

Finally, $\phi_{n,s}$ depends only on n and s, which determine T and T' and the initial locations of the input coordinates $t'_1, \ldots, t'_s$. The preceding construction therefore constructs a description of $\phi_{n,s}$ in time polynomial in n and s. Given this description, evaluating $\phi_{n,s}(\boldsymbol{x})$ consists of initializing n' coordinates to zero, writing the leading 1, and copying the n input values to their prescribed locations. This takes $O(n')$ real-RAM operations, which proves the efficiency property. Finally, the length of the feature map and the dimension of $\boldsymbol{w}$ are

$$n' = d = T + T'k + T = O(ns + s^2),$$

which completes the proof. □

Proof of Theorem 2. Apply the preceding construction with $\sigma(z) = \mathbb{1}[z \geq 0]$ and restrict the inputs to $\{0,1\}^n$. This activation satisfies Assumption 1. Every coordinate of the feature map is an input coordinate, zero, or one, and every generated value is binary. Thus the construction gives a simulation over $\Sigma = \{0,1\}$ with the stated bounds and a polynomial-time computable feature map depending only on n, s. □

Remark 2 (General readouts)**.** Neural networks often use a vector-valued affine output layer with no activation function. Although Theorem 3 is stated for scalar-output σ-circuits, its construction also supports networks with such output layers. Indeed, the exact construction reproduces every gate value $C_i(\boldsymbol{x})$ at a designated generation step $t_i \in [T]$ by Property 1.. Consequently, such networks can be simulated by applying an external affine readout to the designated values in the generated trajectory. Alternatively, in an extension of the model that permits omitting σ at designated output steps, the shared neuron can also compute the affine output coordinates, one per designated step. Our CIFAR-10 experiments use this designated affine-output extension to learn all 10 class logits jointly with the hidden computation.

C.2 Approximate simulation

The following theorem strengthens the approximate part of Theorem 3 by approximating the entire simulation trajectory, including all intermediate gate values.

Theorem 5 (Approximate simulation)**.** *Suppose that σ satisfies Assumption 2. For every input size n and circuit size s, use the feature map $\phi_{n,s} : [0,1]^n \to [0,1]^{n'}$ and designated steps $1 \leq t_1 < \cdots < t_s = T$ from the construction in the proof of Theorem 3. For every size-s σ-circuit $C : [0,1]^n \to \mathbb{R}$ and every $\varepsilon > 0$, there exists $\boldsymbol{w} \in \mathbb{R}^d$ such that*

1. $|f^{\mathsf{e2e}\text{-}t_i}_{\boldsymbol{w},\sigma}(\phi_{n,s}(\boldsymbol{x})) - C_i(\boldsymbol{x})| \leq \varepsilon, \quad$ *for every $\boldsymbol{x} \in [0,1]^n$ and $i \in [s]$.*
2. $|f^{\mathsf{e2e}\text{-}t}_{\boldsymbol{w},\sigma}(\phi_{n,s}(\boldsymbol{x}))| \leq \varepsilon, \quad$ *for every $\boldsymbol{x} \in [0,1]^n$ and $t \in [T] \setminus \{t_1, \ldots, t_s\}$.*

In particular, $|f^{\mathsf{e2e}\text{-}T}_{\boldsymbol{w},\sigma}(\phi_{n,s}(\boldsymbol{x})) - C(\boldsymbol{x})| \leq \varepsilon$ for every $\boldsymbol{x} \in [0,1]^n$. The bounds $n', d = O(ns + s^2)$ and $T = O(s^2)$, and the efficiency properties of the feature map and designated steps, remain unchanged. The feature map and designated steps are independent of C and ε.

Proof. Use the construction from the proof of Theorem 3, changing only the coordinates of $\boldsymbol{v}$ associated with undesignated steps. Let M_C be the bound on absolute gate values used in that proof, and set $B = \|\boldsymbol{z}\|_1$. Choose $R \geq 1$ such that every circuit gate preactivation lies in $[-R, R]$ for every input. These bounds exist by Observation 2, since local Lipschitz continuity implies boundedness on bounded intervals. Let $L \geq 1$ be a Lipschitz constant for σ on $[-R - B, R + B]$, and set $q = \max\{1, LB\}$ and $\delta = \min\{1, \varepsilon\} \cdot q^{-T}$.

