LEARNING WEIGHTED AUTOMATA FROM RECURRENT NEURAL NETWORKS USING HIDDEN-STATE PARTITIONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recurrent neural networks (RNNs) are able to learn complex sequential behaviours, but their hidden-state representations make them difficult to interpret. This paper presents a new method for learning Weighted Finite Automata (WFAs) from RNNs. The method works within the standard L^* active learning framework, where the main challenge is in comparing a hypothesised WFA with an RNN. The key novelty lies in how this comparison is realised, thereby answering equivalence queries. This is achieved through a partitioning of the RNN hidden-state space that is iteratively refined until each partition has been sufficiently explored and has small relative variance. The partitioning guides the search for counterexamples between the RNN and the learned WFA and provides a practical criterion for terminating the algorithm when no significant discrepancy is identified. Theoretical guarantees on the correctness and termination of the extraction procedure are provided under appropriate conditions. An empirical comparison of the approach with existing methods for WFA extraction from RNNs is given, and the experiments show that the proposed method efficiently extracts compact WFAs with high fidelity to the underlying RNN while generally achieving substantially lower extraction times than existing approaches.

1 INTRODUCTION

RNNs are widely used for modelling sequential data, including language modelling and other sequence-processing tasks. However, understanding the internal behaviour remains challenging owing to their black-box nature (Jacobsson, 2005; Lipton, 2018). Representing the internal dynamics of RNNs using automata has a long history of research and provides a means of improving their interpretability. While the behaviour of RNNs with Boolean outputs is well studied using Deterministic Finite Automata (DFA) (Kleene, 1956; Siegelmann & Sontag, 1992; Giles et al., 1992; Zeng et al., 1993; Omlin & Giles, 1996; Weiss et al., 2018), the RNNs that map input strings to real-valued outputs have received comparatively less attention. Weighted Finite Automata (WFA) provide a natural finite-state representation for approximating such quantitative behaviours. A theoretical connection between RNNs and WFAs has been established for second-order RNNs, which are expressively equivalent to WFAs (Rabusseau et al., 2019). Previous research has also empirically demonstrated that WFAs can provide an accurate approximation of the input-output behaviour of RNNs (Ayache et al., 2018; Eyraud & Ayache, 2024; Okudono et al., 2020).

Formally, a WFA defines a real-valued function over strings, which are characterised by a finite-rank Hankel matrix (Berstel & Reutenauer, 1988; Droste et al., 2009). This connection with Hankel matrices provides the theoretical basis for spectral learning methods (Balle et al., 2014). Existing approaches for learning WFAs from RNNs mainly include spectral learning and clustering-based methods (Bailly et al., 2009; Lecorvé & Motlicek, 2012; Balle et al., 2014; Balle & Mohri, 2015; Ayache et al., 2018; Zhang et al., 2021; Wei et al., 2022). Active learning provides an alternative in which a hypothesised automaton is incrementally refined using information obtained from membership and equivalence queries, as characterised by the L^* algorithm (Angluin, 1987). When the target model is an RNN, membership queries can be answered by evaluating the RNN on a given input string. Equivalence queries are considerably more challenging, as determining formal equivalence between an RNN and a WFA is undecidable in general (Marzouk & de la Higuera, 2020).

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.

Therefore, a practical mechanism is required to approximate equivalence queries and identify counterexamples between a hypothesised WFA and the RNN.

Active learning has been investigated for WFA extraction from RNNs (Balle & Mohri, 2015; Weiss et al., 2019; Okudono et al., 2020). More recently, Okudono et al. (2020) introduced an active learning framework in which equivalence queries are approximated using a regression-based approach. However, this requires learning and refining a continuous regression mapping during extraction, which introduces additional computational overhead leading to excessive extraction times. Furthermore, unlike DFA extraction, where discretisation of the continuous RNN state space has been investigated to construct abstraction models for equivalence checking (Weiss et al., 2018; Wickramasinghe et al., 2026), the use of state-space discretisation for WFA extraction has received limited attention (Wickramasinghe et al., 2025). Such an abstraction can provide a finite representation of the continuous RNN state space, allowing it to be explored systematically during equivalence queries and providing a way to identify regions where the behaviour of the hypothesis WFA differs from that of the RNN.

To address this gap, a novel method for learning a WFA from an RNN using the weighted variation of Angluin (1987)'s L^* active learning framework by Balle & Mohri (2015) is presented. Membership queries are answered by evaluating the RNN, while spectral learning is used to construct the WFA hypothesis. Equivalence queries are addressed through a discrete abstraction of the RNN hidden-state space, constructed and iteratively refined using hyperplane-based partitioning. This is the first approach to WFA extraction from RNNs that combines active learning with discretisation of the RNN hidden-state space for equivalence queries. The resulting abstraction provides a lightweight mechanism for guiding the search for counterexamples and reducing the computational cost for equivalence queries.

The main contributions of this work are as follows:

- A new equivalence query procedure is introduced for determining whether there is a significant discrepancy between a hypothesised WFA and an RNN. This uses hyperplane-based partitioning of the RNN hidden state space, which is iteratively refined based on the variance of the hidden states within each partition. The resulting partitioning guides the search for counterexamples between the RNN and the hypothesised WFA. When a partition has been sufficiently explored, without triggering further refinements, the behaviour of other states within that region is assumed to be sufficiently similar to those already observed. Repeated visits to such partitions are controlled, allowing the search to focus on less explored or behaviourally different regions of the hidden state space.
- Theoretical properties of the extraction procedure under the stated conditions are established, and the impact of the algorithm's hyperparameters on WFA learning is investigated.
- An empirical evaluation of the approach using LSTM RNNs trained on data sampled from randomly generated WFAs is conducted, assessing the fidelity of the extracted WFAs and the computational efficiency of the procedure, as well as comparing the proposed approach with an existing WFA extraction method.

The results show that the proposed approach extracts WFA with high fidelity compared to the target RNNs while generally requiring lower extraction times than the existing approach. The remainder of the paper presents background, introduces the proposed extraction method and its theoretical properties, and evaluates its behaviour experimentally.

2 Related work

Spectral learning of WFAs. WFA compute functions from strings to real numbers, commonly known as a rational series (Berstel & Reutenauer, 1988; Droste et al., 2009). Their connection to Hankel matrices provides the theoretical foundation for spectral learning: a function $f : \Sigma^* \to \mathbb{R}$ can be represented by a WFA if and only if its Hankel matrix $\mathcal{H}_f$ has finite rank, with the rank corresponding to the number of states in a minimal WFA (Carlyle & Paz, 1971; Fliess, 1974). Early spectral methods were applied to sequence models, including Hidden Markov Models (Hsu et al., 2012) and then used for learning weighted automata (Bailly et al., 2009). Balle et al. (2014) provided a general spectral learning framework that recovers a linear representation of a WFA from a

finite Hankel sub-block using a singular value decomposition (SVD)-based low-rank factorisation. Spectral learning has subsequently been applied to extracting WFAs from RNNs (Ayache et al., 2018). However, the quality of the resulting WFA depends on the prefixes and suffixes selected for the finite Hankel basis; if this basis is fixed in advance, any behaviours that are not captured cannot later be incorporated into the hypothesis (Ayache et al., 2018; Balle et al., 2014). This motivates the need for an incremental method for refining the basis, repeatedly extending to capture previously under-represented behaviour.

State-space abstraction. Automata extraction from RNNs has traditionally focused on clustering or quantisation as stand-alone methods. Such approaches have been investigated for both DFA and WFA extraction (Giles et al., 1992; Omlin & Giles, 1996; Cechin et al., 2003; Huynh & Reggia, 2012; Lecorvé & Motlicek, 2012; Du et al., 2019; Zhang et al., 2021; Wei et al., 2022). However, selecting an appropriate number of clusters and scaling the abstraction to high-dimensional state spaces remain challenging (Cechin et al., 2003; Huynh & Reggia, 2012; Du et al., 2019). These approaches use clustering or quantisation to construct the states or transition structure of the extracted automaton. In contrast, the state-space abstraction in the proposed approach is used only to guide equivalence queries; the states of the WFA are obtained through spectral learning.

