Neural Networks as Universal Finite-State Machines: A Constructive ReLU Simulation Framework for NFAs

Anonymous authors
Paper under double-blind review

Abstract

We present a formal and constructive framework establishing the equivalence between nondeterministic finite automata (NFAs) and standard feedforward ReLU neural networks. By encoding automaton states as binary vectors and transitions as sparse linear layers, we show that ReLU activations simulate nondeterministic branching, subset construction, and ϵ-closures in a mathematically precise manner. Our core theoretical results prove that a three-layer ReLU network of width $\mathcal{O}(n)$ can exactly recognize any regular language accepted by an n-state NFA—without recurrence, memory, or approximation. Furthermore, we show that gradient descent over structure-preserving networks preserves symbolic semantics and acceptance behavior. Extensive experiments across multiple validation tasks—including parallel path tracking, symbolic subset construction, ϵ-closure convergence, acceptance classification, structural training invariants, and functional equivalence—achieve perfect or near-perfect empirical alignment with ground-truth automata. This work provides the first provably complete symbolic simulation of NFAs within standard deep learning architectures, uniting automata theory with neural computation through ReLU dynamics.

1 Introduction

The relationship between symbolic computation and neural networks has long fascinated both the theoretical computer science and machine learning communities. Finite automata Rabin & Scott (1959); Hopcroft et al. (2006); Sipser (1996) are among the most fundamental models of computation, capturing the essence of regular languages, while modern neural architectures such as ReLU networks Nair & Hinton (2010) are the cornerstone of deep learning systems. Despite their seemingly disparate origins, a growing body of research has sought to reconcile these paradigms by simulating automata within neural frameworks (Giles et al., 1992; Weiss et al., 2018; Graves et al., 2014).

Early attempts primarily employed recurrent networks (RNNs) to approximate deterministic finite automata (DFAs), leveraging their sequential dynamics to process string inputs (Giles et al., 1991; Korsky & Berwick, 2019). However, these approaches often required complex training, lacked interpretability, and provided no symbolic guarantees. More recent work has explored extracting automata from trained networks using queries and counterexamples (Weiss et al., 2018), or designing architectures with embedded stack or memory structures (DuSell & Chiang, 2022; Reed & de Freitas, 2016), but symbolic fidelity remained elusive.

In this paper, we propose a novel and constructive equivalence between nondeterministic finite automata (NFAs) and feedforward ReLU networks. Unlike prior work that approximates or reverse-engineers automata behavior, we show that every NFA can be simulated *exactly* using a fixed-depth, width-bounded ReLU network—without recurrences, memory modules, or learning heuristics. Our formulation encodes NFA states as binary vectors, transitions as sparse linear layers, and nondeterministic branches as parallel activations (Nair & Hinton, 2010; Goodfellow et al., 2016).

We provide a full theoretical framework characterizing this simulation, including symbolic encoding of subset construction, ϵ-closures, and final accept/reject decisions. We further prove that under structural constraints, gradient descent preserves the automaton's behavior during supervised training. Our work builds upon but significantly extends previous studies on neural-symbolic systems (Bhattamishra et al., 2020; Merrill

et al., 2020; Ergen & Grillo, 2024), and complements concurrent research on DFA simulation by feedforward networks (Dhayalkar, 2025b).

The key contributions of this work are:

- A symbolic encoding of NFA transitions and ϵ-closures using ReLU matrix compositions, with provable correctness and convergence guarantees.
- A constructive proof that every regular language (recognized by some NFA) is exactly simulatable by a 3-layer ReLU network of width $\mathcal{O}(n)$, where n is the number of NFA states.
- A learning-theoretic result showing that gradient descent over symbolic ReLU networks preserves automaton semantics under structure-preserving updates.
- A full suite of empirical validations demonstrating perfect agreement between symbolic NFAs and their neural equivalents on string acceptance, ϵ-closures, subset construction, and structural preservation.

Together, these results provide the first formal bridge between classical automata theory and deep learning, showing that ReLU networks are not only universal function approximators—but also *universal finite-state machines*.

2 Related Work

This paper contributes to a long-standing line of research at the intersection of automata theory, neural networks, and symbolic computation. Below we review the most relevant prior efforts and clarify the novelty of our approach.

Automata Simulation by Neural Networks: The idea of using neural networks to model automata has received significant attention. Early work explored the approximation of regular languages using RNNs Giles et al. (1992), including techniques for training second-order networks to accept deterministic finite automata (DFAs) Giles et al. (1991). More recent work has focused on extracting symbolic automata from trained RNNs using queries and counterexamples Weiss et al. (2018) or differentiable memory models such as Neural Turing Machines Graves et al. (2014). However, these models typically rely on recurrent or sequential architectures with opaque representations.

Symbolic Interpretability of Neural Models: Efforts to interpret neural networks symbolically have grown with the popularity of neural-symbolic learning. Notable examples include differentiable pushdown automata DuSell & Chiang (2022), differentiable interpreters Reed & de Freitas (2016), and formal grammars embedded into transformer and RNN architectures Bhattamishra et al. (2020); Merrill et al. (2020). These approaches aim to bridge symbolic logic and differentiable computation but often involve approximate, heuristic, or complex inductive mechanisms. Our work provides an exact, symbolic simulation of NFAs using only feedforward ReLU layers—no recurrence, memory modules, or training heuristics are required.

ReLU Networks and Formal Language Theory: The theoretical properties of ReLU networks have been widely studied in machine learning Nair & Hinton (2010); Goodfellow et al. (2016). Their universality and expressivity have been linked to their piecewise linearity, with recent studies exploring their capacity to represent formal structures Dhayalkar (2025a); Ergen & Grillo (2024). However, no prior work formally characterizes the class of regular languages as exactly simulatable by feedforward ReLU networks. In contrast, we show that every NFA (and thus every regular language) can be encoded via sparse ReLU layers, and that subset construction, ϵ-closures, and acceptance can be realized as exact compositions of linear and ReLU operations.

Learning Formal Languages with Neural Networks: Supervised learning of regular languages using neural networks has also been explored Butoi et al. (2025). However, these works treat neural networks as black-box approximators, aiming for empirical accuracy without structural guarantees. In contrast, our

Theorem 4 shows that when trained under structure-preserving constraints, ReLU networks not only approximate but also converge to exact simulations of the underlying NFA.

