# World Model Architectures for Model-Based Reinforcement Learning

Triston Grayston
*University of Victoria*
tristongrayston@uvic.ca

Ari Van Everdingen
*University of Victoria*
arivaneverdingen@uvic.ca

***Abstract*—World models offer several theoretical benefits, such as enhanced planning capabilities, and faster, safer, and cheaper sampling. However, training an effective world model is difficult. This work explores this challenge by testing 3 neural network architectures - neural networks with a residual connection, recurrent neural networks, and Neural Circuit Policies - in approximating the dynamics of 3 environments: the Lorenz system, Open AI gym's Pendulum, and a modified, partially observed Pendulum.**

## I. Introduction

### A. Motivation

Reinforcement learning typically divides into Model-Free and Model-Based Reinforcement Learning. In Model-Free, agents optimize their behavior to achieve a specific goal, implicitly learning the environment's dynamics on the way. Model-Based Reinforcement Learning explicitly learns a dynamics model separate to the agent. Incorporating an accurate dynamics model provides substantial advantages, including enhanced planning capabilities, and faster, safer, and cheaper sampling compared to interactions with real-world or high-fidelity simulated environments. Real-world interactions present significant challenges due to slow sampling speeds, high costs, and safety risks associated with failures. Learned neural network-based world models offer a promising alternative to traditional simulations, as precisely capturing and replicating real-world dynamics through explicit modeling can be infeasible or prohibitively complex.

Despite these advantages, model-based approaches remain relatively underexplored, primarily due to difficulties in generating accurate and representative simulated data. Effective world models must simultaneously achieve correctness, sample efficiency, and computational simplicity—criteria that few existing architectures satisfy simultaneously. Those that do meet these conditions often struggle with limited expressiveness or excessive complexity.

### B. Related Works

Model-based reinforcement learning approaches seek to explicitly capture environmental dynamics through a learned world model, which then facilitates planning and decision-making. Early successes in this field utilized ensembles of neural networks to enhance generalization and reduce model uncertainty, demonstrating improved sample efficiency compared to model-free counterparts. However, these ensemble methods introduce additional complexity and computational overhead, motivating research into more efficient yet expressive representations of dynamics [1]. Notably, recent model-based algorithms such as DreamerV3 have demonstrated state-of-the-art performance in benchmark tasks, yet reproducibility and transparency remain significant barriers, limiting their broader adoption in practical scenarios [2].

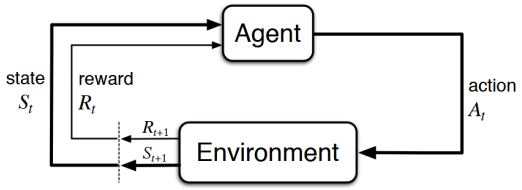


Fig. 1. The agent–environment interaction in a Markov decision process. [6]

Advances in continuous-time modeling of system dynamics have inspired the development of Neural Ordinary Differential Equations (Neural ODEs) [3]. Neural ODEs parameterize the derivative of the state as a neural network, integrating forward through time to capture complex, continuous-time dynamical systems in a differentiable manner. This continuous-time framework has motivated biologically inspired architectures such as Liquid Time-Constant (LTC) Networks [4] and Neural Circuit Policies (NCPs) [5]. These models leverage structured neural circuits to enhance expressive capability, robustness, and interpretability. NCPs, in particular, have demonstrated promising results in challenging real-world reinforcement learning tasks by effectively capturing intricate temporal relationships.

Collectively, these advances underscore an ongoing trend toward developing models that balance expressiveness, sample efficiency, interpretability, and computational simplicity, a crucial intersection seldom achieved by existing reinforcement learning architectures.

### C. Problem Definition

We consider an agent interacting with an environment, which can be a physical setting for a biological entity or a simulated domain for a computational system. This agent–environment interaction is often modeled as a Markov decision process (MDP), as illustrated in Figure 1.