By the tail condition, there exists $h \in \mathbb{R}$ such that $|\sigma(z)| \leq \delta$ whenever $z < h$. At every undesignated step t, set $\boldsymbol{v}[-t] = h - \sum_{r=1}^{k} \|\boldsymbol{z}_r\|_1 - (M_C+1)B - 1$. At designated steps, retain $\boldsymbol{v}[-t_i] = \theta_i$. During the first T steps, previously generated values interact only with $\boldsymbol{z}$, so changing these coordinates of $\boldsymbol{v}$ does not change the coefficients propagating errors.

We prove by induction that, through step t, every generated value differs from its prescribed value by at most δq^t, uniformly over all inputs. Here the prescribed value is $C_i(\boldsymbol{x})$ at t_i and zero otherwise. The claim is vacuous at $t = 0$. Suppose it holds through step $t-1$. Since $\delta q^{t-1} \le 1$, all previously generated values have absolute value at most $M_C + 1$. At an undesignated step, the chosen coordinate of $\boldsymbol{v}$ therefore makes the preactivation at most $h - 1$, so the output has absolute value at most $\delta \le \delta q^t$. At a designated step t_i, the construction with prescribed previous values gives exactly the preactivation of gate v_i. The error in this preactivation is at most $B\delta q^{t-1} \le B$. Both preactivations thus lie in $[-R - B, R + B]$, and the output error is at most $LB\delta q^{t-1} \le \delta q^t$. This completes the induction.

Finally, $\delta q^T \le \varepsilon$. Only the suppressing coordinates of $\boldsymbol{v}$ have changed, so the feature map, designated steps, dimension, number of steps, and stated efficiency properties are unchanged. □

C.3 Additional Claims

Observation 2. *Suppose that σ is bounded on every bounded interval. For every σ-circuit $C : [0,1]^n \to \mathbb{R}$ there exists an integer $M_C < \infty$ such that $|C_i(\boldsymbol{x})| \le M_C$ for all $\boldsymbol{x} \in [0,1]^n$ and $i \in [s]$, where $C_i(\boldsymbol{x})$ is the value of the i-th non-input node.*

Proof. Order the non-input gates as $v_1, \dots, v_s$ such that every gate appears after all of its non-input predecessors. We prove inductively that each gate function $C_i : [0,1]^n \to \mathbb{R}$ is bounded. Suppose this holds for every $j < i$. The preactivation at v_i is a finite linear combination of input coordinates and preceding gate functions, with fixed coefficients. Therefore, it takes values in some bounded interval. Since σ is bounded on every bounded interval, C_i is bounded. Thus, all C_i are bounded, and we may take

$$M_C := \lceil \sup\{|C_i(\boldsymbol{x})| : \boldsymbol{x} \in [0,1]^n,\ i \in [s]\} \rceil < \infty.$$

□

Definition 5 (Sidon set)**.** *A finite set $S \subset \mathbb{N}$ is called a* Sidon set *if whenever $a + b = c + d$ with $a, b, c, d \in S$, then $\{a, b\} = \{c, d\}$. Equivalently,*

$$|\{a + b : a, b \in S,\ a \le b\}| = \binom{|S| + 1}{2}.$$

Definition 6 (Golomb ruler)**.** *A finite set $G \subset \mathbb{N}$ is called a* Golomb ruler *if all pairwise differences are distinct, that is, if $g_b - g_a = g_d - g_c$ with $g_a, g_b, g_c, g_d \in G, g_a < g_b, g_c < g_d$, then $\{g_a, g_b\} = \{g_c, g_d\}$. Equivalently,*

$$|\{g_b - g_a : g_a, g_b \in G,\ g_a < g_b\}| = \binom{|G|}{2}.$$

Observation 3. *A set $S \subset \mathbb{N}$ is a Sidon set if and only if it is a Golomb ruler.*

Proof of Observation 3. Suppose S is a Sidon set. Let $g_a, g_b, g_c, g_d \in S$ with $g_a < g_b$ and $g_c < g_d$. If $g_b - g_a = g_d - g_c$, then $g_b + g_c = g_d + g_a$. Since S is Sidon, we have $\{g_b, g_c\} = \{g_d, g_a\}$. The inequalities $g_a < g_b$ and $g_c < g_d$ rule out the swapped identification, so $g_a = g_c$ and $g_b = g_d$, i.e. $\{g_a, g_b\} = \{g_c, g_d\}$. Thus S is a Golomb ruler. The converse follows similarly. □

The following result is based on the classical construction of Erdos & Turán (1941), originally stated for Sidon sets. Establishing the Sidon property is the main step in the proof. Our formulation additionally records that the resulting set is also a Golomb ruler, can be constructed in polynomial time, and can be translated and restricted to a prescribed size while preserving both properties. For completeness, we include a proof.