Equivalence and Completeness Results: While neural networks are universal function approximators, their symbolic equivalence to classical models remains an underdeveloped area. Our Theorem 5 provides a formal equivalence between NFAs and feedforward ReLU networks in the context of language recognition, inspired by classical results like the Rabin-Scott power-set construction Rabin & Scott (1959), but executed in a neural substrate. A concurrent line of work by Dhayalkar (2025b) develops a constructive theory of deterministic finite automata (DFAs) simulated by feedforward neural networks, showing that DFA transitions can be unrolled into ReLU or threshold-activated layers. While their work is restricted to deterministic models, our results extend these ideas to *non-deterministic* finite automata, incorporating ϵ-transitions and symbolic closure dynamics, and establishing exact recognizability of all regular languages using ReLU networks. To our knowledge, this is the first symbolic and constructive completeness theorem connecting finite-state automata and standard feedforward neural networks.

3 Preliminaries

We briefly review the formal foundations of non-deterministic finite automata (NFAs) and feedforward ReLU neural networks. Our notation and constructions follow standard automata theory texts Hopcroft et al. (2006); Sipser (1996) and build on recent neural-symbolic research Graves et al. (2014); Weiss et al. (2018).

Non-Deterministic Finite Automaton (NFA): A non-deterministic finite automaton is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q = \{q_1, \ldots, q_n\}$ is a finite set of states,
- Σ is the input alphabet,
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$ is the transition function, allowing ϵ-transitions,
- $q_0 \in Q$ is the initial state, and
- $F \subseteq Q$ is the set of accepting states.

The automaton accepts a string $x = x_1 x_2 \ldots x_T \in \Sigma^*$ if there exists a sequence of transitions beginning at q_0 and ending in some $q \in F$ that consumes x, possibly including intermediate ϵ-transitions Hopcroft et al. (2006).

State Vector Encoding: We encode the current active NFA states using a binary vector $s_t \in \{0, 1\}^n$, where $[s_t]_i = 1$ if and only if q_i is active at time t. The initial vector s_0 typically encodes ϵ-closure($\{q_0\}$).

Symbolic Transition Matrices: Each input symbol $x \in \Sigma$ has an associated symbolic transition matrix $T^x \in \{0, 1\}^{n \times n}$ where $T^x_{ij} = 1$ if $q_j \in \delta(q_i, x)$. Similarly, the ϵ-transition matrix T^ϵ encodes $\delta(q_i, \epsilon)$. A single transition step is expressed as:

$$s_{t+1} = \mathrm{ReLU}(T^{x_t} s_t)$$

This matrix-based view of automata enables efficient symbolic reasoning and lends itself to neural implementations.

ReLU Neural Networks: We focus on standard feedforward ReLU networks with activation $\mathrm{ReLU}(z) = \max(0, z)$ applied element-wise. Such networks have been extensively studied in machine learning Nair & Hinton (2010); Goodfellow et al. (2016) and are widely used in modern architectures. In our setting, ReLU layers serve as interpretable symbolic operators that propagate state vectors under automaton dynamics.

Subset Construction: Given an NFA $\mathcal{A}$, the subset construction algorithm converts it into a DFA by interpreting each DFA state as a subset of Q Hopcroft et al. (2006). In our formulation, this is realized through a sequence of ReLU layer compositions, where each layer computes the reachable subset from the previous one using matrix-based updates.

4 Theoretical Framework

4.1 Theorem 1: ReLU State Vector Representation

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a non-deterministic finite automaton (NFA), where $|Q| = n$. Then there exists a representation of the state of $\mathcal{A}$ at time step t using a binary vector $s_t \in \{0,1\}^n$ such that, for each input symbol $x_t \in \Sigma$, a feedforward ReLU network with weights derived from the transition matrix $T^{x_t} \in \{0,1\}^{n \times n}$ computes the next state vector as

$$s_{t+1} = \mathrm{ReLU}(T^{x_t} s_t),$$

which correctly encodes the set of active NFA states after consuming x_t. (Proof provided in Appendix A.1).

This theorem formalizes how the internal state of an NFA, normally represented as a subset of active states, can be encoded as a binary vector and propagated using a single matrix multiplication followed by a ReLU activation. Each entry in the state vector s_t corresponds to whether a particular state q_i is active at time t (1 if active, 0 otherwise). The transition matrix T^{x_t} encodes the possible transitions on input x_t: $T^{x_t}_{ij} = 1$ if $q_j \in \delta(q_i, x_t)$, and 0 otherwise.

This theorem shows that ReLU networks can inherently represent symbolic computation by treating NFA state transitions as linear transformations followed by a thresholding nonlinearity. The use of ReLU ensures that activations are non-negative and binary when the input state vector is binary. The insight is that a forward pass through a ReLU network layer corresponds precisely to the propagation of nondeterministic state activations in an NFA, thereby simulating one time step of computation in a neural substrate.

Note that this theorem assumes that ϵ-transitions are disabled. The reason is that the ReLU formulation $s_{t+1} = \mathrm{ReLU}(T^{x_t} s_t)$ models only transitions explicitly triggered by input symbols. Incorporating ϵ-closures would require additional computation between steps and is handled separately in Lemma 2 and Theorem 3. Disabling ϵ-transitions ensures the correctness and tractability of the presented construction without loss of generality.

While prior research has explored the simulation of automata using recurrent neural networks (RNNs) and other models Korsky & Berwick (2019), the specific mechanism of using ReLU activations in feedforward networks to simulate NFA transitions step-by-step is novel. Previous approaches often treat automata learning as a black-box classification or embedding problem, whereas our formulation yields an interpretable, symbolic simulation using standard neural components.

4.2 Lemma 1: Parallel Path Tracking

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be an NFA and let $s_t \in \{0,1\}^{|Q|}$ be the binary vector representing the active states at time t. Then the ReLU-based update

$$s_{t+1} = \mathrm{ReLU}(T^{x_t} s_t)$$

simulates all valid parallel transition paths of $\mathcal{A}$ upon consuming input symbol $x_t \in \Sigma$ in a single matrix-vector operation. (Proof provided in Appendix A.2).

