Structured State Space Models for In-Context Reinforcement Learning

Chris Lu [*1] **Yannick Schroecker** [2] **Albert Gu** [2] **Emilio Parisotto** [2]
Jakob Foerster [1] **Satinder Singh** [2] **Feryal Behbahani** [2]

Abstract

Structured state space sequence (S4) models have recently achieved state-of-the-art performance on long-range sequence modeling tasks. These models also have fast inference speeds and parallelisable training, making them potentially useful in many reinforcement learning settings. We propose a modification to a variant of S4 that enables us to initialise and reset the hidden state in parallel, allowing us to tackle reinforcement learning tasks. We show that our modified architecture runs asymptotically faster than Transformers in sequence length and performs better than RNN's on a simple memory-based task. We evaluate our modified architecture on a set of partially-observable environments and find that, in practice, our model outperforms RNN's while also running over five times faster. Then, by leveraging the model's ability to handle long-range sequences, we achieve strong performance on a challenging meta-learning task in which the agent is given a randomly-sampled continuous control environment, combined with a randomly-sampled linear projection of the environment's observations and actions. Furthermore, we show the resulting model can adapt to out-of-distribution held-out tasks. Overall, the results presented in this paper show that structured state space models are fast and performant for in-context reinforcement learning tasks.

1. Introduction

Structured state space sequence (S4) models (Gu et al., 2021a) and their variants such as S5 (Smith et al., 2022) have recently achieved impressive results in long-range sequence modelling tasks, far outperforming other popular sequence models such as the Transformer (Vaswani et al., 2017) and LSTM (Hochreiter & Schmidhuber, 1997) on the Long-Range Arena benchmark (Tay et al., 2020). Notably, S4 was the first architecture to achieve a non-trivial result on the difficult Path-X task, which requires the ability to handle extremely long-range dependencies of lengths $16k$.

Furthermore, S4 models display a number of desirable properties that are not directly tested by raw performance benchmarks. Unlike transformers, for which the per step runtime usually scales quadratically with the sequence length, S4 models have highly-scalable inference runtime performance, asymptotically using *constant* memory and time per step with respect to the sequence length. While LSTMs and other RNNs also have this property, empirically, S4 models are far more performant while also being *parallelisable across the sequence dimension* during training.

While inference-time is normally not included when evaluating on sequence modelling benchmarks, it has a large impact on the scalability and wallclock-time for reinforcement learning (RL) because the agent uses inference to collect data from the environment. Thus, transformers usually have poor runtime performance in reinforcement learning (Parisotto & Salakhutdinov, 2021). While transformers have become the default architecture for many supervised sequence-modelling tasks (Vaswani et al., 2017), RNNs are still widely-used for memory-based RL tasks (Ni et al., 2022).

The ability to efficiently model contexts that are orders of magnitude larger may enable new possibilities in RL. This is particularly applicable in meta-reinforcement learning (Meta-RL), in which the agent is trained to adapt across *multiple* environment episodes. One approach to Meta-RL, RL^2 (Duan et al., 2016; Wang et al., 2016), uses sequence models to directly learn across these episodes, which can often result in trajectories that are thousands of steps long. Most instances of RL^2 approaches, however, are limited to narrow task distributions and short adaptation horizons because of their limited effective memory length and slow training speeds.

Unfortunately, simply applying S4 models to reinforcement learning is challenging. This is because the most popular

[*]Equal contribution [1]Department of Engineering Sciences, University of Oxford, Oxford, United Kingdom [2]DeepMind. Correspondence to: Chris Lu <christopher.lu@exeter.ox.ac.uk>.

Workshop on New Frontiers in Learning, Control, and Dynamical Systems at the International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA, 2023. Copyright 2023 by the author(s).

