Synthesizing State Space Models: Learning Meets Logic

William Fishell*
*Columbia University, USA
Email: wf2322@columbia.edu

***Abstract*—State space models (SSM) have gained popularity recently due to their ability to perform self-attention in $O(n)$, whereas Transformers require $O(n^2)$. SSMs are trained by backpropagation and gradient descent, yielding a continuous state space in which the outputs at timestep t are defined by its inputs and the current state at time t. Clustering the states generated creates a finite state machine that represents the learned SSM, much like finite state machines generated by reactive synthesis. Despite similarities in the learned controllers between these two approaches the processes of SSM learning and reactive synthesis are fundamentally different. SSM learning is an optimization problem and reactive synthesis is solved as a symbolic two-player game. In order to understand the connections between these two approaches, we compare SSM learning to reactive synthesis over the same set of problems, with the goal of generating the same controller. To do this, we generate sets of example traces from temporal logic specifications for the underlying reactive system and try to recreate this system through specification mining and gradient based methods. We train the SSM end-to-end by sampling non-vacuously true traces, so that the SSM's states can be clustered to learn a controller for the reactive system. We also mine a Linear Temporal Logic (LTL) specification from the training subset of example traces of the environment assumptions and system guarantees and use these LTL specifications to construct and synthesize a controller for the underlying reactive system. We illustrate the tradeoffs of gradient-based learning and synthesis methods to uncover correct controllers for the same underlying task by comparing the sample complexity of SSM learning and specification mining.**

I. Introduction

Deep neural networks can approximate highly complex functions, yet popular sequence models (RNNs, Transformers) do not explicitly learn state spaces [1], [2]. Unlike Transformers and RNNs, SSMs parameterize continuous state spaces and learn a transition system that is similar to a Mealy machine. The resulting set of continuous states can be clustered into a finite set that is a Mealy machine. This enables direct comparison with LTL reactive synthesis [3]–[5]. SSMs provide a natural bridge between gradient-trained models and explicit automata produced by LTL reactive synthesis, because they can simulate rich sequence transductions and expose an interpretable state representation that can be made finite. For SSM learning, we generate all training traces from the full reactive specification φ_0. In contrast, for specification mining [6] we generate two disjoint trace sets—assumption traces from φ_{0A} and guarantee traces from φ_{0G}—since Scarlet cannot distinguish inputs vs. outputs otherwise. We then train the SSM on the φ_0 traces, collect its latent states by re-inputting those same traces, and cluster them to form a finite-state abstraction. Separately, we mine the assumption and guarantee traces to recover formulas φ_A^* and φ_G^*, compose them as $(\varphi_A^* \rightarrow \varphi_G^*)$, and synthesize a controller. Looking at sample complexity of specification mining plus synthesis versus gradient-based learning reveals what classes of problems each of these learning methods are adept at. Better understanding of what problems can be learned using specification mining and formal logic compared to gradient based learning helps demonstrate how reactive synthesis should be used alongside deep learning.

II. Problem Statement

SSMs are models defined by two equations which control the state and outputs of the system as a function of the input $u(t)$ and current state $x(t)$ [7].

$$x'(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + Du(t)$$

These dynamics resemble a Mealy machine, but with state transitions and outputs learned through gradient descent.

In contrast, LTL reactive synthesis produces correct-by-construction controllers, yet this requires a complete formal specification φ. We assume no access to the ground-truth specification denoted φ_0 in learning a controller. Instead, we rely on observed assumption and guarantee traces. We use Scarlet [8], an LTL specification mining tool that applies logical inference to separate positive and negative examples to derive candidate specifications φ_A^* and φ_G^* representing the learned assumptions and guarantees.

A. Finite-Trace Semantics (LTL_f)

Because our data consist of finite execution traces rather than infinite streams, all LTL formulas are interpreted under finite-trace semantics (LTL_f). The SSM-generated and synthesized controllers are evaluated against those finite-trace definitions [9]. Concretely: A finite trace satisfies an LTL_f formula if all obligations are met in the trace, and the infinite continuation of its last state would not violate the corresponding LTL formula.