NFAs may transition to multiple next states from a given active state and input. This lemma states that a linear map followed by ReLU can simultaneously compute all such next states—effectively performing parallel symbolic execution over all nondeterministic branches. This reveals that nondeterminism, often viewed as inherently sequential or branching, can be encoded in the superposition of ReLU activations. Note that this lemma assumes that ϵ-transitions are disabled with the same reasoning as explained in Theorem 1.

Traditional methods for simulating NFAs often rely on sequential processing or recursive algorithms Korsky & Berwick (2019). The use of ReLU activations in feedforward networks to capture all possible transitions in parallel introduces a novel perspective in automata simulation within neural architectures.

4.3 Theorem 2: Subset Construction via ReLU Composition

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be an NFA with $|Q| = n$. Then a composition of ReLU layers can simulate the subset construction algorithm deterministically, such that the final activation vector $s_t \in \{0,1\}^n$ at time t encodes the DFA-equivalent subset of active NFA states after consuming the input string $x = x_1 x_2 \ldots x_t$.

Formally, let $T^{x_i} \in \{0,1\}^{n \times n}$ denote the transition matrix for symbol x_i. Then the composed ReLU update:

$$s_t = \mathrm{ReLU}(T^{x_t}\,\mathrm{ReLU}(T^{x_{t-1}} \cdots \mathrm{ReLU}(T^{x_1} s_0) \cdots))$$

computes the subset construction path for input $x = x_1 \cdots x_t$ without ϵ-transitions. Each intermediate activation vector s_i represents the reachable NFA states after symbol x_i, and the final s_t encodes the deterministic DFA state after the full input. (Proof provided in Appendix A.3.)

Subset construction transforms an NFA into an equivalent DFA by encoding each DFA state as a subset of Q. This theorem asserts that such subsets can be represented and deterministically updated by ReLU layers, where each forward pass propagates the subset under the current input symbol. The composition of ReLU layers acts as a deterministic automaton over the space of subsets of Q, simulating the power-set construction within a neural architecture. Note that this theorem assumes that ϵ-transitions are disabled with the same reasoning as explained in Theorem 1.

The subset construction method is a classical approach in automata theory for converting NFAs to DFAs Rabin & Scott (1959). However, implementing this method within neural network architectures using ReLU activations presents a novel integration of symbolic computation and deep learning frameworks.

4.4 Lemma 2: ReLU Closure under ϵ-Transitions

Let $\mathcal{A} = (Q, \Sigma \cup \{\epsilon\}, \delta, q_0, F)$ be an NFA with $|Q| = n$, and let $T^\epsilon \in \{0,1\}^{n \times n}$ be the transition matrix corresponding to ϵ-transitions. Then the iterative update

$$s^{(k+1)} = \mathrm{ReLU}(T^\epsilon s^{(k)}), \quad s^{(0)} = s_t,$$

converges in at most n steps to the ϵ-closure of s_t, i.e., the smallest superset of active states reachable via zero or more ϵ-transitions. (Proof provided in Appendix A.4).

The ϵ-closure of a set of NFA states consists of all states reachable via chains of ϵ-transitions. This lemma shows that repeated applications of a ReLU-activated matrix product over T^ϵ converges to this closure within a number of steps bounded by the number of states. Unlike symbolic graph traversal algorithms used classically to compute ϵ-closures, this formulation provides a fully differentiable method using a fixed linear transformation and ReLU activations. The finite convergence bound arises from the nilpotent nature of ϵ-transition graphs in acyclic form, making this method computationally efficient and differentiable.

Standard algorithms compute ϵ-closures via depth-first traversal or dynamic programming Aho & Hopcroft (1974). To the best of our knowledge, no previous work frames ϵ-closure computation as a convergent ReLU iteration. This offers a symbolic mechanism expressible inside deep learning pipelines without external recursion.

4.5 Theorem 3: ReLU Networks Simulate ϵ-NFA Acceptors

Let $\mathcal{A} = (Q, \Sigma \cup \{\epsilon\}, \delta, q_0, F)$ be an ϵ-NFA with $|Q| = n$. Then there exists a 3-layer feedforward ReLU neural network of width $\mathcal{O}(n)$ that, for any input string $x = x_1 x_2 \ldots x_T$, accepts if and only if $\mathcal{A}$ accepts x, by simulating all transitions and ϵ-closures via differentiable computation. (Proof provided in Appendix A.5).

The aforementioned 3-layer ReLU network with the following components:

- Layer 1: Computes the ϵ-closure of the start state q_0, resulting in the initial state vector s_0^ϵ.
- Layer 2: For any input string $x = x_1 x_2 \ldots x_T$, the network performs T sequential transition updates followed by ϵ-closures, using unrolled matrix multiplications and ReLU.
- Layer 3: Checks if any state in F is active using an inner product with the final state vector s_T.

This theorem demonstrates that a feedforward ReLU network with three functional stages can simulate an ϵ-NFA's behavior exactly: the first stage initializes and computes the ϵ-closure of the start state; the second stage composes transitions and closures per input; and the final stage checks acceptance. This result elevates symbolic string recognition into a neural framework without recursion, RNNs, or sampling-based approximations. To our knowledge, no prior work achieves end-to-end ϵ-NFA acceptance using strictly feedforward ReLU networks.

Previous efforts at learning automata behavior with neural networks rely on recurrent mechanisms Weiss et al. (2018) or non-symbolic encodings Graves et al. (2014). This theorem departs from such approaches by establishing an exact and symbolic simulation using feedforward ReLU layers.

4.6 Corollary 1: ReLU Acceptors are NFA-Complete

Any regular language recognized by a NFA can be accepted by a 3-layer feedforward ReLU network of width $\mathcal{O}(n)$, where $n = |Q|$ is the number of NFA states. (Proof provided in Appendix A.6).