Active learning. Active learning methods construct an automaton from an unknown teacher that provides answers to membership and equivalence queries (Angluin, 1987). The learner uses these answers to iteratively refine a hypothesised automaton until the teacher says that it is equivalent to the target. For DFA extraction from RNNs, Weiss et al. (2018) combines active learning with a state-space abstraction that is refined during equivalence queries. Related abstraction methods have been investigated later on for RNN-to-DFA extraction (Wickramasinghe et al., 2026). In the weighted setting, Weiss et al. (2019) also use active learning to extract probabilistic deterministic finite automata (PDFAs), by querying the target network's output distribution, treating it as a black-box model. (Balle & Mohri, 2015) extend active automata learning to WFAs by combining it with spectral learning, and Okudono et al. (2020) applies this framework to RNN extraction using regression over continuous hidden-state representations to guide equivalence queries. This showed that the RNN hidden-state information can be exploited to search for counterexamples, but requires constructing and refining a continuous regression model during extraction. In contrast, this work constructs a discrete, iteratively refined partitioning of the RNN hidden state space to guide equivalence queries, extending the L^*-based framework in a weighted setting.

3 Background

A finite alphabet is a non-empty finite set of elements. A string over an alphabet Σ is a finite sequence of elements of Σ, and the set of all strings over Σ is denoted by Σ^*. The empty string of length 0 is denoted by ε. The length of a string $w \in \Sigma^*$ is denoted by $|w|$. Given two strings u and v, their concatenation is denoted by uv. In that case, u is called a prefix and v a suffix of uv. A set $\mathcal{P}$ of strings is said to be prefix-closed, respectively suffix-closed, if for all strings w in $\mathcal{P}$, all the prefixes, respectively all the suffixes, of w are also in $\mathcal{P}$.

Recurrent Neural Networks. Recurrent Neural Networks (RNNs) are a type of neural network introduced to handle sequential data. An RNN takes a string $w \in \Sigma^*$ as an input and processes it sequentially, passing each element through feedback loops one at a time. Several types of RNNs exist, such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) (Hochreiter & Schmidhuber, 1997; Chung et al., 2014). In this paper, we consider LSTMs mapping each string of Σ^* to a real-valued output.

Definition 1. An *RNN* R over alphabet Σ is a tuple $R = \langle h_0, g_R, f_R \rangle$, where $h_0 \in \mathbb{R}^d$ for some $d \in \mathbb{N}$ is the initial state, $g_R : \mathbb{R}^d \times \Sigma \to \mathbb{R}^d$ is the transition function and $f_R : \mathbb{R}^d \to \mathbb{R}$ is the output function. The *hidden states* of R are defined recursively by $h(\varepsilon) = h_0$ and for $w \in \Sigma^*$ and $\sigma \in \Sigma$, $h(w\sigma) = g_R(h(w), \sigma)$. The output for w is given by $R(w) := f_R(h(w))$. The *hidden state space* of R is $H_R = \{h(w) : w \in \Sigma^*\} \subseteq \mathbb{R}^d$.

Weighted Finite Automata and Rational Series. Weighted finite automata (WFA) are finite automata with transitions weighted by real values, computing functions mapping strings to reals. The value a WFA assigns to a string is the sum of the weights of all paths travelled by the string.

Definition 2. A *Weighted Finite Automaton* over alphabet Σ with n states is a tuple $A = \langle \alpha_0, (A_\sigma)_{\sigma\in\Sigma}, \alpha_\infty \rangle$, where $\alpha_0, \alpha_\infty \in \mathbb{R}^n$ are the initial and final weight vectors, and each $A_\sigma \in \mathbb{R}^{n\times n}$ is the transition matrix associated with $\sigma \in \Sigma$. The *output* for input string $w = \sigma_1\sigma_2\cdots\sigma_k$ (where $\sigma_i \in \Sigma$) is $A(w) = \alpha_0^\top A_{\sigma_1} A_{\sigma_2} \cdots A_{\sigma_k} \alpha_\infty$.

A function $f : \Sigma^* \to \mathbb{R}$ is called a rational series (Berstel & Reutenauer, 1988; Droste et al., 2009) if it is computed by a WFA, that is, if there exists a WFA A such that $f(w) = A(w)$ for all $w \in \Sigma^*$.

The extraction framework presented in this work uses *active learning* to search for a WFA that can model the behaviour of an RNN. Active learning is a framework involving a teacher and a learner. The teacher holds a model computing a function f (the RNN in this case) and the learner iterates through two types of queries to learn it: 1) membership queries: the learner chooses an input w for f (a string in this work) and the teacher has to give the value $f(w)$. 2) equivalence queries: the learner constructs a model computing a function f' and asks if $f = f'$. If not, the teacher gives an input w such that $f(w) \neq f'(w)$. Such a w is called a counterexample.

(Angluin, 1987) gives the first polynomial-time algorithm to learn a (non-weighted) automaton actively. For WFA, polynomial-time algorithms have been developed to learn rational series. The general framework for these algorithms is recapped here as it will be used later in the paper. It constructs WFA from their Hankel matrices.

Definition 3. Let $f : \Sigma^* \to \mathbb{R}$. The *Hankel matrix* of f is a bi-infinite matrix $\mathcal{H} \in \mathbb{R}^{\Sigma^*\times\Sigma^*}$, whose entries are defined as $\mathcal{H}(u, v) = f(uv)$ for any $u, v \in \Sigma^*$.

Given a pair $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ where $\mathcal{P}$ and $\mathcal{S}$ are sets of strings of Σ^*, the finite Hankel sub-matrix $\mathcal{H}_\mathcal{B}$ is obtained by restricting the rows of $\mathcal{H}$ to $\mathcal{P}$ and its columns to $\mathcal{S}$. The pair $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ is referred to as a basis. Given a basis $\mathcal{B}$, the learner can compute $\mathcal{H}_\mathcal{B}$ using membership queries.

Definition 4. A basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ is said to be *closed* if $\mathcal{P}$ is a prefix-closed set and $\mathcal{S}$ is a suffix-closed set. Let $\mathcal{P}\Sigma = \{p\sigma | p \in \mathcal{P}, \sigma \in \Sigma\}$ and $\mathcal{B}'$ be the basis $(\mathcal{P}\Sigma, \mathcal{S})$. Then $\mathcal{B}$ is said to be *complete* if:

$$\operatorname{rank}(\mathcal{H}_\mathcal{B}) = \operatorname{rank}\left(\begin{bmatrix}\mathcal{H}_\mathcal{B} \\ \mathcal{H}_{\mathcal{B}'}\end{bmatrix}\right) \tag{1}$$

Algorithms for learning WFA work as follows: Starting from a basis $\mathcal{B} = (\{\varepsilon\}, \{\varepsilon\})$, the basis is extended, using membership queries, until it is closed and complete. Provided that the basis $\mathcal{B}$ is closed and complete, the learner can construct a WFA from $\mathcal{H}_\mathcal{B}$ using, for example, the Singular Value Decomposition (SVD)-based low-rank factorisation and the Moore–Penrose pseudo-inverse as in (Ben-Israel & Greville, 2003; Golub & Loan, 2013; Balle et al., 2014). Once this WFA is constructed, an equivalence query is performed. If a counterexample is given, the basis $\mathcal{B}$ is extended again, taking into account that counterexample until it is once more closed and complete, and the process is iterated until no counterexample can be found.

4 Extracting WFA from RNN

This section presents the algorithm developed for learning a WFA from an RNN. The specific original contribution of this work is explained, as are the guarantees for termination and correctness that are obtained from it. A running example is used to illustrate the process. The pseudo-code and proofs are left to the Appendix A and B.

Example 1. Let $\Sigma = \{a, b\}$ be the alphabet and consider the WFA A defined by

$$\boldsymbol{\alpha}_0 = [1 \quad 0 \quad 0] \qquad \boldsymbol{\alpha}_\infty = [0.302 \quad 0.147 \quad 0.092]$$

and the transition matrices

$$A_a = \begin{bmatrix} 0 & 0.53 & 0.47 \\ 0.62 & 0.38 & 1 \\ 0 & 0.30 & 0.70 \end{bmatrix} \qquad A_b = \begin{bmatrix} 0 & 0.49 & 0.51 \\ 0.48 & 0.52 & 0 \\ 0 & 0.30 & 0.70 \end{bmatrix}$$

An RNN R was trained using the data generated by A (achieving a test MSE of 2.54×10^{-9} against A, indicating that R closely approximates the behaviour of A). The new algorithm from Section 4.1 was run on R, with the expectation to obtain a WFA computing a function close to the one computed by A. There is no expectation that the algorithm returns the same syntactic WFA A but rather a semantically equivalent one.

4.1 ALGORITHM

The algorithm follows the standard active learning framework for WFA, as described in the previous section, with a new way to handle the crucial equivalence queries, given in Step 2 below. Given an RNN computing a function f, the algorithm proceeds in two stages, starting from a basis $\mathcal{B} = (\{\varepsilon\}, \{\varepsilon\})$.