Through this interaction, the agent observes a sequence of states and actions, denoted by $(s_0, a_0, s_1, a_1, ...)$, or equivalently by two tuples $[(s_0, s_1, ...), (a_0, a_1, ...)]$. For the purpose of capturing state transitions, we omit the reward signal from this formulation. We thus define the transition dynamics of the environment via a probability function $p$:

$$p(s_{t+1}|s_t, a_t) = Pr\{S_{t+1} = s'|S_t = s_t, A_t = a_t\}$$

for $\forall s', s \in \mathcal{S}, a \in \mathcal{A}$.

Given the rapid progress in deep learning, it is increasingly feasible to approximate $p$ with neural networks, making data-driven models of the environment accurate and robust. However, despite these promising developments, model-based reinforcement learning—and particularly the use of neural networks as learned "world models"—remains relatively underexplored compared to model-free approaches. This work focuses on bridging that gap, aiming to advance our understanding of neural network–based world models within RL frameworks.

We investigate the challenge of making useful world models by experimenting with three distinct neural network architectures. Two of which have a well-established track record and extensive literature detailing their advantages for sequence modeling and temporal dynamics, among other tasks. The third architecture, Neural Circuit Policies, shows particular promise in domains that involve physical interactions and causal reasoning. By comparing these three approaches, we aim to highlight their strengths and limitations, ultimately guiding the development of more robust and interpretable world models for model-based reinforcement learning. We aim to test these architectures on environments displaying separate properties. We capture their effectiveness in each environment as determined by their performance in next-step prediction and n-step prediction. Additionally, we compare their abilities of generalization to samples found beyond the training set, as well their ability to capture dynamics with varying sample sizes.

## II. Methodology

For the purposes of our paper, we wished to be convinced of the performance of specific neural network architectures as world models. We tested three architectures and three environments to measure the performance of the architectures. In addition, we varied the amounts of data the architectures were trained on to provide insight on their capability of sample efficiency.

### A. Architectures

The three neural network-based architectures we used for this work are Neural Networks with a Residual Connection or a *Residual Block*, *Recurrent Neural Networks*, and *Neural Circuit Policies*.

**Residual blocks**, first popularized by ResNet architectures, enable neural networks to grow exceptionally deep while mitigating gradient-related training difficulties. The core idea is to learn a residual function—that is, the change from an input to the desired output—rather than the direct mapping itself. Formally, for an input $x_t$, the updated state is given by.

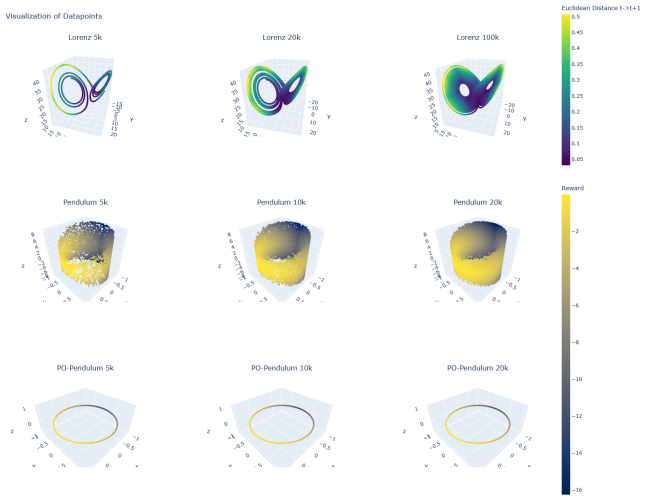


Fig. 2. Visualization of Data from random trajectories on each environment with three different amounts of sampling. The top row of plots describe samples from the Lorenz system, the second row is from Pendulum, and the third row is from Partially Observed Pendulum.

$$x_{t+1} = f_\theta(x_t) + x_t$$

where $f_\theta$ represents a neural network. Residual connections bear a conceptual resemblance to Euler's method in numerical analysis, in which an incremental update is applied to the current state to approximate the next value.

**Recurrent Neural Networks** (RNNs) model sequential data and time-series phenomena. At each time step t, an RNN cell receives the current input $x_t$ and a hidden state $h_{t-1}$ from the prior step, generating a new hidden state $h_t$ and an output $\hat{y}_t$. This hidden state contains information about previous states, serving as a sort of memory, making RNNs well-suited for tasks with sequential relationships.