This result follows from Theorem 3, showing that for any NFA (with or without ϵ-transitions), a fixed-depth feedforward ReLU network can simulate its full behavior — including parallel transitions, ϵ-closures, and final acceptance — entirely through linear operations and thresholding. This corollary formalizes the expressive power of ReLU networks in relation to automata theory. While regular languages are traditionally associated with symbolic string-processing machines, we show they are also realizable within the standard architecture of deep learning — without recurrent components. To our knowledge, this is the first proof of full regular language recognition by shallow ReLU networks via symbolic simulation.

Previous works have focused on extracting automata-like behaviors from trained recurrent models Weiss et al. (2018), or training neural networks to approximate regular languages via sequence classification Butoi et al. (2025). However, no prior results have proven that standard feedforward ReLU networks are functionally equivalent to nondeterministic finite automata in their recognition capacity.

4.7 Lemma 3: Gradient Flow Preserves NFA Semantics

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be an NFA with $|Q| = n$, and let f_θ be a feedforward ReLU network parameterized by θ, constructed to simulate $\mathcal{A}$ as per Theorem 3. Suppose f_θ is trained via gradient descent on a dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$, where each $x^{(i)} \in \Sigma^*$ and $y^{(i)} = \mathbf{1}[x^{(i)} \in L(\mathcal{A})]$. If the loss function $\mathcal{L}(\theta)$ is minimized such that the network's predictions match the NFA's acceptances on $\mathcal{D}$, and the updates to θ preserve the sparsity and structure of the transition matrices corresponding to δ, then the trained network f_θ continues to simulate $\mathcal{A}$ accurately on all inputs. (Proof provided in Appendix A.7).

This lemma asserts that when a ReLU network simulating an NFA is trained using gradient descent on data labeled according to the NFA's language, and the training process maintains the structural integrity of the network's transition representations, the network's behavior remains consistent with the original NFA. While prior work has explored the extraction of automata from trained neural networks or the approximation of automata behavior using neural models, this lemma provides a formal guarantee that the symbolic behavior of an NFA can be preserved during gradient-based training of a ReLU network, provided certain structural constraints are maintained. This bridges the gap between symbolic automata theory and gradient-based learning in neural networks.

Previous studies, such as Weiss et al. (2018), have investigated the extraction of finite automata from recurrent neural networks, focusing on interpretability post-training. However, these approaches often involve approximations or lack guarantees about the preservation of automata semantics during training. Our lemma provides a novel theoretical foundation ensuring the preservation of NFA behavior in a ReLU network trained via gradient descent.

4.8 Theorem 4: Learnability of NFA via Supervised ReLU Networks

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be an NFA with $|Q| = n$, and let f_θ be a ReLU network parameterized by θ with width $\mathcal{O}(n)$ and depth $\mathcal{O}(T)$, where T is the maximum length of inputs. Suppose f_θ is initialized with sparse structured weights and trained on a dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}$ with labels $y^{(i)} = \mathbf{1}[x^{(i)} \in L(\mathcal{A})]$.

Then gradient descent over the binary cross-entropy (BCE) loss:

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{D}|} \sum_i \mathrm{BCE}(f_\theta(x^{(i)}), y^{(i)})$$

with structure-preserving updates (i.e., weight masks aligned with δ) converges to a model where $f_\theta(x) \approx \mathbf{1}[x \in L(\mathcal{A})]$ on both seen and unseen strings. (Proof provided in Appendix A.8).

This theorem connects automata theory with learning theory by showing that a ReLU network with symbolic structure can *learn* an NFA's behavior through supervised training. If the initial network respects the sparse transition structure of the automaton, gradient descent will adjust only relevant connections, leading to faithful behavior on all strings in the language. While prior work shows that RNNs or transformers can *approximate* automata behavior, this is the first result demonstrating that standard gradient-based training of ReLU networks can converge to an exact simulation of an NFA. Crucially, we highlight that sparsity and structural constraints are essential to ensure interpretability and generalization.

4.9 Theorem 5: Equivalence of ReLU Networks and NFA Recognizers

Let $\mathcal{L} \subseteq \Sigma^*$ be any regular language. Then there exists a feedforward ReLU network f_θ of width $\mathcal{O}(n)$ and input-dependent depth $\mathcal{O}(T)$ such that:

$$f_\theta(x) = 1 \iff x \in \mathcal{L},$$

where n is the number of states in some NFA $\mathcal{A}$ that recognizes $\mathcal{L}$, and $T = |x|$ is the input length. Conversely, for every ReLU network constructed as in Theorem 3, there exists an NFA $\mathcal{A}'$ that accepts exactly the language recognized by f_θ. (Proof provided in Appendix A.9).

This theorem formally establishes the functional equivalence between two widely studied computational models: nondeterministic finite automata (NFAs) and feedforward ReLU networks (under symbolic transition encoding). It asserts that the set of regular languages is precisely the class of languages recognizable by structurally constrained ReLU networks. While neural networks are widely celebrated for their expressive power, this result establishes a symbolic correspondence between ReLU networks and finite-state machines. It bridges two historically distinct paradigms—automata theory and deep learning—by showing that standard neural components can structurally simulate the behavior of classical computational models.

Prior work has investigated simulation of automata by recurrent neural networks Weiss et al. (2018), and neural language models approximating regular languages Butoi et al. (2025). However, none of these establish a symbolic and structural equivalence between NFAs and feedforward ReLU networks, as we do here.

5 Experiments

5.1 Experimental Setup

To empirically validate our theoretical results, we designed a rigorous experimental framework that simulates nondeterministic finite automata (NFAs) and tests their equivalence with structurally constrained ReLU networks. All experiments are conducted using synthetic NFAs with interpretable, low-dimensional structure.

Synthetic NFA Generation. We consider two configurations. The first follows the baseline setup with $n = 6$ states and input alphabet $\Sigma = \{a, b\}$. The second is a more complex variant with $n = 10$ states and $\Sigma = \{a, b, c, d\}$. In both cases, each state has one or more outgoing transitions for each symbol, sampled uniformly from the set of states. Additionally, with a probability of 0.3, we insert ϵ-transitions between

random pairs of states. The start state is fixed as $q_0 = 0$, and the accepting set F consists of one randomly selected state.

Dataset Construction. In the 6-state setup, we sample strings uniformly from Σ^* of lengths between 1 and 10. In the 10-state setup, the maximum string length is increased to 15. In both cases, we use the automaton itself to label strings with binary labels indicating acceptance. We generate 200 training samples and 100 test samples for each random seed for both configurations.