- A finite trace $\pi = s_1\, s_2 \cdots s_n$ satisfies an LTL_f formula φ if and only if φ holds at position 1 under the usual inductive rules for X, U, G, F, plus three stipulations:
  1) Any X at position n is false (we require a real "next" state)
  2) All F and U must be satisfied in the finite trace

3) A globally formula G must hold at *all* positions $1 \leq i \leq n$, in particular at the final state n.

A finite trace is consistent with the underlying LTL system if and only if it is one in which the corresponding LTL_f system is not violated. Finite traces that do not violate are termed *positive traces*, whereas finite traces that violate the system are termed *negative traces*.

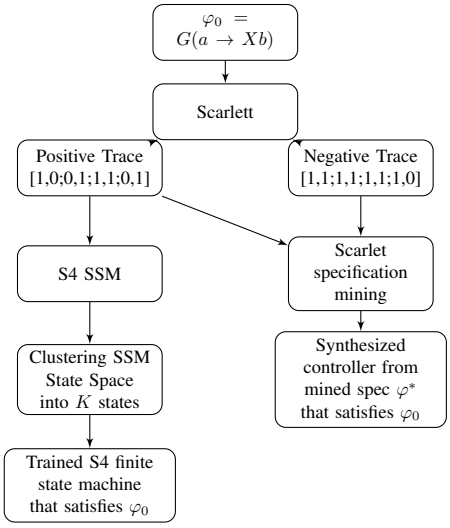
Problem 1. *Let $\{(AP_I^1, AP_O^1), \ldots, (AP_I^n, AP_O^n)\}$ be a finite sequence of input–output pairs. Given this sequence and an additional input AP_I^{n+1}, the task is to generate the corresponding output AP_O^{n+1} such that the extended sequence*

$$(AP_I^1,\ AP_O^1),\ (AP_I^2,\ AP_O^2), \ldots,$$
$$(AP_I^n,\ AP_O^n),\ (AP_I^{n+1},\ AP_O^{n+1})$$

is consistent with the behavior of the underlying LTL system.

We evaluate SSMs and LTL reactive synthesis by looking at how many finite traces are needed perfectly create a controller that only generates positive examples. This measurement is treated as analogous to sample complexity in learning theory [10], and demonstrates how well synthesis learns controllers for these reactive systems compared to SSMs

III. Methodology



WLOG, φ_0 denotes the full reactive system due to space constraints; formally this methodology also comprises separate assumption and guarantee formulas, φ_{0A} and φ_{0G}.

We choose S4 [11], [12] for the SSM architecture because its parallel training efficiently captures long-range dependencies of the underlying reactive system. At inference, these parameters admit a recurrent rollout, treating the model as a Mealy-style input $\rightarrow$ state $\rightarrow$ output system. Scarlet is used to mine the corresponding LTL system and LTLSYNT [13] is used to generate a controller from this specification. SSM training traces are all drawn from one LTL spec φ_0 with vacuously true specs removed, whereas spec-mining uses two separate generators: one on φ_{0A} (assumptions) and one on φ_{0G} (guarantees).

The SSM and reactive synthesis system are validated by feeding finite traces to their respective controllers and evaluating whether they solve ***problem 1***. To determine whether a finite trace τ satisfies the LTL specification φ_0 for ***problem 1***, we first translate φ_0 into an equivalent finite-trace LTL_f formula φ_{0f}, which is interpreted over finite executions. We then use Spot's [14] LTL_f support to construct a corresponding HOA from φ_{0f}, yielding an automaton that accepts exactly those infinite traces whose finite prefixes satisfy φ_{0f}. Because Spot only handles infinite traces, we convert our finite trace

$$\tau = s_0, s_1, \ldots, s_n$$

into an infinite trace

$$\tau_\infty = s_0, s_1, \ldots, s_n, s_n, s_n, \ldots$$

by "cycling" the last state s_n indefinitely. In φ_{0f} we replace each standard next operator X with the strong-next operator $X[!]$ to enforce next-state requirements indefinitely. Finally, we feed τ_∞ into the constructed HOA: if the automaton accepts, then the original finite trace τ satisfies φ_0; if it rejects, then τ violates φ_0.