Step 1. Membership queries are used to extend $\mathcal{B}$ until it is closed and complete, and a so-called hypothesis WFA A is constructed from the part $\mathcal{H}_\mathcal{B}$ of the Hankel matrix of f, following (Balle & Mohri, 2015). A *rank tolerance* parameter t is used to restrict the size of the hypothesised WFA.

Step 2. Equivalence queries check the hypothesis WFA against the RNN teacher, once it has been constructed in Step 1. If it is close enough (for a specific notion explained below), it is returned; otherwise, a counterexample is returned, which guides further membership queries to construct a better hypothesis WFA. This second step is an original contribution of this paper. While equivalence queries are in theory doable when comparing two WFAs (WFA equivalence is decidable), the RNN under consideration in this work might not compute a rational series, and there is no canonical way to compare a hypothesis WFA and an RNN. Here a different, and novel, way to perform the equivalence queries between the hypothesis WFA and the RNN is given. This new method is also more computationally efficient.

Error tolerance. An *error tolerance* threshold e is introduced. The outputs of the RNN R and the hypothesised WFA A are considered equivalent for a string w if $|A(w) - R(w)| \leq e$. If this condition holds, w is not considered a counterexample. Otherwise, w can be considered a counterexample and used to refine the hypothesised WFA.

Partition-based exploration. The equivalence query requires identifying counterexamples. Since an exhaustive search over all strings in Σ^* is obviously not feasible, this work introduces a guided search for counterexamples via a partitioning-based exploration strategy over the hidden state space of the RNN.

Let $R = \langle h_0, g_R, f_R \rangle$, and H_R be its hidden-state space. At all times, a partition of H_R is kept. In practice, as H_R is not known, it is actually a partition of $\mathbb{R}^d$ that is kept. Initially, the partition has only one element: $\mathbb{R}^d$. Additionally, for every set P in the partition, a set $\hat{P}$ of elements of $\mathbb{R}^d$ corresponding to the hidden states in P that have already been visited is recorded. At first, $\hat{\mathbb{R}^d} = \{h(\varepsilon)\} = \{h_0\}$. A queue of strings to be considered is also kept at all times, initially $\mathcal{Q} = \{\varepsilon\}$. When a string w is popped from the queue to be treated, the strings $w\sigma$ are added to the queue for all letters σ, unless some termination parameters have been reached as detailed below.

Treatment of the front element w of the queue $\mathcal{Q}$ and variance. Firstly, if w is a counterexample, then it is returned and used to refine the basis. The counterexample is then dealt with as in Step 1. Otherwise, w might trigger a partition refinement as follows: let $h(w)$ be the hidden state for w and P the set of the current partition of $\mathbb{R}^d$ satisfying $h(w) \in P$. The variance of $\hat{P}$ is now used to decide whether P should be refined.

Definition 5. Let $\hat{P} = \{h_1, h_2, \ldots, h_m\}$ be a finite subset of $\mathbb{R}^d$. The *centroid* of $\hat{P}$ is defined as $\mu = \frac{1}{m}\sum_{j=1}^{m} h_j$. The *variance* of $\hat{P}$ is defined as the mean squared Euclidean distance between each h_i and μ, that is, $\mathrm{Var}(\hat{P}) = \frac{1}{m}\sum_{j=1}^{m} \|h_j - \mu\|_2^2$.

After adding w, the increase in the variance of $\hat{P}$ is checked. If the following two conditions are satisfied, P is refined:

- $\mathrm{Var}(\hat{P} \cup \{h(w)\})$ is large enough in absolute value, i.e. $\mathrm{Var}(\hat{P} \cup \{h(w)\}) > \tau$. The *absolute variance threshold* parameter τ specifies the minimum absolute variance required for refinement, so the updated partition's variance is sufficiently large.
- $\mathrm{Var}(\hat{P} \cup \{h(w)\})$ has increased enough, i.e. $\mathrm{Var}(\hat{P} \cup \{h(w)\}) > (1 + v)\mathrm{Var}(\hat{P})$. The *variance tolerance* parameter v determines whether the variance of a partition has increased significantly after observing a new hidden state.

If so, the hidden state $h(w)$ is added to $\hat{P}$ and P is refined into P_1 and P_2 as follows: Find w' such that $h(w') \in \hat{P}$ and $\|h(w) - h(w')\|_2$ is maximal. Compute the median hyperplane of $h(w)$ and $h(w')$ and denote by S_1 and S_2 the two half-spaces created, with S_1 closed and S_2 open, and define $P_1 = S_1 \cap P$ and $P_2 = S_2 \cap P$. Finally, define $\hat{P}_1 = \hat{P} \cap P_1$ and $\hat{P}_2 = \hat{P} \cap P_2$.

Maximal visits and lengths. To ensure termination, two other parameters are used: a *maximal number of visits* M a set in the partition can be visited and a *maximal string length* $L_{\max}$. When treating w as above, the number of visits to P is incremented. Once P has been visited M times without requiring further refinement, the behaviour within the partition is assumed to be sufficiently represented by the states already explored. Thus, if w reaches that partition, its successors $w\sigma$ are not added to the queue, terminating further exploration through extensions of w. Similarly, a string is not extended once its length reaches $L_{\max}$.

Running example. Suppose the parameters are $e = t = \tau = v = 0.01$ and $M = 25$. Starting from $\mathcal{B} = (\{\varepsilon\}, \{\varepsilon\})$, the initial Hankel matrix $H_{\mathcal{B}}$ has rank 1 and Step 1 produces a one-state hypothesis WFA A_1. The first equivalence query then evaluates A_1 against R. The exploration queue is initially $\{\varepsilon\}$ and the partition $\Gamma = \{\mathbb{R}^d\}$.

Processing ε adds $\{a, b\}$ to the queue. Each string is then popped and processed in order. Processing a subsequently adds $\{aa, ab\}$ to the queue. Processing b adds $\{ba, bb\}$. The next string aa is then popped from the queue and identified as the first counterexample, since $|R(aa) - A_1(aa)| > e$. Adding its prefixes and suffixes to the basis $\mathcal{B} = (\{\varepsilon, a, aa\}, \{\varepsilon, a, aa\})$ gives a rank-3 Hankel matrix and Step 1 produces a three-state hypothesis WFA A_2. A new equivalence query is then performed for A_2. Since the current exploration state is continued to the next equivalence query, the search continues from the remaining strings rather than starting from ε. When ab is processed, the variance condition triggers the refinement of Γ, splitting it into $\Gamma = \{P_1, P_2\}$. As ab is processed, this adds $\{aba, abb\}$ to the queue.

The next counterexample encountered is abb for which $|R(abb) - A_2(abb)| > e$. Adding the prefixes and suffixes to the basis gives $\mathcal{B} = (\{\varepsilon, a, aa, ab, abb\}, \{\varepsilon, a, b, aa, bb, abb\})$. Although additional prefixes and suffixes are introduced, the rank of the resulting Hankel basis, determined using the rank tolerance $t = 0.01$, remains 3. Therefore, no additional WFA state is required, and Step 1 produces another three-state hypothesis WFA A_3.

The equivalence query is repeated for A_3. Exploration continues over the remaining strings in the queue without encountering a counterexample or triggering partition refinement. Finite M and L_{max} ensure that eventually the exploration queue become empty and the process terminates, returning A_3 as the final learned WFA. In particular, a partitioning that does not need to be refined, along with M means that the counterexample search can be dramatically pruned, even with a large L_{max}.

4.2 Termination and Correctness: some guarantees

This section gives theoretical results on guarantees on the algorithm. Strong guarantees are ensured in the idealised cases where the parameters take extreme values (error tolerance 0, for example). Parameters M and $L_{\max}$ are allowed to take infinite values (no forced termination). When not specified in the statement of the results, parameters can take any values. To express these guarantees, consider the sequence of hypothesis WFA created within the algorithm, which aim to converge towards a good approximation of the RNN.

Given an RNN R, consider the sequence of hypothesis automata generated by the algorithm under some specific parameters as the ones created in Step 1 of the algorithm, in the order they are created.

Remark 1. The sequence of hypothesis automata may be finite or infinite. If it is infinite, the algorithm does not terminate. However, the algorithm might not terminate but the sequence be finite. If the algorithm terminates, the sequence is finite and the last element is returned.

Proposition 1. Assume rank tolerance $t = 0$ and R computes a rational series. Then the sequence of hypothesis automata generated by the algorithm is finite.