	Inference	Training	Parallel	Variable Lengths	bsuite Score
RNNs	**O**(1)	**O**(L)	No	**Yes**	No
Transformers	O(L^2)	O(L^2)	**Yes**	**Yes**	**Yes**
S5 with $\bullet$	**O**(1)	**O**(L)	**Yes**	No	N/A
S5 with $\oplus$	**O**(1)	**O**(L)	**Yes**	**Yes**	**Yes**

Table 1. The different properties of the different architectures. The asymptotic runtimes are in terms of the sequence length L assume a constant hidden size. The bsuite scores correspond to whether or not they achieve a perfect score in the median runs on the bsuite memory length environment.

training paradigm in RL involves collecting fixed-length environment trajectories, which often cross episode boundaries. RNNs handle episode boundaries by resetting the hidden state at those transitions when performing backpropagation through time. Unlike RNNs, S4 models cannot simply reset their hidden states within the sequence because they train using a fixed convolution kernel instead of using backpropagation through time.

A recent modification to S4, called Simplified Structured State Space Sequence Models (S5), replaces this convolution with a *parallel scan* operation (Smith et al., 2022), which we describe in Section 2. In this paper, we propose a modification to S5 that enables resetting its hidden state within a trajectory during the training phase, which in turn allows practitioners to simply replace RNNs with S5 layers in existing frameworks. We then demonstrate S5's performance and runtime properties on the simple bsuite memory-length task (Osband et al., 2019), showing that S5 achieves a higher score than RNNs while also being nearly two times faster when using their provided baseline algorithm. We also evaluate our architecture on the environments in the recently-proposed Partially Observable Process Gym (POPGym) suite (Morad et al., 2023) and show that S5 outperforms GRU's while also running over six times faster, achieving state-of-the-art results on the "Repeat Hard" task, which all other architectures previously struggled to solve. Finally, we evaluate S5 on a challenging Meta-RL task in which the environment samples a random DMControl environment (Tassa et al., 2018) and a random linear projection of the state and action spaces at the beginning of each episode. We show that the S5 agent achieves higher returns than LSTMs in this setting. Furthermore, we demonstrate that the resulting S5 agent performs well even on random linear projections of the state and action spaces of out-of-distribution held-out tasks.

2. Background

2.1. Structured State Space Sequence Models

State Space Models (SSMs) have been widely used to model various phenomenon using first-order differential equations (Hamilton, 1994). At each timestep t, these models take an input signal $u(t)$. This is used to update a latent state $x(t)$ which in turn computes the signal $y(t)$. Some of the more widely-used canonical SSMs are continuous-time linear SSMs, which are defined by the following equations:

$$\begin{aligned} \dot{x}(t) &= \mathbf{A}x(t) + \mathbf{B}u(t) \\ y(t) &= \mathbf{C}x(t) + \mathbf{D}u(t) \end{aligned} \tag{1}$$

where $\mathbf{A}, \mathbf{B}, \mathbf{C}$, and $\mathbf{D}$ are matrices of appropriate sizes. To model sequences with a fixed step size Δ, one can *discretise* the SSM using various techniques, such as the zero-order hold method, to obtain a simple linear recurrence:

$$\begin{aligned} x_n &= \bar{\mathbf{A}}x_{n-1} + \bar{\mathbf{B}}u_n \\ y_n &= \bar{\mathbf{C}}x_n + \bar{\mathbf{D}}u_n \end{aligned} \tag{2}$$

where $\bar{\mathbf{A}}, \bar{\mathbf{B}}, \bar{\mathbf{C}}$, and $\bar{\mathbf{D}}$ can be calculated as functions of $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$, and Δ.

S4 (Gu et al., 2021a) proposed the use of SSMs for modelling long sequences and various techniques to improve its stability, performance, and training speeds when combined with deep learning. For example, S4 models use a special matrix initialisation to better preserve sequence history called HiPPO (Gu et al., 2020).

One of the primary strengths of the S4 model is that it can be converted to both a recurrent model, which allows for fast and memory-efficient inference-time computation, and a convolutional model, which allows for efficient training that is *parallelisable across timesteps* (Gu et al., 2021b).