Claim 1 (Explicit Golomb ruler and Sidon set by Erdos & Turán (1941))**.** *For every $n \in \mathbb{N}$, one can construct in time $\mathrm{poly}(n)$ a set $G \subseteq [1, 4n^2]$ of size n that is both a Golomb ruler and Sidon set.*

Proof of Claim 1. We use the classical construction of Sidon sets (Erdos & Turán, 1941). Recall that a set $S \subseteq \mathbb{N}$ is *Sidon* if all unordered pairwise sums are distinct, i.e., $a + b = c + d$ with $a, b, c, d \in S$ implies $\{a, b\} = \{c, d\}$.

For $n = 1$, take $G = \{1\}$, and for $n = 2$, take $G = \{1, 2\}$. Both sets satisfy the claim. Assume henceforth that $n \geq 3$. Let p be a prime satisfying $n \leq p < 2n$, which exists by Bertrand's postulate, and can be found in deterministic time of $O(n \log \log n)$, say by Eratosthenes' sieve. Since $p \geq 3$, it is odd. Consider $S' = \{(x, x^2) : x \in \mathbb{F}_p\} \subseteq \mathbb{F}_p^2$. We first note that S' is Sidon. Indeed, suppose $(i, i^2) + (j, j^2) = (k, k^2) + (\ell, \ell^2)$. Then we have the next equalities in $\mathbb{F}_p$:

$$i + j = k + \ell,$$
$$i^2 + j^2 = k^2 + \ell^2.$$

Since $(i + j)^2 = i^2 + 2ij + j^2$ and 2 is invertible in $\mathbb{F}_p$, the two equalities imply $ij = k\ell$. Thus the unordered pairs $\{i, j\}$ and $\{k, \ell\}$ have the same sum and product, hence they are roots of the same quadratic polynomial $x^2 - (i + j)x + ij$, and thus $\{i, j\} = \{k, \ell\}$. To embed the Sidon set S' into the nonnegative integers, we use the mapping $f(x, y) = 2px + y$, where $x, y \in \{0, \ldots, p - 1\}$. We argue that $S := f(S')$ is a Sidon set. First, note that the mapping f is injective, as each pair (x, y) corresponds to a unique integer representation in base $2p$ and thus $|f(S')| = p \geq n$. To confirm that the Sidon property is preserved, suppose that for $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4) \in S'$, we have $f(x_1, y_1) + f(x_2, y_2) = f(x_3, y_3) + f(x_4, y_4)$. Expanding this using our mapping yields:

$$2p(x_1 + x_2 - x_3 - x_4) = (y_3 + y_4 - y_1 - y_2).$$

Since $|y_3 + y_4 - y_1 - y_2| < 2p$ and the left-hand side is a multiple of $2p$, both sides of the equation must be zero, which implies that $x_1 + x_2 = x_3 + x_4$ and $y_1 + y_2 = y_3 + y_4$ simultaneously in $\mathbb{N}$. These are exactly the conditions defining the Sidon property for S' in $\mathbb{F}_p^2$. Since S' is Sidon, these equalities imply that $\{(x_1, y_1), (x_2, y_2)\} = \{(x_3, y_3), (x_4, y_4)\}$. Finally, because f is injective, this coordinate-wise equality forces the equality of the mapped values: $\{f(x_1, y_1), f(x_2, y_2)\} = \{f(x_3, y_3), f(x_4, y_4)\}$. Thus, the set S is a Sidon set in $\mathbb{N}$. Finally, define

$$G = \{2px + (x^2 \bmod p) + 1 : 0 \leq x < n\},$$

where x ranges over integers and $x^2 \bmod p$ denotes the least nonnegative residue modulo p. Since $n \leq p$, the set G is a subset of $S + 1$ and has exactly n elements by injectivity of f. Translation and taking subsets preserve the Sidon property, so G is Sidon and, by Observation 3, a Golomb ruler. Moreover,

$$\max G \leq 2p(n - 1) + (p - 1) + 1 = p(2n - 1) < 4n^2.$$

Thus $G \subseteq [1, 4n^2]$, and the construction takes $\mathrm{poly}(n)$ time. □