Proposition 2. Suppose $L_{\max} = M = \infty$ and let $e \geq 0$ be the error tolerance. Assume rank tolerance $t = 0$ and R computes a rational series. Then the final hypothesis automaton A generated

by the algorithm satisfies that for all strings $w \in \Sigma^*$

$$|A(w) - R(w)| \leq e$$

Corollary 1. Suppose $L_{\max} = M = \infty$. Assume error tolerance $e = 0$, rank tolerance $t = 0$, and that R computes a rational series, then the last automaton in the sequence of hypothesis automata is the minimal WFA realising the function computed by R.

Proposition 3. Assume rank tolerance $t = 0$ and R computes a rational series. If either the maximum exploration length $L_{\max}$ or the maximum number of visits per partition M is finite, then the algorithm terminates.

Remark 2. If both $L_{\max} = \infty$ and $M = \infty$, even after a correct automaton has been found and no counterexample exists, the exploration queue may grow by generating increasingly longer strings. Consequently, although no additional hypothesis automata are produced, the overall extraction algorithm need not terminate.

Proposition 4. Assume the algorithm terminates and returns WFA A. Let e be the error tolerance. Then for every string $w \in \Sigma^*$, there exists a string $w' \in P_w$ (where P_w is the set of the partition containing $h(w)$ at the end of the procedure) such that

$$|A(w') - R(w')| \leq e$$

5 Experimental Results

The algorithm in Section 4.1 was implemented in Python 3.9.6. For empirical evaluations, two-layer LSTM-RNNs with a hidden dimension of 16 and 50 were trained to approximate randomly generated WFAs with $|Q|$ states and alphabet Σ. The WFA parameters α_0, α_∞ and A_σ were randomly initialised, with α_0 and each row of A_σ normalised to sum to one, and α_∞ entries bounded between zero and one. Unless otherwise specified, α_0 and A_σ were generated with 30% zero entries.

The datasets contained 500,000 strings of maximum length 20 generated using systematic enumeration, with 90% used for training, 5% for validation and 5% for testing. The models were trained using PyTorch 2.6.0, for up to 50 epochs with a learning rate of 0.001 and early stopping. All RNNs achieved an $\mathrm{MSE} < 5 \times 10^{-7}$ against their target WFA. Extraction used $e = 0.01$ and $L_{max} = 20$, while $t = v = \tau = 0.01$ were selected through hyperparameter tuning described in the Appendix D. Different values of M were evaluated. For each $(|\Sigma|, |Q|)$ configuration, 10 WFA models were trained with different random seeds, and extraction was performed independently on each model. The reported results are aggregated across these runs.

The empirical evaluation investigated the fidelity and efficiency of the extracted WFA in reproducing the input–output behaviour of the RNN, its performance against an existing WFA extraction method, and the effectiveness of the equivalence-query procedure in exploring the RNN hidden-state space.

5.1 Fidelity and the efficiency of the extraction

The *fidelity* of the extracted WFA was evaluated using Mean Squared Error (MSE): $\mathrm{MSE}(A, R) = \frac{1}{N}\sum_{i=1}^{N}\left(A(w_i) - R(w_i)\right)^2$, and Normalised Root Mean Squared Error (NRMSE):

$$\mathrm{NRMSE}(A, R) = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(A(w_i) - R(w_i)\right)^2}}{\max_i R(w_i) - \min_i R(w_i)}$$

NRMSE normalises the root mean squared error by the range of the RNN outputs and is reported as a percentage (Goudarzi et al., 2014). Lower MSE and NRMSE values indicate higher fidelity. Efficiency was measured using the extraction time, with the number of counterexamples, and extracted WFA states providing additional characterisation of the extraction process. Table 1 reports averages over ten runs for each $(|\Sigma|, |Q|)$ configuration on the hold out test set. Results for the proposed method are denoted by HYP, with $M \in \{5, 10, 25\}$ representing different maximum partition visit thresholds. The regression-based baseline (Okudono et al., 2020), using their best-performing setting with a concentration threshold of 5, is denoted by REG.

For HYP, increasing M generally improves fidelity, particularly from $M = 5$ to $M = 10$, with smaller gains from $M = 10$ to $M = 25$. For example, for $(|\Sigma|, |Q|) = (8, 8)$, NRMSE decreases

Table 1: Extraction fidelity and efficiency for HYP with $M \in \{5, 10, 25\}$ and REG baseline. Fidelity entries report MSE / NRMSE, where MSE is in units of 10^{-5} and NRMSE is reported as a percentage. Time is reported in seconds. $N_{\text{cex}}/|\hat{Q}|$ reports the average number of counterexamples and extracted WFA states, respectively. For HYP, $\tau = e = v = t = 0.01$; for REG, $e = t = 0.01$.

Config.		HYP5		HYP10		HYP25			REG		
$\|\Sigma\|$	$\|Q\|$	Fidelity	Time	Fidelity	Time	Fidelity	Time	$N_{\text{cex}}/\|\hat{Q}\|$	Fidelity	Time	$N_{\text{cex}}/\|\hat{Q}\|$
2	3	42.96 / 29.40	0.06	29.52 / 22.00	0.08	7.99 / 13.60	0.19	2.40 / 3.70	0.98 / 5.10	0.93	3.60 / 4.10
4	6	7.70 / 5.40	0.35	2.01 / 3.00	0.43	0.59 / 1.80	0.66	4.10 / 6.00	0.45 / 1.60	11.35	5.50 / 6.30
5	8	7.47 / 4.80	0.89	1.12 / 2.10	1.37	0.76 / 1.70	1.19	4.70 / 7.00	0.77 / 1.70	98.88	7.50 / 7.40
8	8	11.96 / 4.30	2.41	0.56 / 1.20	3.15	0.45 / 1.10	3.96	6.30 / 8.60	0.28 / 0.80	858.53	9.80 / 9.40
8	10	3.75 / 5.22	1.83	0.63 / 2.10	2.58	0.54 / 1.98	2.88	5.60 / 7.30	0.68 / 2.23	100.14	8.90 / 8.10
10	10	5.39 / 4.80	2.17	0.95 / 2.20	3.31	0.34 / 1.50	5.92	7.10 / 8.30	0.36 / 1.50	498.07	9.00 / 9.40
10	12	3.43 / 5.20	1.09	1.13 / 3.20	2.85	0.56 / 2.30	5.20	6.20 / 7.30	0.52 / 2.20	186.95	8.50 / 8.80
10	15	1.91 / 6.10	0.64	1.21 / 5.00	1.17	0.86 / 4.20	2.52	4.60 / 5.70	0.87 / 4.20	67.54	7.20 / 6.80
15	15	2.20 / 5.60	1.64	1.04 / 3.90	4.21	0.74 / 3.40	6.67	5.80 / 7.20	0.53 / 2.90	185.44	9.70 / 9.45
15	20	1.84 / 7.60	0.43	1.42 / 6.70	0.74	0.86 / 5.50	1.78	2.90 / 3.70	0.80 / 5.30	18.53	4.90 / 4.70

from 4.3% for HYP5 to 1.2% for HYP10, while increasing M further to 25 reduces it to 1.1%. Similar pattern can be observed for other configurations.

HYP25 achieves NRMSE between 1.1% and 5.5% across the reported configurations, excluding $(2, 3)$, and provides the best fidelity among evaluated M. The improvement in fidelity with increasing M is accompanied by a small increase in computational cost. However, HYP25 completes all reported extraction runs in less than 6.67 seconds. Runtime does not monotonically increase with $|Q|$, indicating that target WFA size alone does not determine the difficulty of extracting the behaviour learned by the RNN. Overall, the results show that fidelity increases with M and controls the trade-off between fidelity and the extraction time.

5.2 Comparison

Consider Table 1 again, comparing HYP25 against REG. REG achieves comparable errors. The difference is evident in the extraction times. HYP25 requires less than 6.67 seconds across the displayed configurations, compared with 0.93–858.53 seconds for REG. For $(10, 10)$, both methods obtain an NRMSE of 1.5%, while HYP25 requires 5.92 seconds compared with 498.07 seconds for REG. Similarly, for $(8, 8)$, REG has slightly lower NRMSE (0.8% versus 1.1%), but requires 858.53 seconds compared with 3.963 seconds for HYP25. A Wilcoxon signed-rank test (Wilcoxon, 1945) found no statistically significant difference in NRMSE between HYP25 and REG ($p = 0.106$), while the difference in extraction time was statistically significant ($p = 0.002$). HYP25 also requires fewer counterexamples than REG across all the configurations, leading to smaller WFA.