More recently, Smith et al. (2022) proposed multiple simplifications to S4, called S5. One of its contributions is the use of *parallel scans* instead of convolution, which vastly simplifies S4's complexity and enables more flexible modifications. Parallel scans take advantage of the fact that the composition of *associative* operations can be computed in any order. Recall that for an operation $\bullet$ to be associative, it must satisfy $(x \bullet y) \bullet z = x \bullet (y \bullet z)$.

Given an associative binary operator $\bullet$ and a sequence of length N, parallel scan returns:

$$[e_1, e_1 \bullet e_2, \cdots, e_1 \bullet e_2 \bullet \cdots \bullet e_N] \tag{3}$$

For example, when $\bullet$ is addition, the parallel scan calculates the prefix-sum, which returns the running total of an input

sequence. Parallel scans can be computed in $O(\log(N))$ time when given a sequence of length N, given N parallel processors.

S5's parallel scan is applied to initial elements $e_{0:N}$ defined as:

$$e_k = (e_{k,a}, e_{k,b}) := (\bar{\mathbf{A}}, \bar{\mathbf{B}} u_k) \tag{4}$$

Where $\bar{\mathbf{A}}$, $\bar{\mathbf{B}}$, and u_k are defined in Equation 2. S5's parallel operator is then defined as:

$$a_i \bullet a_j = (a_{j,a} \odot a_{i,a}, a_{j,a} \otimes a_{i,b} + a_{j,b}) \tag{5}$$

where $\odot$ is matrix-matrix multiplication and $\otimes$ is matrix-vector multiplication. The parallel scan then generates the recurrence in the hidden state x_n defined in Equation 2.

$$e_1 = (\bar{\mathbf{A}}, \bar{\mathbf{B}} u_1) \qquad = (\bar{\mathbf{A}}, x_1) \tag{6}$$

$$e_1 \bullet e_2 = (\bar{\mathbf{A}}^2, \bar{\mathbf{A}} x_1 + \bar{\mathbf{B}} u_2) = (\bar{\mathbf{A}}^2, x_2) \tag{7}$$

$$e_1 \bullet e_2 \bullet e_3 = (\bar{\mathbf{A}}^2, \bar{\mathbf{A}} x_2 + \bar{\mathbf{B}} u_3) = (\bar{\mathbf{A}}^3, x_3) \tag{8}$$

Note that the model assumes a hidden state initialised to $x_0 = 0$ by initialising the scan with $e_0 = (\mathbf{I}, 0)$.

2.2. Reinforcement Learning

A Markov Decision Process (MDP) (Sutton & Barto, 2018) is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, R, P, \gamma \rangle$, which defines the environment. Here, $\mathcal{S}$ is the set of states, $\mathcal{A}$ the set of actions, R the reward function that maps from a given state and action to a real value $\mathbb{R}$, P defines the distribution of next-state transitions given a state and action, and γ defines the discount factor. The agent's objective is to find a policy π (a function which maps from a given state to a distribution over actions) which maximises the expected discounted sum of returns.

$$\mathbb{E}[\mathrm{R}^\gamma | \pi] = \mathbb{E}_{s_0 \sim d, a_{0:\infty} \sim \pi, s_{1:\infty} \sim P} \Big[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \Big]$$

In a Partially-Observed Markov Decision Process (POMDP), the agent receives an observation $o_t \sim O(s_t)$ instead of directly observing the state. Because of this, the optimal policy π^* does not depend just on the current observation o_t, but (in generality) also on all previous observations $o_{0:t}$ and actions $a_{0:t}$.

3. Method

We first modify to S5 to handle variable-length sequences, which makes the architecture more suitable for tackling POMDPs. We then introduce a challenging new Meta-RL setting that tests for broad generalisation capabilities.

Algorithm 1 Pseudocode for the Multi-Environment Meta-Learning environment step.