C.4 Proof of Corollary 1

Proof of Corollary 1. Set $s_n = \lceil \tau(n) \log(\tau(n)) \log(n) \rceil$ and $\phi = \phi_{n,s_n}$, where ϕ_{n,s_n} is from Theorem 2. These choices depend only on n and τ. Fix a deterministic Turing machine M running in time at most $\tau(n)$. By Pippenger & Fischer (1979), its computation on inputs of length n can be simulated by a Boolean circuit C_n consisting of AND, OR, and NOT gates, of size $O(\tau(n) \log(\tau(n)))$[11], where the implicit constant may depend on M. The circuit satisfies $C_n(\boldsymbol{x}) = M(\boldsymbol{x})$ for every $\boldsymbol{x} \in \{0, 1\}^n$. For all sufficiently large n depending on M, the factor $\log(n)$ exceeds this constant, so C_n has size at most s_n. Fix such an n.

For threshold activation $\sigma(z) = \mathbb{1}[z \geq 0]$, each Boolean gate can be simulated by a threshold gate as follows:

$$x \wedge y = \sigma(x + y - \tfrac{3}{2}), \qquad x \vee y = \sigma(x + y - \tfrac{1}{2}), \qquad \neg x = \sigma(\tfrac{1}{2} - x).$$

Hence, C_n is a σ-circuit of size at most s_n. By Remark 1, we may pad it to size s_n without changing its output. Applying Theorem 2 with the feature map $\phi = \phi_{n,s_n}$ chosen above gives a vector $\boldsymbol{w} \in \mathbb{R}^d$ such that

$$f_{\boldsymbol{w}}^{\text{e2e-}T}(\phi(\boldsymbol{x})) = C_n(\boldsymbol{x}) = M(\boldsymbol{x}) \qquad \text{for every } \boldsymbol{x} \in \{0, 1\}^n.$$

The feature map and the generated tokens are binary by Theorem 2, so the entire autoregressive computation uses $\Sigma = \{0, 1\}$. Moreover,

$$n' = d = O(ns_n + s_n^2) = \widetilde{O}(n\tau(n) + \tau(n)^2),$$
$$T = O(s_n^2) = \widetilde{O}(\tau(n)^2).$$

[11] See also Sec. 6 and the proof sketch of Theorem 6.7 in Arora et al. (2009).

Since $\tau(n) = O(\mathrm{poly}(n))$, we have $s_n = \mathrm{poly}(n)$. The map ϕ_{n,s_n} is computable in time polynomial in n and s_n, so ϕ is computable in $\mathrm{poly}(n)$ time. The feature maps and the constants in the dimension and generation-length bounds are independent of M. □

D LEARNING

D.1 LEARNING NEURAL NETWORKS WITH COT

We consider binary classification by scalar-output ReLU networks, using $\sigma(z) = \max\{0, z\}$ and $\Sigma = \mathbb{R}$. For a network $C : [0,1]^n \to \mathbb{R}$, the target label is $\mathbb{1}[C(\boldsymbol{x}) > 0]$. Each training example reveals an input $\boldsymbol{x}$ together with all neuron activations $C_1(\boldsymbol{x}), \ldots, C_s(\boldsymbol{x})$. By Property 1, the observed activations suffice to construct the complete training trajectory of the simulating autoregressive neuron: place $C_i(\boldsymbol{x})$ at step t_i and zero at every other step. The designated steps depend only on s, so this conversion requires no knowledge of the network weights. In particular, every prefix and its next-token target are known. Property 1 guarantees that a single shared ReLU neuron is consistent with all these training pairs.

Finding a consistent neuron. Each observed trajectory provides prefix–next-token pairs $(\boldsymbol{z}, y)$. Since all target tokens are nonnegative, consistency of a ReLU neuron $f_{\boldsymbol{w},b}$ with such a pair is equivalent to

$$\begin{aligned} \langle \boldsymbol{w}, \boldsymbol{z} \rangle + b &= y && \text{if } y > 0, \\ \langle \boldsymbol{w}, \boldsymbol{z} \rangle + b &\le 0 && \text{if } y = 0. \end{aligned}$$

Thus, fitting m trajectories of length T reduces to a linear feasibility problem with $d+1$ variables and at most mT constraints. CoT preservation guarantees feasibility. Moreover, consistency on every observed prefix implies, by induction over the generation steps, that the learned neuron reproduces each training trajectory under autoregressive generation. In particular, it predicts every training label correctly. For rational input data, a feasible solution can be found in time polynomial in the encoding length of this linear program.