5.3 Robustness and State-Space Exploration

To examine robustness, experiments were repeated with a hidden-state dimension of 50, using RNNs trained on 500,000 strings generated using either systematic enumeration, as in the earlier experiment, or random length-stratified sampling. Again, 10 runs were performed, and results are from the hold-out test set. Results are given in Table 2. Systematic enumeration results in a higher concentration of shorter sequences, particularly for larger alphabets, whereas the stratified sampling provides representation across sequence lengths. Both strategies produce comparable MSE and number of partitions, although the length-stratified random sampling results in longer extraction times.

How the equivalence queries (EQ) explore the hidden-state space is investigated. An independent reference set of $25,000$ hidden states, obtained from strings randomly sampled across sequence lengths, was clustered using k-means, with each k selected using the Elbow method (Kodinariya & Makwana, 2013). Each EQ-visited hidden state was assigned to its nearest reference cluster and considered within the reference region if it fell within the cluster radius. Two measures are reported: *cluster coverage*, the proportion of reference clusters reached by EQ states; and *EQ-state support*, the proportion of EQ states within the reference-cluster regions. EQ state support exceeds

98% for all configurations, while cluster coverage is 100%, except for one systematic-enumeration configuration. These results indicate that EQ exploration reaches nearly all independently identified reference regions across both training strategies.

Table 2: HYP25: state exploration across training-data generation strategies for RNNs with hidden dimension 50. MSE is in units of 10^{-5}, N_p reports number of partitions and $\tau = e = v = t = 0.01$

Config.		Systematic enumeration					Length-stratified random sampling				
$\|\Sigma\|$	$\|Q\|$	MSE	Time	N_P	Support %	Coverage %	MSE	Time	N_P	Support %	Coverage %
2	3	11.97	0.18	3.70	98.68	100.00	6.09	0.17	3.00	98.62	100.00
5	8	0.67	1.39	5.75	99.54	100.00	1.09	2.37	6.70	99.63	100.00
8	8	0.25	4.02	7.50	99.78	100.00	0.19	6.96	7.40	99.80	100.00
10	10	0.41	6.71	6.50	99.84	100.00	0.28	10.93	6.60	99.83	100.00
10	15	0.59	3.39	3.40	99.73	100.00	0.75	5.21	3.50	99.71	100.00
15	15	0.54	12.33	4.30	99.86	97.69	0.56	20.09	4.20	99.87	100.00

6 Discussion and Conclusion

In order to apply the active learning framework to extract WFA from RNNs, algorithms are needed to answer membership queries and equivalence queries. Membership queries are straightforward: simply evaluate the output from the RNN for the given input. It is less clear how to realise equivalence queries, and only a handful of papers have approached the topic. This paper rethinks equivalence queries, giving a new procedure utilising the structure of hidden-state space to efficiently establish equivalence. The procedure iteratively constructs a discrete abstraction of the RNN state space, using hyperplane partitioning. The extraction requires only the trained RNN and its input alphabet, without access to its original training data or target WFA.

The empirical results demonstrate that the new procedure can extract WFA from RNNs that closely match the outputs of the RNN. Increasing maximal number of visits to a partition (M) improves the fidelity, where the largest improvement is from $M = 5$ to 10, while the increase from $M = 10$ to $M = 25$ provided additional improvement. This suggests that the partitioning process converges towards a more stable representation of the RNN state space after a moderate amount of exploration. Compared with existing work by Okudono et al. (2020), the HYP25 approach achieves a comparable fidelity (Table 1), whilst requiring substantially less extraction time and fewer counterexamples. Therefore, the results indicate that the partitioning provides an efficient and scalable approach to equivalence queries while achieving good fidelity.

The hidden state coverage analysis further supports these findings. Across RNNs trained with both systematic enumeration and length-stratified random sampling, equivalence query exploration reaches nearly all independently identified reference clusters, while most visited states remained within the reference cluster regions. This indicates that the exploration is not limited to a small set of observed hidden states. Furthermore, variation of fidelity and extraction time across $(|\Sigma|, |Q|)$ configurations suggest that the extraction difficulty is not determined by the size of the target WFA, but may also depend on the structure of the representation learned by the RNN. The robustness of the results across the hidden dimensions of 16 and 50 further demonstrate the applicability of the method to RNNs with different state space dimensions.

The work presented in this paper has some limitations indicating the direction of future work. The experimental results are applied only to the synthetic (though very illustrative) problem of a validation loop: generate a WFA, learn an RNN from it, then extract a WFA from the RNN. As well working with different problem sets, working with larger RNNs with different architectures should be explored. The procedure has a number of parameters and an important direction for future work is adaptive or model-specific parameter selection.

In conclusion, this work represents a breakthrough in efficiently extracting WFA from RNNs, utilising hyperplanes to provide a lightweight iterative partitioning of the RNN hidden-state space that can be exploited in realising equivalence queries.

AI USE STATEMENT

This research was conducted without any use of AI tools, neither for the research nor for the writing of the article.

REPRODUCIBILITY STATEMENT

Upon acceptance of this work, the authors will make the code and results openly accessible. The complete extraction algorithm is presented in Appendix A and further details on the experiments are provided in Appendix E.

REFERENCES

Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

Stéphane Ayache, Rémi Eyraud, and Noé Goudian. Explaining Black Boxes on Sequential Data using Weighted Automata. *International Conference on Grammatical Inference*, 93:81–103, 2018. URL http://arxiv.org/abs/1810.05741.

Raphaël Bailly, François Denis, and Liva Ralaivola. Grammatical inference as a principal component analysis problem. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 33–40. Association for Computing Machinery, 2009.

Borja Balle and Mehryar Mohri. Learning Weighted Automata. In *International Conference on Algebraic Informatics*, pp. 1–21. Springer International Publishing, 2015.

Borja Balle, Xavier Carreras, Franco M. Luque, and Ariadna Quattoni. Spectral learning of weighted automata. *Machine Learning*, 96(1-2):33–63, 2014.

Adi Ben-Israel and Thomas N. E. Greville. Linear Systems and Characterization of Generalized Inverses. In *Generalized Inverses: Theory and Applications*, pp. 52–103. Springer, 2003.

Jean Berstel and Christophe Reutenauer. *Rational series and their languages*. Springer-Verlag, 1988.

J. W. Carlyle and A. Paz. Realizations by Stochastic Finite Automata. *Journal of Computer and System Sciences*, 5(1):26–40, 1971.

Adelmo Luis Cechin, Denise Regina Pechmann Simon, and Klaus Stertz. State automata extraction from recurrent neural nets using k-means and fuzzy clustering. In *23rd International Conference of the Chilean Computer Science Society, 2003. SCCC 2003. Proceedings.*, pp. 73–78, 2003.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. In *NIPS 2014 Workshop on Deep Learning*, 2014. URL http://arxiv.org/abs/1412.3555.

Manfred Droste, Werner Kuich, and Heiko Vogler (eds.). *Handbook of Weighted Automata*. Springer, 2009.

Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. DeepStellar: model-based quantitative analysis of stateful deep learning systems. In *Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 477–487. ACM, 2019.

Rémi Eyraud and Stéphane Ayache. Distillation of weighted automata from recurrent neural networks using a spectral approach. *Machine Learning*, 113(5):3233–3266, 2024.

Michel Fliess. Matrices de Hankel. *Journal de Mathématiques Pures et Appliquées*, 53, 1974.

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, 1992.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. JHU Press, February 2013.

Alireza Goudarzi, Peter Banda, Matthew R. Lakin, Christof Teuscher, and Darko Stefanovic. A Comparative Study of Reservoir Computing for Temporal Signal Processing, 2014. URL http://arxiv.org/abs/1401.2224.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Daniel Hsu, Sham M. Kakade, and Tong Zhang. A spectral algorithm for learning Hidden Markov Models. *Journal of Computer and System Sciences*, 78(5):1460–1480, 2012.

Thuan Q. Huynh and James A. Reggia. Symbolic Representation of Recurrent Neural Network Dynamics. *IEEE Transactions on Neural Networks and Learning Systems*, 23(10):1649–1658, 2012.

Henrik Jacobsson. Rule Extraction from Recurrent Neural Networks: A Taxonomy and Review. *Neural Computation*, 17(6):1223–1263, 2005.

S. C. Kleene. Representation of Events in Nerve Nets and Finite Automata. In *Automata Studies. (AM-34)*, pp. 3–42. Princeton University Press, 1956.