**Neural Circuit Policies** (NCPs) are a distinct architecture that blends concepts from the preceding methods, drawing particular inspiration from the compact nervous system of the *nematode C. elegans*, which features only a small number of neurons yet displays highly adaptive behaviors. This design integrates biologically motivated principles—sparse, layered neural circuits and continuous-time neuronal dynamics—with contemporary training techniques. Unlike standard RNNs, NCPs rely on two key innovations: (1) neurons governed by ordinary differential equations (ODEs), and (2) a sparse, structured connectivity pattern reminiscent of biological networks. Concretely, NCPs employ a four-tier hierarchical layout: *sensory neurons* receive external inputs, *interneurons* and *command neurons* jointly process information and make decisions, and *motor neurons* produce the final outputs. The specific wiring among these neuron layers is randomized according

to a *Bernoulli distribution*, controlled by hyperparameters that dictate the feed-forward connection probabilities.

During training, NCPs utilize backpropagation through time, but the unrolled computational graph must also include the internal ODE solver. The hidden state of the entire NCP $\mathbf{x}(t) \in \mathbb{R}^{D\times 1}$ evolves according to

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = -\big[\mathbf{w}_\tau + f_\theta\big(\mathbf{x}(t), \mathbf{I}(t)\big)\big] \odot \mathbf{x}(t) \; + \; A \odot f_\theta\big(\mathbf{x}(t), \mathbf{I}(t)\big)$$

where $\mathbf{I}(t) \in \mathbb{R}^{m\times 1}$ is an exogenous input, $\mathbf{w}_\tau$ and $A$ are parameter vectors, $f$ is a neural network parameterized by $\theta$, and $\odot$ denotes elementwise multiplication. Because gradients must propagate through each integration step, a neuron's current state influences the loss for all previous time points.

### B. Environments

Each of the environments exhibits distinct characteristics that allow for rigorous evaluation of our architectures, testing their abilities across varying complexities and dynamics.

Our first environment is the **Lorenz System**, defined by the following ordinary differential equations:

$$\begin{aligned}\frac{dx}{dt} &= \sigma(y - x),\\ \frac{dy}{dt} &= x(\rho - z) - y,\\ \frac{dz}{dt} &= xy - \beta z.\end{aligned}$$

parameters $\sigma = 10, \rho = 28, \beta = \frac{8}{3}$. The Lorenz System is a canonical example of chaotic systems, characterized by deterministic yet highly sensitive dynamics dependent on initial conditions. Its chaotic nature implies that minute variations in initial states lead to exponentially diverging trajectories, making it particularly challenging for predictive modeling. Moreover, the absence of actions in this system isolates the difficulty of accurately capturing and generalizing complex temporal dynamics, enabling a focused assessment of our models' intrinsic predictive capabilities without the added complexity of inferring the impacts of external control inputs. The second environment is the **Pendulum** from *OpenAI's Gym*, which represents a simplified yet richly informative physical system possessing Markovian dynamics. The environment's state encapsulates the Cartesian coordinates of the pendulum's tip along with its angular velocity, while the action applies torque directly to the end of the pendulum. The objective is to stabilize and balance the pendulum upright. This environment introduces the challenge of explicitly modeling the interplay between state transitions and control actions, thus evaluating our models' proficiency in handling action-dependent dynamics and control-oriented predictive tasks. The third environment is the **Partially Observed Pendulum**, a variant of the previous Pendulum environment. This environment removes the angular velocity component from the observable state, resulting in a non-Markovian setting. The lack of full observability demands that models infer latent dynamics through temporal integration.