Neural Architecture. We construct a symbolic feedforward ReLU network that exactly encodes the NFA's transition and acceptance behavior. The full construction consists of three conceptual stages: (i) an ϵ-closure layer that expands reachable states from the current set using the ϵ-transition matrix, (ii) a sequence of transition layers, one per input symbol, interleaved with ϵ-closure updates, and (iii) an acceptance head that performs a dot product with the accepting state indicator vector followed by a thresholding operation. This forms the 3-layer symbolic ReLU network described in Theorem 3.

This full 3-layer architecture is used in the *Acceptance Accuracy*, *Weight Sparsity and Structure*, and *Symbolic Equivalence Test* experiments, which require full end-to-end ϵ-NFA simulation. Simpler network variants are used for other validations: *Path Enumeration Test* uses a single-layer ReLU model, *Subset Construction Validation* uses a stacked ReLU model with one layer per input symbol, and ϵ-*Closure Dynamics* uses a recurrent-style single-matrix ReLU iteration to compute closure sets. All networks share the same symbolic, interpretable design principles.

Evaluation Protocol. Each model is evaluated on the test set using binary classification accuracy. We repeat all experiments across 5 random seeds for both configurations, independently regenerating the NFA, dataset, and model for each seed. We report the mean accuracy, standard deviation, and 95% confidence intervals using Student's t-distribution.

Implementation Details. All experiments are implemented in PyTorch Paszke et al. (2019) and executed on an NVIDIA GeForce RTX 4060 GPU with CUDA acceleration.

5.2 Path Enumeration Test

To validate Theorem 1 and Lemma 1, we evaluate whether a single-layer symbolic ReLU network correctly simulates all reachable NFA states under nondeterministic transitions. We disable ϵ-transitions for this experiment. For both configurations, we generate 100 test strings per seed and compare the ReLU activation mask to the ground-truth reachable states.

Across 5 random seeds, all runs in both configurations achieved 100% exact match between symbolic NFA reachability and ReLU activations, with mean accuracy 1.0000, standard deviation 0.0000, and the 95% confidence interval collapsing to a single point due to zero variance. This confirms that the symbolic ReLU construction correctly encodes nondeterministic paths as predicted.

5.3 Subset Construction Validation

To validate Theorem 2, we test whether a stacked ReLU network—comprising one symbolic layer per input symbol—simulates the DFA-style subset construction of an NFA without ϵ-transitions. For each randomly generated NFA, we generate 100 random strings and trace the reachable subset of NFA states after each input symbol. We compare this against the ReLU activations in the symbolic network at each layer, which deterministically composes transitions via $s_{t+1} = \mathrm{ReLU}(T^{x_t} s_t)$.

We repeat the experiment across 5 random seeds under both experimental configurations, yielding 100 total trace comparisons per configuration. In all cases, ϵ-transitions are disabled. All ReLU traces exactly matched the ground-truth subset sequences, resulting in a mean accuracy of 1.0000, standard deviation 0.0000, and a 95% confidence interval collapsing to a single point due to zero variance. This empirically confirms that stacked ReLU layers simulate subset construction deterministically, as predicted by the theorem.

5.4 ϵ-Closure Dynamics

To validate Lemma 2, we test whether a recurrent-style symbolic ReLU network using the ϵ-transition matrix converges to the true ϵ-closure. For each run, we randomly generate a synthetic NFA with ϵ-transitions inserted with probability 0.3. Given a random start state q_i, we compute the symbolic ϵ-closure of $\{q_i\}$ and compare it to the result obtained by iteratively applying the ReLU recurrence $s^{(k+1)} = \text{ReLU}(T^\epsilon s^{(k)})$ with accumulation.

We repeat the experiment across 5 random seeds under both experimental configurations. In each case, we perform 100 random tests per seed, totaling 500 validation instances per configuration. A test is considered successful if the final ReLU activation pattern exactly matches the symbolic ϵ-closure. All 5 seeds in both configurations achieved perfect match accuracy, resulting in a mean of 1.0000, standard deviation 0.0000, and the 95% confidence interval collapses to a single point due to zero variance. This confirms the correctness of Lemma 2 and empirically demonstrates that ReLU dynamics reliably simulate ϵ-closure computation within n steps.

5.5 Acceptance Accuracy

To validate Theorem 3 and Corollary 1, we test whether a full 3-layer symbolic ReLU network correctly accepts and rejects strings in accordance with the ground-truth ϵ-NFA. For each random seed, we generate a new NFA with ϵ-transitions (probability of 0.3), construct the corresponding symbolic network with ϵ-closure, transition composition, and acceptance detection layers, and evaluate its binary predictions on a held-out test set of 100 strings.

We repeat this procedure across 5 random seeds under both experimental configurations. In the 6-state setup with $\Sigma = \{a, b\}$, the mean test accuracy was 0.9960, with standard deviation 0.0089 and a 95% confidence interval of $(0.9849, 1.0071)$. In the 10-state setup with $\Sigma = \{a, b, c, d\}$, the mean accuracy was 0.9540, with standard deviation 0.0451 and a 95% confidence interval of $(0.8981, 1.0099)$. These results confirm that the ReLU model accurately simulates full ϵ-NFA acceptance behavior across diverse automata in both settings, thereby validating the theoretical construction.

5.6 Weight Sparsity and Structure

To validate Lemma 3 and Theorem 4, we investigate whether symbolic transition structure in the 3-layer ReLU network is preserved during gradient-based learning. We initialize the model with symbolic transition matrices of a synthetic ϵ-NFA and allow gradient descent to train the weights on labeled string acceptance data, while enforcing updates only at the non-zero entries of the original structure.

After training on 200 strings for 5 epochs across 5 random seeds under both experimental configurations, we inspect whether any weights became non-zero outside the initial symbolic structure. In all 5 runs for both configurations, we observe zero violations, with mean preservation rate 1.0000, standard deviation 0.0000, and 95% confidence interval collapsing to a single point due to zero variance. This empirically confirms that supervised training maintains the sparse automaton structure, preserving interpretability and validating our theoretical claims.