Trupti M. Kodinariya and Prashant R. Makwana. Review on determining number of Cluster in K-Means Clustering. *International Journal of Advance Research in Computer Science and Management Studies*, 1(6):90–95, 2013.

Gwénolé Lecorvé and Petr Motlicek. Conversion of recurrent neural network language models to weighted finite state transducers for automatic speech recognition. In *Interspeech 2012*, pp. 1668–1671. ISCA, 2012.

Zachary C. Lipton. The mythos of model interpretability. *Communications of the ACM*, 61(10): 36–43, 2018.

Reda Marzouk and Colin de la Higuera. Distance and equivalence between finite state machines and recurrent neural networks: Computational results. *arXiv preprint arXiv:2004.00478*, 2020. doi: 10.48550/arXiv.2004.00478.

Takamasa Okudono, Masaki Waga, Taro Sekiyama, and Ichiro Hasuo. Weighted Automata Extraction from Recurrent Neural Networks via Regression on State Spaces. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):5306–5314, 2020.

Christian W. Omlin and C. Lee Giles. Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM*, 43(6):937–972, 1996.

Guillaume Rabusseau, Tianyu Li, and Doina Precup. Connecting Weighted Automata and Recurrent Neural Networks through Spectral Learning. In *Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pp. 1630–1639, 2019. doi: doi.org/10.48550/arXiv.1807.01406.

Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 440–449. ACM, 1992.

Zeming Wei, Xiyue Zhang, and Meng Sun. Extracting Weighted Finite Automata from Recurrent Neural Networks for Natural Languages. In *International Conference on Formal Engineering Methods*, pp. 370–385. 2022.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples. In *35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5244–5253. PMLR, 2018.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Learning Deterministic Weighted Automata with Queries and Counterexamples. *Advances in Neural Information Processing Systems*, 32, 2019.

Sandamali Yashodhara Wickramasinghe, Jacob M Howe, and Laure Daviaud. Extracting Weighted Finite Automata from RNNs via iterative partitioning and spectral learning. In *CEUR Workshop Proceedings*, volume 7, 2025.

Sandamali Yashodhara Wickramasinghe, Jacob M. Howe, and Laure Daviaud. Extracting Deterministic Finite Automata from RNNs via Hyperplane Partitioning and Learning. In *Computational Intelligence*, pp. 116–135. Springer Nature Switzerland, 2026.

Frank Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

Zheng Zeng, Rodney M. Goodman, and Padhraic Smyth. Learning Finite State Machines With Self-Clustering Recurrent Networks. *Neural Computation*, 5(6):976–990, 1993.

Xiyue Zhang, Xiaoning Du, Xiaofei Xie, Lei Ma, Yang Liu, and Meng Sun. Decision-Guided Weighted Automata Extraction from Recurrent Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(13):11699–11707, 2021.

A Pseudo-code of the proposed algorithm

Algorithm 1 is to treat Step 1, and follows (Balle & Mohri, 2015). We present it here for completeness. Algorithm 2 outlines the new equivalence query procedure used in our approach.

Algorithm 1 From (Balle & Mohri, 2015): WFA Extraction via Membership and Equivalence Queries

Require: RNN $R = \langle h_0, g_R, f_R \rangle$ ($R(w)$ will denote the value output by R on string w)
Require: Alphabet Σ, initial basis $\mathcal{B} = (\mathcal{P}, \mathcal{S}) = (\{\varepsilon\}, \{\varepsilon\})$
Require: Error tolerance $e \geq 0$, rank tolerance $t \geq 0$, variance tolerance $v \geq 0$
Ensure: Extracted WFA A

while True **do**
  Perform membership queries
  Extend basis to ensure prefix-closure:

$$\mathcal{P} \leftarrow \mathcal{P} \cup \{u\sigma \mid u \in \mathcal{P}, \sigma \in \Sigma\}$$

  Extend basis to ensure suffix-closure:

$$\mathcal{S} \leftarrow \mathcal{S} \cup \{\sigma v \mid v \in \mathcal{S}, \sigma \in \Sigma\}$$

  Build Hankel matrix:

$$H_{\mathcal{B}}(u, v) = R(uv), \quad \forall u \in \mathcal{P}, v \in \mathcal{S}$$

  Completeness:
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{\, p\sigma \mid p \in \mathcal{P}, \sigma \in \Sigma, \mathrm{rank}(H_{(\mathcal{P} \cup \{p\sigma\}, \mathcal{S})}) > \mathrm{rank}(H_{(\mathcal{P},\mathcal{S})}) \,\}$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{\, \sigma s \mid \sigma \in \Sigma,\ s \in \mathcal{S},\ \mathrm{rank}(H_{(\mathcal{P}, \mathcal{S} \cup \{\sigma s\})}) > \mathrm{rank}(H_{(\mathcal{P},\mathcal{S})}) \,\}$
  Compute SVD of $H_{(\mathcal{P},\mathcal{S})}$ and determine rank k using tolerance t
  Extract WFA A from $H_{(\mathcal{P},\mathcal{S})}$ using spectral learning
  Perform equivalence query using Algorithm 2
  if a counterexample w is returned **then**
    Update basis:

$$\mathcal{P} \leftarrow \mathcal{P} \cup \mathrm{Pref}(w), \quad \mathcal{S} \leftarrow \mathcal{S} \cup \mathrm{Suff}(w)$$

  else
    return A
  end if
end while

Algorithm 2 Equivalence Query with State Space Refinement

Require: RNN $R = \langle h_0, g_R, f_R \rangle$ ($R(w)$ will denote the value output by R on string w and $h(w)$ the hidden state on w)
Require: Alphabet Σ, hypothesis WFA A
Require: Partition Γ (initialised $\Gamma = \{\mathbb{R}^d\}$) and for each $P \in \Gamma$, a finite set of vectors $\hat{P}$ initialised to be empty
Require: Error tolerance $e \geq 0$, variance tolerance $v \geq 0$, absolute variance threshold $\tau > 0$
Require: Max length $L_{\max}$, Max cluster visits M
Ensure: Counterexample w or **Equivalent**

Initialise queue $\mathcal{Q} \leftarrow \{\varepsilon\}$
Initialise partition visit counts: visits$(P) \leftarrow 0$ for all $P \in \Gamma$
while $\mathcal{Q} \neq \emptyset$ **do**
  Initialise pruning flags: prune_branch $\leftarrow$ False, prune_partition $\leftarrow$ False
  $w \leftarrow$ **pop**$(\mathcal{Q})$
  if $|A(w) - R(w)| > e$ **then**
    return w **counterexample**
  end if
  Compute hidden state: $h(w)$
  Identify set P in partition Γ such that $h(w) \in P$
  $\text{Var}_{\text{old}}(\hat{P}) \leftarrow \text{Var}(\hat{P})$
  Update $\hat{P} \leftarrow \hat{P} \cup \{h(w)\}$
  visits$(P) \leftarrow$ visits$(P) + 1$
  if $|w| \geq L_{\max}$ **then**
    prune_branch $\leftarrow$ True
  end if
  if visits$(P) \geq M$ **then**
    prune_cluster $\leftarrow$ True
  end if
  if $\text{Var}(\hat{P}) > (1 + v) \cdot \text{Var}_{\text{old}}(\hat{P})$ **and** $\text{Var}(\hat{P}) > \tau$ **then**
    Refinement: split P
    Find $h_* \leftarrow \arg\max_{h \in \hat{P}} \|h - h(w)\|$
    Compute hyperplane separating $h(w)$ and h_*:

$$z = h_* - h(w), \quad m = \frac{h_* + h(w)}{2}, \quad b = -z \cdot m$$

    Define P_1 and P_2 as half-spaces delimited by this hyperplane
    Update partition list:

$$\Gamma \leftarrow (\Gamma \setminus \{P\}) \cup \{P_1, P_2\}$$

    Initialize visits for new partitions: visits$(P_1) \leftarrow 0$, visits$(P_2) \leftarrow 0$
  end if
  if NOT prune_branch AND NOT prune_cluster **then**
    for each $\sigma \in \Sigma$ **do**
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{w\sigma\}$
    end for
  end if
end while
return **Equivalent**

B PROOFS OF RESULTS FROM SECTION 4.2

Proposition 5 (Proposition 1 in main part)**.** *Assume rank tolerance $t = 0$ and R computes a rational series. Then the sequence of hypothesis automata generated by the algorithm is finite.*

Proof.