### C. Experiments

We evaluated the models prediction generalization capabilities on given environments using their autoregressive performance, defined in algorithm 1, as the cumulative sum of errors sustained by the models will illuminate their performance. Each of the models were trained on some training set to convergence on one-step predictive accuracy. For Lorenz Systems, we place importance on the model's ability to approximate the underlying ODEs which define the system. We track the trajectories achieved by the models when initial conditions are changed dramatically, achieved by negating the initial conditions on the Z-axis. We look for deviations in one step and autoregressive prediction, evaluating specifically on distance to the original path and similar patterns of progression. For Pendulum and Partially Observed Pendulum, we initialized each model at a variety of points found both within and outside of the training set. By analyzing how well the models follow the true pendulum motion in an autoregressive manner, we captured their capacity to maintain low prediction error over time while staying within the regions they were trained on. We distinguish a maximum tolerance for residuals at 0.24, which is the average residual given by the identity (which predicts the same state it is given). We use this threshold to suggest when predicted trajectory has strayed significantly from the sampled trajectory. By performing these autoregressive rollouts numerous times, we get a statistical measure of the performance of each of those models. We trained each model on datasets of varying sample sizes to fully capture how the amount of available data influences learning and generalization. Smaller datasets reveal a model's capacity to handle data-scarce conditions, while larger datasets test the model's ability to leverage more abundant information. The different datasets we created has been illustrated in Figure 2

**Algorithm 1** Autoregressive Rollout

**Inputs**
WM ▷ world model
States ▷ reference trajectory states
Actions ▷ reference trajectory actions
T ▷ rollout length
wl ▷ window_length

**Array** Error
**Array** Window ← (States[:wl], Actions[:wl])
**while** $i < T$ **do**
  prediction ← $WM$(Window)
  Error[i] ← States[i+1] - prediction
  Window ← (Window[1:] + prediction, Actions[i:i+wl])
**end while**

## III. Results

### A. Lorenz Systems

We primarily care about the models effectiveness at capturing the underlying dynamics of the ODE at play: can

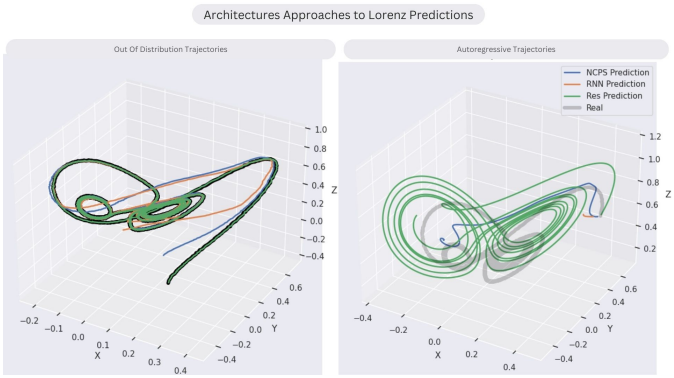


Fig. 3. Graphs detailing architectures trajectory on Lorenz System: OOD and Autoregressive Rollouts

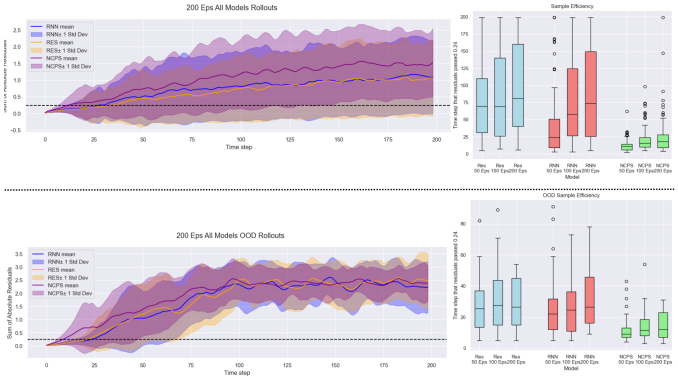


Fig. 4. These box plots describe at which step individual predicted trajectories cross the 0.24 threshold. These results are from 150 different random trajectories.

the models generalize to out of distribution data, and can they maintain lobe-switching characteristics when applied auto regressively. We see on one-step predictions on *in distribution data*, all models performed effectively equivalently as shown in Figure 3. We see a switch however, with only the Residual Block being capable of generalizing to the out of distribution data. Further, we see that the residual block maintains lobe-switching characteristics while the other two models fail to do so, and falling into stable points instead. While the RNN's relatively poor performance may be attributed to limitations inherent in its structure, there is intrigue with the distinct performance gaps between NCPS and the Residual Block. We hypothesize they are training on fundamentally seperate tasks, with the resnet approximating a vector field with some margin allowed in the precision of it's approximation. Conversely, the NCP architecture appears to attempt a more rigid, closed-form representation of the underlying dynamical equations, leaving it more susceptible to cumulative errors and sensitive to deviations during autoregressive inference.