Generalization. We bound the complexity of the learned autoregressive model by unrolling it into a feedforward network. The bound applies to every choice of the shared parameters, including solutions that do not correspond to the original teacher architecture.

Claim 2. *Fix $d, T \ge 1$ and a feature map ψ taking values in $\mathbb{R}^*$ that is independent of the training sample. The classifier class*

$$\left\{ \boldsymbol{x} \mapsto \mathbb{1}\left[f_{\boldsymbol{w},b}^{\text{e2e-}T}(\psi(\boldsymbol{x})) > 0 \right] : \boldsymbol{w} \in \mathbb{R}^d,\ b \in \mathbb{R} \right\}$$

has VC dimension $O((d+1)T^2 \log(2(d+1)T))$. Consequently, in the realizable setting, a sufficient number of i.i.d. examples for every consistent classifier to have error at most ε, with probability at least $1-\delta$, is

$$m = O\left(\frac{(d+1)T^2 \log(2(d+1)T) \log(2/\varepsilon) + \log(1/\delta)}{\varepsilon} \right).$$

Proof. Unrolling the autoregressive computation produces a feedforward network with T ReLU units. Each unit receives at most d inputs from the initial context and preceding units, and has one bias. Allowing separate weights and biases at each step gives a larger class with $W = O((d+1)T)$ parameters and $L = O(T)$ layers. To match the architecture assumed by Bartlett et al. (2019), we append a fixed identity output unit that copies the final generated token. This leaves the computed function unchanged.

By the VC-dimension bound of Bartlett et al. (2019), this enlarged classifier class has VC dimension $O(WL \log(2W))$. The shared-parameter class is a subclass and satisfies the same upper bound. Composing with the fixed map ψ cannot increase VC dimension. The sample bound follows from the standard realizable generalization bound for consistent classifiers. □

Corollary 6. *Let $C : [0,1]^n \to \mathbb{R}$ be a size-s ReLU network, and let $\mathcal{D}$ be any distribution over $[0,1]^n$. Given complete neuron activations on sufficiently many i.i.d. inputs drawn from $\mathcal{D}$, one*

can learn a single autoregressive ReLU neuron of dimension $d = O(ns + s^2)$ using $T = O(s^2)$ generation steps such that, with probability at least $1 - \delta$,

$$\Pr_{\boldsymbol{x}\sim\mathcal{D}}\left[\mathbb{1}\left[f^{\text{e2e-}T}_{\boldsymbol{w},b}(\phi_{n,s}(\boldsymbol{x})) > 0\right] \neq \mathbb{1}[C(\boldsymbol{x}) > 0]\right] \leq \varepsilon.$$

The required number of training examples is polynomial in $n, s, 1/\varepsilon$, and $\log(1/\delta)$. Learning reduces to a linear program of polynomial size; for rational data, its running time is polynomial in the encoding length.

Remark 3 (Regression)**.** The argument also extends to approximating $C(\boldsymbol{x})$ under any bounded nonnegative loss that is a nondecreasing function of the absolute prediction error and vanishes at zero. Here we normalize the loss to take values in $[0, 1]$. Examples include bounded absolute-error loss $\ell(\hat{y}, y) = \min\{1, |\hat{y} - y|\}$ and bounded squared-error loss $\ell(\hat{y}, y) = \min\{1, |\hat{y} - y|^2\}$. Indeed, the bounds of Bartlett et al. (2019) also apply to pseudodimension. For these losses, the induced loss class has pseudodimension bounded by a constant multiple of that of the predictor class, so standard uniform convergence bounds yield polynomial sample complexity. The same linear-programming procedure reproduces every training output exactly, giving zero empirical loss. Thus, sufficiently many samples ensure arbitrarily small expected loss.

D.2 Hardness of Learning with E2E supervision

We consider $\sigma(z) = \mathbb{1}[z \geq 0]$ and $\Sigma = \{0, 1\}$.