Require: Distribution of environments $E \sim \rho_E$, a fixed output observation dimension size O, and a fixed action dimension size A. Agent action a and Environment termination d

function StepEnvironment(a, d)
  if the environment terminated (d) **then**
    Sample random environment $E \sim \rho_E$
    Initialise random observation projection matrix $M_o \in \mathbb{R}^{E^o \times O}$ where E^o is E's observation size
    Initialise random action projection matrix $M_a \in \mathbb{R}^{A \times E^a}$ where E^a is E's action size
    Reset E to receive an initial observation o
    Apply the random observation projection matrix to the observation $o' = M_o o$
    Append $r = 0$ and $d = 0$ to o' to get o''
    Return o''
  else
    Apply the projection matrix $a' = M_a a$
    Step E using a' to receive the next observation o, reward r, and done signal d.
    Apply the projection matrix $o' = M_o o$
    Append r and d to o' to get o''
    Return o'', r, and d
  end if
end function

3.1. Resettable S5

Implementations of RL algorithms commonly collect fixed-length trajectory "rollouts" from the environment for training, despite the fact that the environment episode lengths are often far longer and vary significantly. Thus, the collected rollouts (1) often begin within an episode and (2) may contain episode boundaries.

To handle trajectory rollouts that begin in the middle of an episode, sequence models must be able to access the state of memory that was present prior to the rollout's collection. Usually, this is done by storing the RNN's hidden state at the beginning of each rollout to perform truncated backpropagation through time (Williams & Zipser, 1995). This is challenging to do with transformers because they do not normally have an explicit recurrent hidden state, but instead simply retain the entire history during inference time. This is similarly challenging for S4 models since they assume that all hidden states are initialised identically to perform a more efficient backwards pass.

To handle episode boundaries within the trajectory rollouts, memory-based models must be able to reset their hidden state, otherwise they would be accessing memory and context from other episodes. RNNs can trivially reset their hid-