Lemma 1. *Assume rank tolerance $t = 0$. Let $\mathcal{B}_1, \mathcal{B}_2, \ldots$ the (potentially infinite) sequence of closed and complete basis computed in Algorithm 1 which induces the sequence of hypothesis automata generated by the algorithm. Then, for all $i > 0$,*

$$\mathrm{rank}(H_{\mathcal{B}_i}) < \mathrm{rank}(H_{\mathcal{B}_{i+1}}).$$

Proof. For all i, since rank tolerance $t = 0$, $\mathcal{B}_{i+1}$ is constructed from $\mathcal{B}_i$ by adding all the prefixes and suffixes of a counterexample, hence adding at least one row in $H_{\mathcal{B}_{i+1}}$ that is not generated by the rows of $H_{\mathcal{B}_i}$, and so:

$$\mathrm{rank}(H_{\mathcal{B}_i}) < \mathrm{rank}(H_{\mathcal{B}_{i+1}}).$$

□

By definition, all $H_{\mathcal{B}_i}$ are submatrices of the Hankel matrix of the rational series computed by R (which has then finite rank), hence their rank is bounded. So by Lemma 1, the sequence of basis is finite and so is the sequence of hypothesis automata. □

Proposition 6 (Proposition 2 in main part)**.** *Suppose $L_{\max} = M = \infty$. Assume rank tolerance $t = 0$ and R computes a rational series. Then the final hypothesis automaton A generated by the algorithm satisfies that for all strings w in Σ^**

$$|A(w) - R(w)| \leq e$$

where $e \geq 0$ is the error tolerance.

Proof. Since $L_{\max} = M = \infty$, there is no forced termination of the algorithm. Hence, all strings will be explored in search of a counterexample. Since $t = 0$, by Lemma 1, the sequence of hypothesis automata is finite, and as in the proof of Lemma 1, with increasing rank. Consider A the final hypothesis automaton generated by the algorithm. If there was a string w such that $|A(w) - R(w)| > e$, the basis would be refined and another hypotheses automaton (with higher rank) would be generated. So there is no such w which completes the proof of the lemma. □

Corollary 2 (Corollary 1 in main part)**.** *Suppose $L_{\max} = M = \infty$. Assume the error tolerance $e = 0$, the rank tolerance $t = 0$, and R computes a rational series, then the last automaton in the sequence of hypothesis automata is the minimal WFA realising the function computed by R.*

Proof. Immediate from Proposition 2. Minimality comes from the fact that the rank of the Hankel matrix of the rational series computed by R equals the minimum number of states of any WFA realising it. Therefore, the final hypothesis automaton is a minimal WFA for it. □

Proposition 7 (Proposition 3 in main part)**.** *Assume rank tolerance $t = 0$ and R computes a rational series. If either the maximum exploration length $L_{\max}$ or the maximum number of visits per partition M is finite, then the algorithm terminates.*

Proof. By Proposition 1, only finitely many hypothesis automata are generated, and hence Algorithm 1 is called only a finite number of times (and always terminates by definition). Hence, Algorithm 2 is called finitely many times. If either $L_{\max} < \infty$ or $M < \infty$, only finitely many strings are explored and added to the queue in Algorithm 2 which hence always terminates too. □

Proposition 8 (Proposition 4 in main part)**.** *Assume the algorithm terminates and returns a WFA A. Then for every string $w \in \Sigma^*$, there exists a string $w' \in P_w$ such that*

$$|A(w') - R(w')| \leq e$$

where P_w is the set of the partition containing $h(w)$ at the end of the procedure and e is the error tolerance.

Proof. Let Γ be the partition obtained at the end of the algorithm. Let w be a string and $P \in \Gamma$ such that $h(w) \in P$. By definition of the partition refinement via hyperplanes, $\hat{P}$ is non empty so there exists a string w' such that $h(w') \in \hat{P}$. Moreover, any element added to $\hat{P}$ does not correspond to a counterexample, by definition of Algorithm 2. Hence, $|A(w') - R(w')| \leq e$. □

C Additional Experimental statistics

The Table 1 reports averages over ten runs for each $(|\Sigma|, |Q|)$ configuration on the hold-out test set. To compare the statistical significance of the difference between HYP25 and REG, two statistical tests were reported as follows.

First, a Wilcoxon signed-rank test was performed on the averaged NRMSE and time values, a non-parametric test for paired observations. This test was selected due to the small number of observations ($n = 10$), without requiring paired data to be normally distributed. No statistically significant difference was found in NRMSE between HYP25 and REG ($W = 11.0$, $p = 0.106$), whereas the difference in extraction time was statistically significant ($W = 0.0$, $p = 0.002$).

Secondly, for all 100 observations, statistical significance was assessed using a paired t-test, which test whether the mean difference between paired HYP25 and REG observations is significantly different from zero. No statistically significant difference was found in NRMSE ($t_{test} = 1.626, p = 0.107$), with a mean difference of 0.0095 (95% CI: $[-0.002, 0.021]$). In contrast, HYP25 has significantly lower extraction times than REG ($t_{test} = -6.221, p < 0.001$), reducing the extraction time by average 198.914 seconds (95% CI: $[135.469, 262.359]$ seconds).

D Hyper-parameter Tuning

D.1 Effect of Absolute variance threshold for extraction

Table 3: Effect of τ on HYP25 method. MSE values are reported in units of 10^{-5}. Results are averaged across runs. N_P denotes the number of partitions and N_{cex} the number of counterexamples.

Target WFA		MSE(A, R)				Time (s)				N_P				N_{cex}			
$\|\Sigma\|$	$\|Q\|$	.001	.005	.01	.1	.001	.005	.01	.1	.001	.005	.01	.1	.001	.005	.01	.1
2	3	7.893	7.987	7.987	9.938	0.535	0.170	0.194	0.090	12.500	5.300	3.600	1.400	2.500	2.400	2.400	2.200
4	6	0.519	0.519	0.589	0.641	1.726	0.683	0.662	0.460	18.100	4.900	3.100	1.400	4.200	4.200	4.100	4.000
5	8	0.672	0.639	0.757	0.800	4.134	1.480	1.190	0.885	27.100	6.900	3.800	1.400	4.800	4.800	4.700	4.600
8	8	0.299	0.392	0.445	0.550	75.293	6.250	3.963	2.787	98.800	14.300	7.000	1.200	6.800	6.400	6.300	6.000
10	10	0.313	0.344	0.344	0.375	48.229	6.956	5.921	5.161	51.100	8.700	4.200	1.200	7.200	7.100	7.100	7.000
10	12	0.290	0.425	0.562	0.714	48.828	7.508	5.196	3.598	60.700	9.000	3.200	1.100	7.000	6.700	6.200	5.800
10	15	0.396	0.722	0.860	0.907	25.156	3.577	2.515	2.169	42.900	5.800	2.600	1.300	5.500	4.900	4.600	4.400
15	15	0.661	0.680	0.737	0.737	94.342	10.573	6.669	5.801	53.100	6.700	2.700	1.300	6.300	6.100	5.800	5.800
15	20	0.739	0.692	0.862	0.943	53.087	4.652	1.784	1.263	37.800	5.000	2.300	1.100	3.300	4.000	2.900	2.700

The purpose of τ is to prevent small or negligible variance values from triggering partition refinements solely due to a relative increase in variance. It sets a minimum absolute variance threshold when doing refinements based on the RNN hidden state distribution. This experiment was conducted while keeping the other parameters fixed at $v = t = e = 0.01$ and the partition visit threshold at $M = 25$.

As τ increases from 0.001 to 0.1, the number of partitions decreases very strongly across the evaluated configurations. This behaviour is expected, as a larger minimum variance threshold makes refinement more restrictive, creating fewer partitions. This is also reflected in the extraction time, with a particularly large decrease generally observed when τ is increased from 0.001 to 0.005. Thus, a very small τ can lead to higher computational cost due to more extensive partition refinement and exploration.

More extensive exploration provides additional opportunities to identify counterexamples. Accordingly N_{cex} generally increases as τ decreases. However, this increase is considerably smaller than the corresponding increase in the number of partitions. The impact on $\text{MSE}(A, R)$ is also much smaller. This can be explained by the different roles of the variance parameters and the error tolerance in the equivalence query process. While variance-related parameters such as τ and v control the partition refinement and exploration, e determines whether the discrepancy between the RNN and hypothesised WFA is large enough to identify as a counterexample. Thus, changes in τ and v primarily affect the granularity of the partitioning and the computational effort required for the search, while discrepancies exceeding e can still be identified during the exploration.