Theorem 4.4 of Joshi et al. (2025) rules out efficient E2E learning under the assumption that polynomial-size constant-depth threshold circuits are hard to learn. Their reduction uses dimension and generation length $O(s^L n)$ to simulate a size-s, depth-L threshold circuit. This overhead is polynomial for constant L, but it becomes exponential when the depth grows with the running time. Our simulation has polynomial overhead for arbitrary polynomial-time computations and therefore yields hardness from the weaker assumption that one-way functions exist.

We first recall the cryptographic assumption. A one-way function is easy to compute but hard to invert on a random input.

Definition 7 (One-way function)**.** *A function $f\colon \{0,1\}^* \to \{0,1\}^*$ is a one-way function if it is computable in polynomial time and, for every probabilistic polynomial-time algorithm I,*

$$\Pr_{\substack{\boldsymbol{x}\sim\mathrm{Unif}(\{0,1\}^n)\\ \textit{randomness of } I}}\left[I(f(\boldsymbol{x}), 1^n) \in f^{-1}(f(\boldsymbol{x}))\right] \leq n^{-\omega(1)}.$$

One-way functions imply pseudorandom generators (Haastad et al., 1999), which in turn imply pseudorandom function (PRF) families (Goldreich et al., 1986). Such functions have polynomial-size circuits. The guarantee is that no polynomial-time algorithm can distinguish a pseudorandom function from a uniformly random function with non-negligible probability. Existence of PRF implies hardness of learning (Valiant, 1984; Kearns & Valiant, 1994). More specifically, an efficient learner with nontrivial prediction advantage would distinguish a pseudorandom function from a uniformly random function by testing the learned predictor on fresh inputs, contradicting pseudorandomness. This sequence of works give the following result:

Theorem 6 ((Kearns & Valiant, 1994; Goldreich et al., 1986; Haastad et al., 1999))**.** *If one-way functions exist, then there are a polynomial q and function classes $\mathcal{G}_n \subseteq \{0,1\}^{\{0,1\}^n}$ such that every function in $\mathcal{G}_n$ is computable by a Boolean circuit of size at most $q(n)$, but no probabilistic polynomial-time algorithm weakly learns $(\mathcal{G}_n)_{n\geq 1}$, even improperly and under the uniform distribution.*

Combining with Corollary 1 gives the following result:

Corollary 7. *If one-way functions exist, then autoregressive linear-threshold models are not efficiently E2E-learnable (Definition 4), even improperly.*

Proof. Suppose that an efficient E2E learner exists. Let $(\mathcal{G}_n)_{n\geq 1}$ and q be as in Theorem 6, and set $s = q(n)$. By Theorem 2, there is a single polynomial-time computable map $\phi_{n,s} : \{0,1\}^n \to \{0,1\}^{n'}$ such that, for every $g \in \mathcal{G}_n$, there is a linear threshold $f_{\boldsymbol{w}_g} \in \mathcal{F}_{d,\mathrm{lin}}$ satisfying

$$f^{\text{e2e-}T}_{\boldsymbol{w}_g}(\phi_{n,s}(\boldsymbol{x})) = g(\boldsymbol{x}) \qquad \text{for every } \boldsymbol{x} \in \{0,1\}^n,$$

where $n' = O(ns + s^2)$, $d = O(ns + s^2)$, and $T = O(s^2)$. Crucially, $\phi_{n,s}$ depends only on n and the common circuit-size bound s, not on g.

We use the assumed E2E learner to learn $\mathcal{G}_n$. Transform every labeled example $(\boldsymbol{x}_i, g(\boldsymbol{x}_i))$ into $(\phi_{n,s}(\boldsymbol{x}_i), g(\boldsymbol{x}_i))$ and run the E2E learner with parameters d and T. The transformed sample is realizable by $f_{\boldsymbol{w}_g}$. If the learner returns h, output the predictor $\boldsymbol{x} \mapsto h(\phi_{n,s}(\boldsymbol{x}))$. Its error under the original input distribution equals the error of h under the pushforward distribution induced by $\phi_{n,s}$. Since s, n', d, and T are polynomial in n, this gives a probabilistic polynomial-time weak PAC learner for $(\mathcal{G}_n)_{n\geq 1}$, contradicting Theorem 6. □

E Proofs from Section 5

Proof of Theorem 4. Consider a width-1, depth-L σ-circuit with one non-input node v_i at each depth $i \in [L]$ and an edge (v_{i-1}, v_i) for every $i \in \{2, \ldots, L\}$. Choose the weights of these $L-1$ edges to be pairwise distinct. Since the simulation scheme must handle arbitrary σ-circuits, it must handle this circuit.