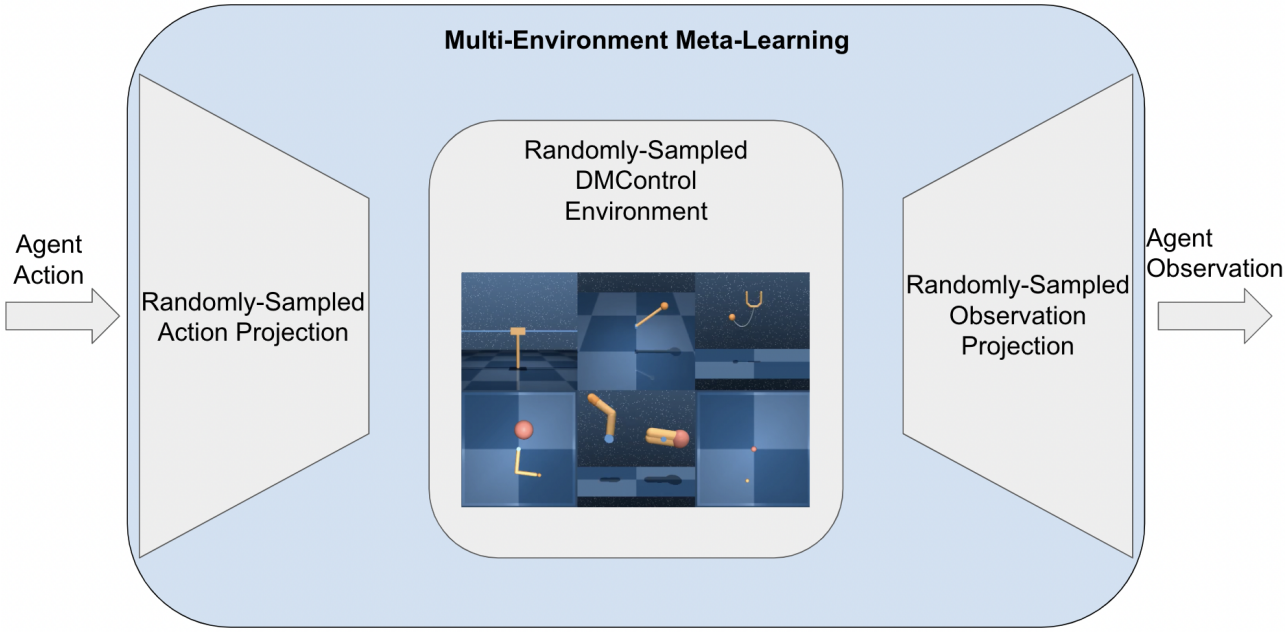


Figure 1. An overview of multi-environment meta-learning with random projections. At the start of each meta-episode, we sample a random observation projection, action projection, and DMControl environment.

den state when performing backpropagation through time, and transformers can mask out the invalid transitions. However, S4 models have no such mechanism to do this.

To resolve both of these issues, we modify S5's associative operator to include a reset flag that preserves the associative property, allowing S5 to efficiently train over sequences of varying lengths and hidden state initialisations. We create a new associative operator $\oplus$ that operates on elements e_k defined:

$$e_k = (e_{k,a}, e_{k,b}, e_{k,c}) := (\bar{\mathbf{A}}, \bar{\mathbf{B}}u_k, d_k) \tag{9}$$

where $d_k \in \{0, 1\}$ is the binary "done" signal for the given transition from the environment.

We define $\oplus$ to be:

$$a_i \oplus a_j := \begin{cases} (a_{j,a} \odot a_{i,a}, a_{j,a} \otimes a_{i,b} + a_{j,b}, a_{i,c}) & \text{if } a_{j,c} = 0 \\ (a_{j,a}, a_{j,b}, a_{j,c}) & \text{if } a_{j,c} = 1 \end{cases}$$

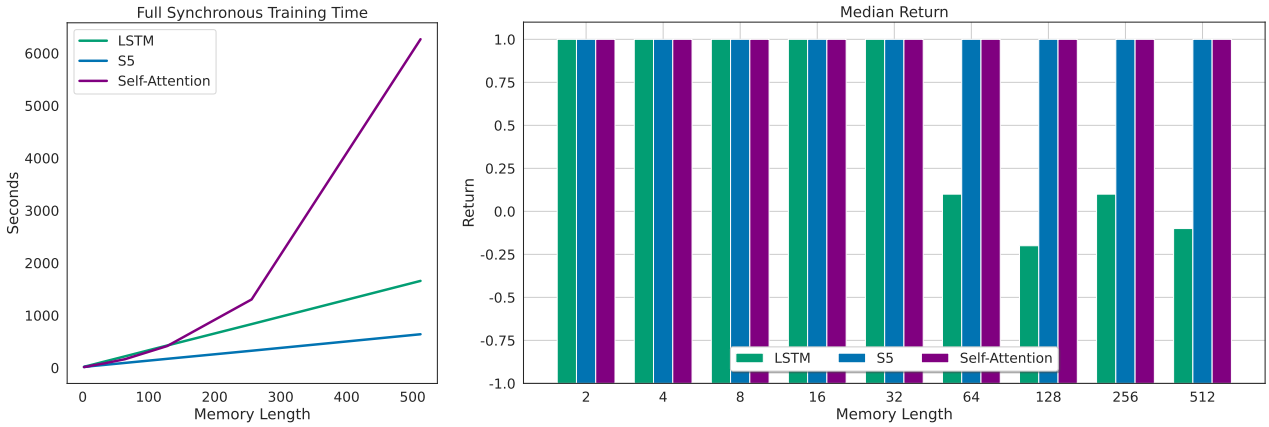
We prove that this operator is associative in Appendix A. We now show that the operator computes the desired value. Assuming $e_{n,c} = 1$ corresponds to a "done" transition while all other timesteps before it ($e_{0:n-1,c} = 0$) and after it ($e_{n+1:L,c} = 0$) do not, we see:

$$\begin{aligned} e_0 \oplus \cdots \oplus e_{n-1} &= (\bar{\mathbf{A}}^{n-1}, \bar{\mathbf{A}}x_{n-2} + \bar{\mathbf{B}}u_{n-1}, 0) \\ &= (\bar{\mathbf{A}}^{n-1}, x_{n-1}, 0) \\ e_0 \oplus \cdots \oplus e_n &= (\bar{\mathbf{A}}, \bar{\mathbf{B}}u_n, 1) \\ &= (\bar{\mathbf{A}}, x_n, 1) \\ e_0 \oplus \cdots \oplus e_{n+1} &= (\bar{\mathbf{A}}^2, \bar{\mathbf{A}}x_n + \bar{\mathbf{B}}u_{n+1}, 1) \\ &= (\bar{\mathbf{A}}^2, x_{n+1}, 1) \end{aligned}$$

Note that even if there are multiple "resets" within the rollout, the desired value will still be computed.

3.2. Multi-Environment Meta-Learning with Random Projections

Most prior work in Meta-RL has only demonstrated the ability to *adapt* to a small range of similar tasks (Beck et al., 2023). Furthermore, the action and observation spaces usually remain identical across different tasks, which severely limits the diversity of meta-learnning environments. To achieve more general meta-learning, ideally the agent should learn from tasks of varying complexity and dynamics. Inspired by Kirsch et al. (2022b), we propose taking *random linear projections* of the observation space and action space to a fixed size, allowing us to use the same model for tasks of varying observation and action space sizes. Furthermore, randomised projections *vastly* increase the number of tasks



(a) (b)

Figure 2. Results in bsuite's Memory Length task. Self-attention achieves perfect performance because it is given access to the entire sequence, but shows poor asymptotic performance in runtime. Meanwhile, LSTMs run more quickly, but show poor performance. S5 achieves a high performance while running almost twice as quickly as LSTMs. Experiments were performed on a single NVIDIA V100.

in the meta-learning space. We can then evaluate the ability of our model to *generalise* to unseen held-out tasks.

We provide pseudocode for the environment implementation in Algorithm 1 and a visualisation explaining the setup in Figure 1.

4. Experiments

4.1. Memory Length Environment

First, we demonstrate our modified S5's improved training speeds in performance in the simple memory length environment proposed in bsuite (Osband et al., 2019).

The environment is based on the well-known 't-maze' environment (O'Keefe & Dostrovsky, 1971) in which the agent receives a cue on the first timestep, which corresponds to the action the agent should take some number of steps in the future to receive a reward. We run our experiments using bsuite's actor-critic baseline while swapping out the LSTM for Transformer self-attention blocks or S5 blocks (Vaswani et al., 2017) and using Gymnax for faster environment rollouts (Lange, 2022).