5.7 Symbolic Equivalence Test

To validate Theorem 5, we test whether the 3-layer symbolic ReLU network is functionally equivalent to its corresponding ϵ-NFA in terms of string acceptance. For each seed, we generate a new NFA and construct its symbolic ReLU equivalent. We then sample 100 random strings and compare the acceptance decisions of the NFA and the ReLU model on each input.

We repeat this across 5 random seeds under both experimental configurations. In the 6-state setup with $\Sigma = \{a, b\}$, the mean equivalence accuracy was 0.9920, with standard deviation 0.0179, and a 95% confidence interval of $(0.9698, 1.0142)$. In the 10-state setup with $\Sigma = \{a, b, c, d\}$, the mean equivalence accuracy was 0.9580, with standard deviation 0.0460, and a 95% confidence interval of $(0.9008, 1.0152)$. This result confirms

that symbolic ReLU networks are empirically equivalent to their NFA counterparts, thereby validating Theorem 5.

6 Conclusion

This paper introduces the first formal and constructive simulation framework that realizes nondeterministic finite automata (NFAs) entirely within the architecture of standard feedforward ReLU neural networks. By translating automaton transitions, ϵ-closures, and subset constructions into sequences of symbolic matrix operations and ReLU activations, we provide a provably correct neural representation for regular languages.

Our theoretical results establish that every regular language can be recognized by a shallow, 3-layer ReLU network of width $O(n)$—with no recurrence, memory, or approximation involved. We further show that gradient-based learning over structure-preserving networks maintains symbolic interpretability and correctness. These results culminate in a formal equivalence between ReLU networks and NFAs as finite-state recognizers.

Empirical validations across seven experimental protocols—covering state propagation, subset construction, ϵ-closure dynamics, string acceptance, structure preservation, and symbolic equivalence—demonstrate perfect or near-perfect alignment between ReLU models and ground-truth automata across diverse settings.

These results position ReLU networks not just as universal approximators, but as universal symbolic recognizers. By formally uniting automata theory with deep learning architectures, this work opens new directions in neural-symbolic reasoning, interpretable computation, and structured learning with formal guarantees.

7 Limitations

This work presents a symbolic framework for simulating NFAs using ReLU networks, with strong theoretical guarantees and empirical support. We assume the structure of the target NFA is either given or learnable through structured training, a standard setting in formal language modeling. Our construction also focuses on NFAs with acyclic or finitely converging ϵ-transitions, which suffices for most practical use cases.

While our learnability results (Theorem 4) rely on structure-preserving updates, relaxing this constraint while retaining symbolic fidelity is a promising direction. Lastly, our analysis is limited to regular languages. Whether similar constructions extend to richer language classes or more expressive architectures such as transformers remains an open question for future work.

8 Broader Impact

This work advances the theoretical understanding of how neural networks can simulate symbolic computation, providing a constructive bridge between automata theory and deep learning. The results have potential implications for neural-symbolic reasoning, formal verification, program synthesis, and interpretable AI. By enabling ReLU networks to operate with symbolic precision, this framework could support applications requiring transparency, correctness, and alignment with classical formal systems. As a theoretical contribution, this work poses no ethical risks.

References

Alfred V. Aho and John E. Hopcroft. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1974. ISBN 0201000296.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the Ability and Limitations of Transformers to Recognize Formal Languages. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7096–7116, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.576. URL `https://aclanthology.org/2020.emnlp-main.576/`.

Alexandra Butoi, Ghazal Khalighinejad, Anej Svete, Josef Valvoda, Ryan Cotterell, and Brian DuSell. Training neural networks as recognizers of formal languages, 2025. URL https://arxiv.org/abs/2411.07107.

Sahil Rajesh Dhayalkar. The geometry of relu networks through the relu transition graph, 2025a. URL https://arxiv.org/abs/2505.11692.

Sahil Rajesh Dhayalkar. Neural networks as universal finite-state machines: A constructive deterministic finite automaton theory, 2025b. URL https://arxiv.org/abs/2505.11694.

Brian DuSell and David Chiang. Learning hierarchical structures with differentiable nondeterministic stacks, 2022. URL https://arxiv.org/abs/2109.01982.

Ekin Ergen and Moritz Grillo. Topological expressivity of relu neural networks, 2024. URL https://arxiv.org/abs/2310.11130.

C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, and Y. C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 05 1992. ISSN 0899-7667. doi: 10.1162/neco.1992.4.3.393. URL https://doi.org/10.1162/neco.1992.4.3.393.

C.L. Giles, D. Chen, C.B. Miller, H.H. Chen, G.Z. Sun, and Y.C. Lee. Second-order recurrent neural networks for grammatical inference. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume ii, pp. 273–281 vol.2, 1991. doi: 10.1109/IJCNN.1991.155350.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014. URL https://arxiv.org/abs/1410.5401.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA, 2006. ISBN 0321455363.

Samuel A. Korsky and Robert C. Berwick. On the computational power of rnns, 2019. URL https://arxiv.org/abs/1906.06349.

William Merrill, Gail Weiss, Yoav Goldberg, Roy Schwartz, Noah A. Smith, and Eran Yahav. A formal hierarchy of RNN architectures. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 443–459, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.43. URL https://aclanthology.org/2020.acl-main.43/.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32, pp. 8024–8035, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

Michael Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 04 1959. doi: 10.1147/rd.32.0114.

Scott Reed and Nando de Freitas. Neural programmer-interpreters, 2016. URL https://arxiv.org/abs/1511.06279.

Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996. ISBN 053494728X.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5247–5256. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/weiss18a.html`.

A Appendix

A.1 Proof of Theorem 1: ReLU State Vector Representation

We define the n-dimensional state vector s_t such that

$$(s_t)_i = \begin{cases} 1, & \text{if } q_i \text{ is active at time } t, \\ 0, & \text{otherwise.} \end{cases}$$

Given a fixed input symbol $x_t \in \Sigma$, let $T^{x_t} \in \{0,1\}^{n \times n}$ be the transition matrix where

$$T^{x_t}_{ij} = \begin{cases} 1, & \text{if } q_j \in \delta(q_i, x_t), \\ 0, & \text{otherwise.} \end{cases}$$

The matrix-vector product $T^{x_t} s_t$ produces a vector $v \in \mathbf{Z}^n_{\geq 0}$ where

$$v_j = \sum_{i=1}^{n} T^{x_t}_{ji} (s_t)_i.$$

This entry is non-zero if and only if there exists at least one state q_i that was active at time t such that $q_j \in \delta(q_i, x_t)$.