Let $1 \leq t_1 < \cdots < t_L \leq T$ be the designated steps, and define $\Delta_i := t_i - t_{i-1}$ for every $i \in \{2, \ldots, L\}$. Direct edge alignment gives $\boldsymbol{w}[-\Delta_i] = w(v_{i-1}, v_i)$. Since the edge weights are pairwise distinct, the positive integers $\Delta_2, \ldots, \Delta_L$ are also pairwise distinct. Therefore,

$$T \geq t_L = t_1 + \sum_{i=2}^{L} \Delta_i \geq 1 + \sum_{q=1}^{L-1} q = 1 + \binom{L}{2},$$

where the last inequality follows because $\Delta_2, \ldots, \Delta_L$ are pairwise distinct. □

F Experiments

Dataset and supervision split. The CIFAR-10 dataset (Krizhevsky, 2009) contains $60{,}000$ 32×32 color images across ten classes. We partition the official $50{,}000$-image training set deterministically, using seed 42, into $45{,}000$ teacher-training images and $5{,}000$ validation images used for teacher selection. In the saved validation ordering, the first $4{,}000$ images supply student recovery trajectories, the next 500 form a holdout from student recovery, and the last 500 are unused for student recovery and evaluation. The recovery holdout belongs to the teacher's validation partition. The official $10{,}000$-image test set is used only for final evaluation after freezing all five step-$10{,}000$ checkpoints. It is not used for recovery, diagnostic tuning, or checkpoint or seed selection.

Teacher architecture. The teacher is a Convolutional Neural Network (CNN) consisting of four convolution–batch-normalization–ReLU stages, followed by global average pooling and a final affine readout layer. Before compiling the network into the autoregressive format, the batch normalization parameters are folded exactly into the weights and biases of the preceding convolutions. The architectural details of the four recovered stages are summarized in Table 2.

Table 2: Architectural details of the teacher's compiled convolutional stages.

Stage	Channels	Kernel	Stride	Activation Shape	#Parameters
1	32	5×5	4	$32 \times 8 \times 8$	2,432
2	64	3×3	2	$64 \times 4 \times 4$	18,496
3	128	3×3	2	$128 \times 2 \times 2$	73,856
4	128	3×3	1	$128 \times 2 \times 2$	147,584

These stages yield $4{,}096$ scalar ReLU gates and contain $242{,}368$ learned parameters. The compiled circuit appends ten affine output gates in class order. The student learns these coefficients and ten output biases, adding $1{,}290$ parameters. In total, recovery covers $4{,}106$ designated gates and $243{,}658$ tied learned parameters.

Teacher training and selection. We train the teacher on the $45{,}000$ labeled training images using cross-entropy and SGD, selecting the checkpoint by validation accuracy. The frozen teacher achieves 72.15% accuracy on the official test set.

Student construction. We construct an autoregressive student with shared weights and a fixed activation schedule, using the affine-output extension in Remark 2. At every step other than the ten designated output steps, the recurrence is

$$h_t^{\mathrm{stu}} = \mathrm{ReLU}(\langle w_{\mathrm{stu}}, z_t \rangle)\,, \tag{3}$$

where w_{stu} is shared across steps and z_t is the shifting window containing the encoded image and preceding student outputs. At the ten designated output steps, we omit ReLU in (3) to produce signed logits.

Following the construction in Theorem 3, the simulation use a shared neuron parametrize by $\boldsymbol{w}$ that contain many structural zeros along the weights of the teacher's circuit. Therefore our goal here is to learn the weights of the teacher's circuit and not these zeros. We use the Sidon construction from Claim 1 to assign each circuit edge and bias to a unique coordinate of w_{stu}. The learner is given the architecture, sparsity pattern, and convolutional parameter tying, but not the teacher's weights. That is, the coordinate assignment is independent of the teacher's weight values. We learn the associated coefficients from activation traces and store them in a sparse coordinate map, leaving structural zeros implicit.