Based on the results, $\tau = 0.01$ was selected as the configuration for the experiments. Although smaller values of τ can produce lower MSE for some configurations, they result in more partitions and longer extraction times. In contrast, $\tau = 0.1$ further reduces the computational cost but generally increases the MSE. The choice of $\tau = 0.01$ therefore provides a balance between relatively low MSE and computational cost, while avoiding the extensive partition refinement associated with smaller values of τ.

D.2 Effect of variance tolerance for extraction

The purpose of parameter v is to control the relative increase in variance required to trigger partition refinement. A partition $\hat{P}$ is refined when, $\text{Var}(\hat{P}) > (1+v) \cdot \text{Var}_{\text{old}}(\hat{P})$ and $\text{Var}(\hat{P}) > \tau$. Thus $v = 0.01, 0.1$ and 0.25 mean requiring a variance increase of more than 1%, 10% and 25% respectively compared to the previous variance $\text{Var}_{\text{old}}(\hat{P})$ before adding the hidden state. This experiment was conducted keeping the other parameters fixed at $t = e = \tau = 0.01$ and $M = 25$.

Based on the results in Table 4, as v increases, the number of partitions generally decreases, since a larger increase in relative variance is required before a refinement occurs. This is accompanied by generally lower extraction times, while changes in MSE and N_{cex} are comparatively smaller for most configurations. Having larger v means fewer partition refinements, and therefore less exploration, potentially limiting the captured behaviours. Since $v = 0.01$ has slightly lower MSEs with more partitions, it is selected for the remaining experiments.

One thing to note is that when v is increased (e.g., 0.25), the algorithm could find instances where $\text{Var}(\hat{P})$ has increased by more than 25%, but not trigger refinements due to resulting variance remaining below τ. It is necessary to avoid small variance values leading to an unnecessary number of refinements; hence, this may partly explain the limited variation in extraction results across different v.

Table 4: Effect of the variance tolerance v on HYP25. MSE values are reported in units of 10^{-5}. Results are averaged across runs. N_P denotes the number of partitions and N_{cex} the number of counterexamples.

Target WFA		MSE(A, R)			Time (s)			N_P			N_{cex}		
$\lvert\Sigma\rvert$	$\lvert Q\rvert$	.01	.1	.25	.01	.1	.25	.01	.1	.25	.01	.1	.25
2	3	7.99	7.99	13.84	0.19	0.13	0.10	3.60	2.10	1.70	2.40	2.40	2.30
4	6	0.59	0.67	0.65	0.66	0.48	0.45	3.10	2.40	1.40	4.10	4.00	4.00
5	8	0.76	1.21	0.80	1.19	0.71	0.84	3.80	1.40	1.00	4.70	4.60	4.60
8	8	0.45	0.49	0.51	3.96	2.74	2.63	7.00	1.40	1.00	6.30	6.20	6.13
8	10	0.54	0.55	0.55	2.88	2.86	2.53	4.30	2.00	1.00	5.60	5.80	5.73
10	10	0.34	0.34	0.34	5.92	5.12	4.93	4.20	2.20	1.30	7.10	7.10	7.10
10	12	0.56	0.71	0.71	5.20	3.48	3.26	3.20	1.60	1.00	6.20	5.90	5.80
10	15	0.86	0.90	0.98	2.52	1.81	1.84	2.60	1.30	1.00	4.60	4.40	4.30
15	15	0.74	0.77	0.74	6.67	4.88	6.19	2.70	1.10	1.10	5.80	5.60	5.80
15	20	0.86	0.81	0.94	1.78	1.87	1.40	2.30	1.60	1.10	2.90	3.10	2.70

D.3 Effect of rank tolerance for extraction

This experiment was conducted keeping the other parameters fixed at $v = e = \tau = 0.01$ and $M = 25$. The results in Table 5 show that the rank tolerance t has a substantial effect on the number of extracted states $|\hat{Q}|$, fidelity and time. At $t = 0$, the extracted WFA is larger, and the fidelity is lower, with very high MSE for several configurations. Increasing t to 0.1, produces a much smaller WFA, but increases the extraction time. In contrast, $t = 0.01$, consistently extracts WFA with lower MSEs, with moderate WFA size and low extraction times. Therefore, $t = 0.01$ was selected as the rank tolerance.

When $t = 0$, no tolerance is applied when determining the Hankel rank. Thus, small singular values arising from minor variations in the RNN output are captured. Since the RNN produces real-valued

Table 5: Effect of the rank tolerance t on HYP25. MSE values are reported in units of 10^{-5}. Results are averaged across runs. $|\hat{Q}|$ denotes the number of states in the extracted WFA and N_{cex} the number of counterexamples.

Target WFA		$\lvert\hat{Q}\rvert$			MSE(A, R)			Time (s)			N_{cex}		
$\lvert\Sigma\rvert$	$\lvert Q\rvert$	0	.01	.1	0	.01	.1	0	.01	.1	0	.01	.1
2	3	9.20	3.70	2.50	579658.45	7.99	9.66	0.15	0.19	8.45	3.00	2.40	9.60
4	6	10.80	6.00	3.80	0.76	0.59	2.71	0.36	0.66	26.76	4.40	4.10	20.40
5	8	12.70	7.00	4.20	716.90	0.76	2.34	0.61	1.19	57.19	5.40	4.70	22.90
8	8	24.20	8.60	6.10	392684.33	0.45	1.30	7.12	3.96	166.09	9.70	6.30	32.40
8	10	17.20	7.30	3.70	666422.91	0.54	2.19	2.55	2.88	192.89	7.40	5.60	34.80
10	10	20.20	8.30	3.80	4.56	0.34	2.43	4.36	5.92	200.89	8.90	7.10	33.00
10	12	18.90	7.30	3.20	131.70	0.56	2.28	3.54	5.20	169.68	8.40	6.20	29.90
10	15	18.60	5.70	2.10	6803252.23	0.86	2.19	3.58	2.52	51.97	8.10	4.60	14.30
15	15	22.78	7.20	2.40	4086.97	0.74	1.97	7.66	6.67	160.07	10.56	5.80	22.00
15	20	17.90	3.70	2.00	30776.31	0.86	1.12	6.21	1.78	19.43	7.70	2.90	7.60

outputs, small numerical differences can occur. Thus, retaining these values can lead to unnecessarily large WFAs with low MSEs. Also, when t is too large, singular values representing important behaviour can get truncated. Therefore, selecting an appropriate rank tolerance is important for balancing fidelity and model complexity.

E Reproducing the experiments

The algorithm is implemented in Python 3.9.6, and RNN models are trained using the PyTorch 2.6.0 library.

E.1 Dataset generation and model training

Target WFAs were generated randomly using Python (the method is explained under 5). These WFAs act as functions that read an input string and assign it a real-valued weight, i.e. f: $\Sigma^* \to \mathbb{R}$. The generation process allows for control over the number of states and the alphabet size. Training data were then sampled from the generated WFAs and used to train the RNNs.

Data were obtained using two different approaches. First, strings were generated through systematic enumeration, with the dataset limited to 500,000 strings and a maximum string length of 20. Moreover, to investigate robustness, we ran additional experiments using data generated with a length-stratified sampling approach, with a similar data limit and maximum length. In length-stratified sampling, data were randomly generated such that each length is fairly represented up to the maximum length. Each dataset was split into training, validation, and test sets using proportions of 90%, 5%, and 5%, respectively. The random seed used for dataset generation and splitting was fixed for each experimental run.

RNNs were implemented as LSTM regression models and trained using the Adam optimiser with a learning rate of 0.001 and mean squared error (MSE) loss. RNNs with two LSTM layers and 16/50 hidden units were trained. Training used a batch size of 64 for a maximum of 50 epochs, with early stopping after 5 epochs without improvement in validation loss. The model with the lowest validation loss was retained for evaluation. The RNN models were trained using an NVIDIA A100 GPU, 6 CPU cores, and 16 GB of memory on an HPC cluster.

For each Σ, $|Q|$ configuration, 10 target WFAs were randomly generated, and an RNN was trained for each target WFA. The same target WFAs and corresponding trained RNNs were reused when comparing different extraction settings and methods.

E.2 Extraction algorithm

The extraction algorithm requires the trained RNN and the corresponding input alphabet. Extraction is performed directly from the RNN's input–output behaviour and hidden-state representations.

The target WFA is not required for extraction. The extraction is controlled by several parameters, including the error tolerance e, partition visit threshold M, max string length L_{max}, absolute variance threshold τ, variance tolerance v, and the rank tolerance used during spectral learning. The extraction algorithm is completely described in Appendix A.