To recover the correct binary activation pattern for s_{t+1}, we apply the ReLU activation:

$$(s_{t+1})_j = \text{ReLU}(v_j) = \begin{cases} v_j, & \text{if } v_j > 0, \\ 0, & \text{otherwise.} \end{cases}$$

Now, observe that since $v_j \in \mathbf{Z}_{\geq 0}$ and $s_t \in \{0,1\}^n$, $v_j > 0$ if and only if q_j is reachable from some active q_i via x_t, i.e., $q_j \in \delta(q_i, x_t)$ for some i with $(s_t)_i = 1$. Therefore, $(s_{t+1})_j = v_j > 0$ if and only if q_j is in the new active set of states.

Finally, we apply a binary thresholding if needed (e.g., $\mathbf{1}(v_j > 0)$), but this is optional since the magnitude of v_j does not affect the semantics — only its positivity matters. Thus, ReLU suffices.

A.2 Proof of Lemma 1: Parallel Path Tracking

Let $s_t \in \{0,1\}^n$ be the state vector at time t, where $(s_t)_i = 1$ if state q_i is active. Let $T^{x_t} \in \{0,1\}^{n \times n}$ be the transition matrix for input symbol x_t as defined in Theorem 1. Then the matrix-vector product yields:

$$v_j = \sum_{i=1}^{n} T^{x_t}_{ji} (s_t)_i.$$

If any i satisfies both $(s_t)_i = 1$ and $T^{x_t}_{ji} = 1$, then $v_j \geq 1$, and $\text{ReLU}(v_j) = v_j > 0$. Otherwise, $v_j = 0$. Thus, $s_{t+1} = \text{ReLU}(T^{x_t} s_t)$ encodes all q_j such that there exists a valid transition from some active q_i to q_j under x_t.

Hence, all nondeterministic paths are captured in parallel by the activations of the next state vector.

A.3 Proof of Theorem 2: Subset Construction via ReLU Composition

We proceed inductively. Let the initial state vector $s_0 \in \{0,1\}^n$ be a one-hot encoding of q_0 or the ϵ-closure thereof. Let T^{x_i} be the transition matrix for symbol x_i.

Base Case ($t = 1$):

$$s_1 = \mathrm{ReLU}(T^{x_1} s_0)$$

correctly computes all states reachable by one transition.

Inductive Step: Assume s_{t-1} encodes the DFA subset corresponding to the set of NFA states reachable by $x_1, \ldots, x_{t-1}$. Then

$$s_t = \mathrm{ReLU}(T^{x_t} s_{t-1})$$

computes the union of all $q_j \in \delta(q_i, x_t)$ for each active $q_i \in s_{t-1}$.

Because ReLU preserves positive activations and the transition matrices are deterministic, the vector s_t will encode the correct set of NFA states reachable after t steps. Thus, repeated composition of ReLU updates realizes subset construction deterministically.

A.4 Proof of Lemma 2: ReLU Closure under ϵ-Transitions

Let $s^{(0)} \in \{0,1\}^n$ be the binary vector representing the initially active states at time t. Define

$$s^{(k+1)} = \mathrm{ReLU}(T^\epsilon s^{(k)}), \quad k \geq 0.$$

Step 1 (Monotonicity): Each application of T^ϵ followed by ReLU adds new states to the active set (or leaves it unchanged), since

$$(s^{(k+1)})_j = \max\left(0, \sum_{i=1}^{n} T^\epsilon_{ji} (s^{(k)})_i\right) \geq (s^{(k)})_j.$$

Thus, the sequence $\{s^{(k)}\}$ is monotonic: $s^{(k)} \leq s^{(k+1)}$ (element-wise).

Step 2 (Finite Convergence): Since each state is either 0 or 1 in the binary vector and there are n total states, the monotonic sequence $\{s^{(k)}\}$ can change at most n times before reaching a fixed point.

Step 3 (Correctness): At convergence, $s^{(K)}$ includes all states reachable from the original set via a chain of ϵ-transitions. Any such state is reachable via a finite path of length at most n, and will be activated through repeated matrix application. Conversely, no unreachable states will be activated because T^ϵ is binary and has no negative entries or spurious transitions.

Therefore, $s^{(K)}$ represents the ϵ-closure of s_t, and $K \leq n$.

A.5 Proof of Theorem 3: ReLU Networks Simulate ϵ-NFA Acceptors

We define a 3-layer ReLU network composed of:

Layer 1 (Initialization and ϵ-Closure): Let $s_0 \in \{0,1\}^n$ be the one-hot vector for q_0, the start state. Compute the initial state vector s_0^ϵ by iterating:

$$s_0^\epsilon = \epsilon\text{-closure}(s_0) = \mathrm{ReLU}\left(T^\epsilon \mathrm{ReLU}(\cdots \mathrm{ReLU}(T^\epsilon s_0) \cdots)\right),$$

until convergence in at most n steps, as shown in Lemma 2. This can be unrolled and embedded into the first ReLU layer with fixed weights and skip connections.

Layer 2 (Per-Symbol Transitions and Closures): For each $x_t \in x$, compute:

$$s'_t = \mathrm{ReLU}(T^{x_t} s_{t-1}), \quad s_t = \epsilon\text{-closure}(s'_t) = \mathrm{ReLU}\left(T^\epsilon \mathrm{ReLU}(\cdots \mathrm{ReLU}(T^\epsilon s'_t) \cdots)\right),$$

using again a finite unrolled series of ReLU layers. The second functional layer computes these recurrently over the input string.

Layer 3 (Acceptance Check): Let $f \in \{0,1\}^n$ be the binary indicator vector for the accepting states F. Define the final output as:

$$y = \sigma(f^\top s_T),$$

where $\sigma(z) = \mathbf{1}[z > 0]$ is a hard threshold (or a smooth approximation such as a sigmoid during training). This detects whether any accepting state is active at the end.