### B. Pendulum

On in-distribution predictions, the residual connection and RNN models are comparable when trained on 200 samples. However, the residual connection model performs well with only 50 samples, while the RNN needs all 200 for the boxplot to display a similar median. This observation implies that the residual connection model is more sample efficient. The NCPS models have worse performance on all training set sizes. Plots depicting the performance of the 3 architectures trained on varying numbers of samples is in Figure 4.

On out of distribution tests, performance is worse across the board. This is expected, as the models have not been trained on these trajectories. However, this also suggests that the models are not learning the underlying causal dynamics of the system, only approximations at points they have been trained on. That said, of the three, the residual connection and RNN models perform the best. Performances are depicted in Figure 4.

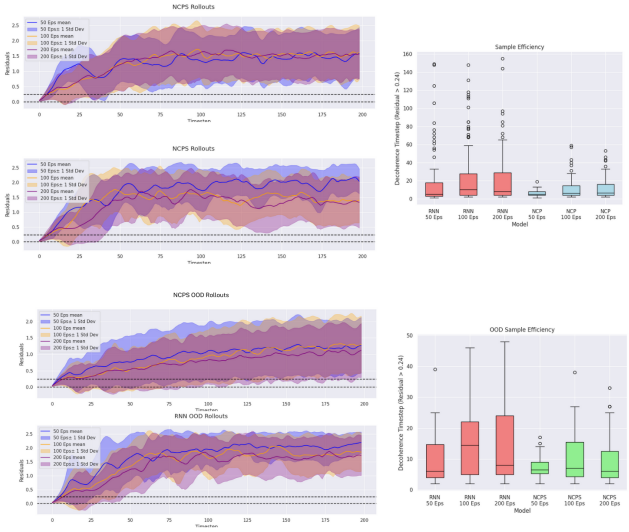


Fig. 5. Graphs detailing architectures performance on training set, Out-of-Distribution Lorenz System, and Autoregressive Rollouts

### C. Partially Observed Pendulum

Partially Observed Pendulum excludes the velocity from the observation, which makes the environment non-Markovian. Therefore in order for models to learn their dynamics, they must have a recurrent component. This feature excludes the recurrent connection model from these tests.

As expected from the restricted information, the performances decreases relative to the regular Pendulum environment. Similar to the Pendulum environment, the NCPs are outperformed by the RNNs, albeit by a smaller margin.

## IV. Conclusion

This work explores the gap between the theoretical benefits of world models, and their immature status in literature. We test 3 neural network architectures - neural networks with a residual connection, recurrent neural networks, and

Neural Circuit Policies - in approximating the dynamics of 3 environments: the Lorenz system, Open AI gym's Pendulum, and a modified, partially observed Pendulum. The residual connection model performed best on Pendulum and the Lorenz system, in terms of both sample efficiency, and residuals. This means regardless of the number of samples the models were trained on, the residual connections' predicted trajectories remained closest to the ground truth, for longest. The performance of the RNN is close behind for larger sample sizes. In all cases, NCPs place third. In Partially Observed Pendulum, for which the residual connection is omitted, performance across the board is worse. This is expected, as the models have less information to work with.

Future work entails integrating these architectures into model based learning with PPO agents and compare their agents respective decision-making performance. A larger issue is forming a hypothesis for why NCPs are under-performing in our benchmarks, and identifying a set of environments better suited for NCPS.

## REFERENCES


[1] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," 2019.

[2] D. Hafner, J. Pasukonis, J. Ba, and T. P. Lillicrap, "Mastering diverse domains through world models," 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:255569874

[3] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," 2018.

[4] R. Hasani, M. Lechner, A. Amini, L. Liebenwein, A. Ray, M. Tschaikowski, G. Teschl, and D. Rus, "Liquid time-constant networks," 2021.

[5] M. Lechner and R. Hasani, "Neural circuit policies enabling auditable autonomy," *Nature Machine Intelligence*, vol. 2, pp. 642–652, 2020.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.