IV. Results

Recovery thresholds and times for various sequence lengths for φ_0 using specification mining

Sequence Length	Recovery Threshold (samples)	Run Time
100	40	0.171s
1,000	40	4.184s
10,000	NA	Timed Out

We focus here on evaluating specification mining; empirical results for the SSM-to-Mealy pipeline are left to future work. Rather than measure SSM sample complexity directly, we draw on theoretical bounds for Transformers—whose expressive power matches that of SSMs [15]—to contextualize our findings. Notably, Transformers can learn long-range dependencies over sequences of length t with only $O(\log t)$ samples [16], suggesting that expressive models may generalize from few examples.

To test whether specification mining achieves similar efficiency, we evaluate it (and later, SSM controllers) on the "gfand03" benchmark from SYNTCOMP [17]:

$$\varphi_0 \ = \ \big(G(a) \wedge F(b)\big) \ \longrightarrow \ F(c),$$

where a and b are inputs and c is an output.

Using SCARLET, specification mining recovers this formula with just a few constant number of traces—indicating that, for simple systems, uncovering long-range dependencies does not require more samples as sequence length grows. Scaling to richer specifications, however, remains an open challenge.

V. Conclusion

Future work of ours will look more deeply into this to gain a better understanding about what problems are best suited for each of these learning dynamics as well as utilizing different tools such as AALPY [18] an automata learning library for better scalability. Further

ACKNOWLEDGEMENTS

I would like to thank Professor Mark Santolucito, my advisor, for their feedback and help in pursuing this research

REFERENCES

[1] N. Elhage, N. Nanda, C. Olsson, T. Henighan, N. Joseph, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly *et al.*, "A mathematical framework for transformer circuits," *Transformer Circuits Thread*, vol. 1, no. 1, p. 12, 2021.

[2] 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*, vol. 4, no. 3, pp. 393–405, 1992.

[3] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1989, pp. 179–190.

[4] A. Pnueli, "The temporal logic of programs," in *18th annual symposium on foundations of computer science (sfcs 1977)*. ieee, 1977, pp. 46–57.

[5] W. Merrill and N. Tsilivis, "Extracting finite automata from rnns using state merging," *arXiv preprint arXiv:2201.12451*, 2022.

[6] S. Germiniani, D. Nicoletti, and G. Pravadelli, "A systematic literature review on mining ltl specifications," *IEEE Access*, 2025.

[7] J. Lin and G. Michailidis, "Deep learning-based approaches for state space models: A selective review," *arXiv preprint arXiv:2412.11211*, 2024.

[8] R. Raha, R. Roy, N. Fijalkow, and D. Neider, "Scalable anytime algorithms for learning fragments of linear temporal logic," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2022, pp. 263–280.

[9] G. De Giacomo, M. Y. Vardi *et al.*, "Synthesis for ltl and ldl on finite traces," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*. AAAI Press, 2015, pp. 1558–1564.

[10] L. G. Valiant, "A theory of the learnable," *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.

[11] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[12] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," *arXiv preprint arXiv:2111.00396*, 2021.

[13] F. Renkin, A. Duret-Lutz, A. Pommellet, and P. Schlehuber, "ltlsynt (spot 2.9+)."

[14] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu, "Spot 2.0—a framework for ltl and-automata manipulation," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2016, pp. 122–129.

[15] W. Merrill, J. Petty, and A. Sabharwal, "The illusion of state in state-space models," *arXiv preprint arXiv:2404.08819*, 2024.

[16] B. L. Edelman, S. Goel, S. Kakade, and C. Zhang, "Inductive biases and variable creation in self-attention mechanisms," in *International Conference on Machine Learning*. PMLR, 2022, pp. 5793–5831.

[17] S. Jacobs, G. A. Pérez, R. Abraham, V. Bruyere, M. Cadilhac, M. Colange, C. Delfosse, T. van Dijk, A. Duret-Lutz, P. Faymonville *et al.*, "The reactive synthesis competition (syntcomp): 2018–2021," *International journal on software tools for technology transfer*, vol. 26, no. 5, pp. 551–567, 2024.

[18] E. Muškardin, B. K. Aichernig, I. Pill, A. Pferscher, and M. Tappler, "Aalpy: an active automata learning library," *Innovations in Systems and Software Engineering*, vol. 18, no. 3, pp. 417–426, 2022.