All three layers are constructed via matrix-vector operations and ReLU activations. Each transformation is deterministic and mimics the transition and closure operations of $\mathcal{A}$. Thus, the ReLU network accepts the input iff $\mathcal{A}$ accepts it.

A.6 Proof of Corollary 1: ReLU Acceptors are NFA-Complete

Let $\mathcal{A} = (Q, \Sigma \cup \{\epsilon\}, \delta, q_0, F)$ be an arbitrary NFA. By Theorem 3, we construct a 3-layer ReLU network with the following components:

- Layer 1: Computes the ϵ-closure of the start state q_0, resulting in the initial state vector s_0^ϵ.
- Layer 2: For any input string $x = x_1 x_2 \dots x_T$, the network performs T sequential transition updates followed by ϵ-closures, using unrolled matrix multiplications and ReLU.
- Layer 3: Checks if any state in F is active using an inner product with the final state vector s_T.

The full network requires: - Width: $\mathcal{O}(n)$ neurons to encode all NFA states. - Depth: $\mathcal{O}(T+2n)$, accounting for input transitions and ϵ-closures. - Fixed weights and ReLU activations — no learned parameters are necessary for this constructive equivalence.

Because every regular language is defined by some NFA $\mathcal{A}$, and $\mathcal{A}$ can be encoded by such a ReLU network, the class of ReLU networks is NFA-complete with respect to language recognition.

A.7 Proof of Lemma 3: Gradient Flow Preserves NFA Semantics

Let us denote the ReLU network f_θ as a composition of layers simulating the NFA's transitions. Each layer corresponds to a symbol in the input string and applies a transformation based on the transition matrix T^{x_t} for symbol $x_t \in \Sigma$.

Assume that the initial network parameters θ_0 are set such that f_{θ_0} exactly simulates $\mathcal{A}$, as established in Theorem 3. During training, the parameters are updated via gradient descent:

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta \mathcal{L}(\theta_k),$$

where η is the learning rate.

Suppose that during training, the updates to θ are constrained to preserve the sparsity pattern of the transition matrices T^{x_t}, meaning that the positions of non-zero entries remain fixed, and only their values are adjusted. This constraint ensures that the structural representation of the NFA's transitions is maintained.

Under these conditions, for any input string $x = x_1 x_2 \dots x_T$, the network's computation proceeds through the sequence of layers corresponding to each x_t, applying the ReLU-activated transition matrices. Since the structure of these matrices remains consistent with the NFA's transition function δ, and the network is trained to match the NFA's acceptances on $\mathcal{D}$, the network's behavior aligns with that of $\mathcal{A}$ on $\mathcal{D}$.

Furthermore, because the structural constraints prevent the introduction of transitions not present in δ, and the training data $\mathcal{D}$ encompasses representative examples from $L(\mathcal{A})$, the network generalizes this behavior to all inputs in Σ^*, preserving the NFA's semantics.

A.8 Proof of Theorem 4: Learnability of NFA via Supervised ReLU Networks

Let f_θ be a ReLU network initialized with weights θ_0 structured as follows: - Each input symbol $x \in \Sigma$ corresponds to a transition matrix T^x embedded into a sparse linear layer. - The initial activation vector s_0 encodes the start state q_0. - The final classification is computed as $\hat{y} = \sigma(f^\top s_T)$, where $f \in \{0,1\}^n$ is the indicator of accepting states.

Training proceeds via gradient descent:

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta \mathcal{L}(f_\theta(x^{(i)}), y^{(i)}),$$

with binary cross-entropy loss. Assume: 1. The loss is minimized to zero on $\mathcal{D}$. 2. Weight updates maintain the zero-patterns of θ_0 (structure-preserving).

Then, for each $x^{(i)}$, the activations s_T must be aligned with the correct DFA subset constructed by $\mathcal{A}$, or else $\hat{y} \neq y^{(i)}$. Because the loss is minimized, this condition holds for all $(x^{(i)}, y^{(i)})$ in the training set.

Now, consider any unseen $x \in \Sigma^*$: - Since f_θ retains the transition structure of $\mathcal{A}$ and weights have only been refined (not added arbitrarily), it generalizes to x exactly as $\mathcal{A}$ would.

Thus, f_θ simulates the NFA $\mathcal{A}$ exactly after training.

A.9 Proof of Theorem 5: Equivalence of ReLU Networks and NFA Recognizers

(Forward Direction) Given a regular language $\mathcal{L}$, let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be an NFA recognizing $\mathcal{L}$ with $n = |Q|$ states. By Theorem 3, we construct a feedforward ReLU network f_θ that:

- Encodes state activations with n ReLU neurons,
- Applies ϵ-closures via repeated matrix multiplication and ReLU (Lemma 2),
- Computes transitions via ReLU-layered matrix products (Theorem 2),
- Outputs 1 if any accepting state is active.

The network simulates the behavior of $\mathcal{A}$ exactly on any $x \in \Sigma^*$, so $f_\theta(x) = 1$ if and only if $x \in \mathcal{L}$.

(Reverse Direction) Let f_θ be a ReLU network constructed with:

- An input-dependent sequence of symbol-conditioned transition matrices $\{T^{x_i}\}$,
- ReLU-activated state vectors $s_t = \text{ReLU}(T^{x_t} s_{t-1})$,
- An acceptance condition based on overlap with an indicator vector f for accepting states.

Construct an NFA $\mathcal{A}' = (Q, \Sigma, \delta', q_0, F')$ where:

- $Q = \{q_1, \ldots, q_n\}$ corresponds to neurons in f_θ,
- $\delta'(q_i, x) = \{q_j : T^x_{ji} > 0\}$,
- F' contains all q_j for which $f_j = 1$,
- q_0 is the initial state corresponding to s_0.

Since f_θ deterministically applies these transitions and accepts iff an output neuron indexed by F' is activated, the language recognized by f_θ matches that of $\mathcal{A}'$.

Conclusion: This establishes a bijective, symbolic correspondence between finite-state NFA recognizers and ReLU networks simulating them via feedforward computation.