In more details, the weights that force undesignated outputs to zero all have the same value. We store this value once and identify the corresponding coordinates from the designated-step schedule. For each final learned student, we bound its outputs at all designated steps over inputs in $[0,1]^n$ by interval propagation, including the ten output logits. We use the largest absolute value among the interval endpoints in place of M_C when computing A as in the proof of Theorem 3. The norms count every occurrence of a tied coefficient. We set the entries of w_{stu} that align with the constant 1 in the encoded input at undesignated steps to $-A-1$. Inductively, undesignated preactivations are at most -1, while Sidon alignment reproduces the learned gates at designated steps. Thus, evaluating only designated gates preserves the recurrence in exact arithmetic.

Then we can represent w_{stu} sparsely as a map from occupied coordinates to learnable parameters, without store or create the elements whose value is zero.

Student training. We train five students on the same $4{,}000$ complete teacher trajectories, using recovery seeds $4042, \ldots, 4046$. The seeds vary parameter initialization and minibatch ordering; the teacher checkpoint, image subsets, and optimization hyperparameters remain fixed. Tied hidden parameters and output biases are initialized independently according to $\mathcal{N}(0, 0.02^2)$; the pooled output coefficients use $\mathcal{N}(0, 0.005^2)$. We use Adam with constant learning rate 0.001, minibatch size 64, and a budget of $10{,}000$ steps.

Training uses designated positions only. The frozen teacher supplies fixed hidden prefixes, post-ReLU activation targets, and raw logit targets. For a hidden target y and student preactivation a, we minimize

$$\ell(a, y) = \begin{cases} (a-y)^2, & y > 0, \\ \mathrm{ReLU}(a)^2, & y = 0. \end{cases} \tag{4}$$

Thus, a positive target determines the corresponding preactivation, whereas a zero target imposes only the constraint $a \le 0$. For affine output gates, we use squared error $(a-y)^2$ for every target sign. With fixed teacher prefixes, this objective is convex in the tied trainable parameters; it need not be strictly convex or identify the teacher's parameters. Training the student on $4{,}000$ trajectories for $10{,}000$ Adam steps took roughly 5 minutes on a single NVIDIA L40S GPU.

Student evaluation. At test time, the student classifies images using its own intermediate activations, followed by global-average pooling and its learned affine readout.

We evaluate all five students on the official test set and report both test accuracy and agreement with the teacher's predictions. Teacher agreement is the fraction of test examples on which the student and teacher predict the same class, irrespective of whether that prediction is correct. We also report two measures of logit error relative to the teacher. Both measures compute the root mean squared difference between student and teacher logits, averaged over all test images and all ten classes. Logits are the class scores before softmax. Free-logit RMSE evaluates the complete student computation: the student generates its own hidden activations and applies global-average pooling and its learned affine readout. This measure therefore reflects errors in both the hidden computation and the readout.

Teacher-forced output RMSE instead supplies the teacher's final hidden activations to the student's pooling and learned affine readout. Using the same hidden activations as the teacher isolates readout error from differences in the hidden computation. Both measures assess agreement with the teacher's logits; test accuracy remains the measure of classification performance.

Evaluation runs sequentially on a single NVIDIA A40 with batch size 64. The teacher uses the same evaluation procedure.

For each student metric, we report the mean and one sample standard deviation across the five students,

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}, \qquad N = 5.$$

For percentage metrics, the standard deviation is in percentage points. It measures variation across students evaluated on the same test images; it is not a confidence interval over newly sampled images or dataset splits.

Table 3: Official-test evaluation of all five final students trained on $4{,}000$ recovery trajectories for $10{,}000$ Adam steps with learned affine outputs. Student metrics show mean $\pm$ sample standard deviation on the same $10{,}000$ test images. Percentage-metric deviations are in percentage points.

Metric	Official test set
Student accuracy	$(72.106 \pm 0.054)\%$
Teacher accuracy	72.150%
Teacher agreement	$(99.484 \pm 0.070)\%$
Free-logit RMSE	0.023852 ± 0.004095
Teacher-forced output RMSE	0.004897 ± 0.002957

Experimental conclusion. With $4{,}000$ recovery trajectories, the student jointly learns the hidden convolutional parameters and affine output head. Across five final checkpoints, it achieves 72.106% mean test accuracy, compared with the teacher's 72.150%, and 99.484% mean teacher agreement. These results support our theoretical finding that complete activation supervision enables a single shared autoregressive neuron to recover the behavior of a multilayer network without access to its weights.