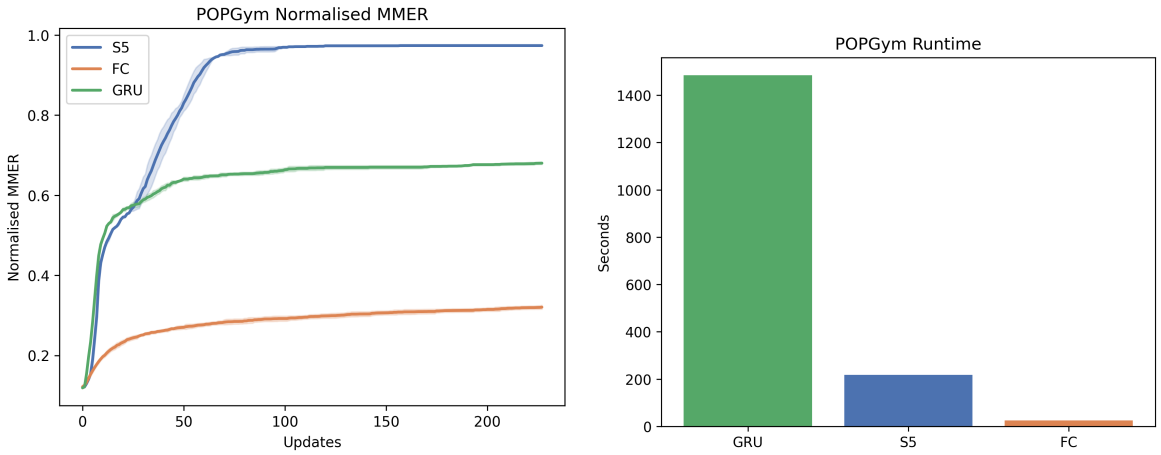
We show the results in Figure 2. In general, S5 displays a better asymptotic runtime than Transformers while far outperforming LSTMs in both performance and speed. Note that while S5 is theoretically $O(\log(N))$ in the backwards pass during training time (given enough processors), it is still bottle-necked by rollout collection from the environment, which takes $O(N)$ time. Because of the poor runtime performance of transformers for long sequences, we did not collect results for them in the following experiments.

4.2. POPGym Environments

We evaluate our S5 architecture on environments from the Partially Observable Process Gym (POPGym) (Morad et al., 2023) suite, a set of simple environments designed to benchmark memory in deep RL. To increase experiment throughput on a limited compute budget, we carefully re-implemented environments from the POPGym suite entirely in JAX (Bradbury et al., 2018) by leveraging existing implementations of CartPole and Pendulum in Gymnax (Lange, 2022). PyTorch does not support the associative scan operation, so we could directly use the S5 architecture in POPGym's RLLib benchmark.

Morad et al. (2023) evaluated several architectures and found the Gated Recurrent Unit (GRU) (Cho et al., 2014) to be the most performant. Thus, we compare our results to the GRU architecture proposed in the original POPGym benchmark. Note that POPGym's reported results use RLLib's (Liang et al., 2018) implementation of PPO, which makes several non-standard code-level implementation decisions. For example, it uses a dynamic KL-divergence coefficient on top of the clipped surrogate objective of PPO (Schulman et al., 2017) – a feature that does not appear in most PPO implementations (Engstrom et al., 2020). We instead use a recurrent PPO implementation that is more closely aligned with StableBaselines3 (Raffin et al., 2021) and CleanRL's (Huang et al., 2022) recurrent PPO implementations. We include more discussion and the hyperparameters in Appendix B.

We show the results in Figure 3 and Table 2. We also include more results in the Appendix. Notably, the S5 architecture performs well on the challenging "Repeat Previous Hard" task, far outperforming all architectures tested in Morad



(a) (b)

Figure 3. (a) Results across implemented environments in POPGym's suite. Scores are normalised by the max-MMER per-environment. The shaded region represents the standard error of the mean across eight seeds. (b) The runtime for a single seed averaged across the environments for each architecture. Note that our implementation is end-to-end compiled to run entirely on a single NVIDIA A40.

	Stateless CartPole Hard	Noisy Stateless CartPole Hard	Stateless Pendulum Hard	Noisy Stateless Pendulum Hard	Repeat Previous Hard
S5 with $\oplus$	$\mathbf{1.0 \pm 0.0}$	$\mathbf{0.28 \pm 0.0}$	$\mathbf{0.79 \pm 0.01}$	0.55 ± 0.01	$\mathbf{0.91 \pm 0.04}$
S5 with $\bullet$	$\mathbf{1.0 \pm 0.0}$	$\mathbf{0.28 \pm 0.0}$	0.71 ± 0.04	0.5 ± 0.01	0.76 ± 0.02
GRU	$\mathbf{1.0 \pm 0.0}$	0.27 ± 0.0	0.75 ± 0.0	$\mathbf{0.61 \pm 0.01}$	-0.45 ± 0.02
MLP	0.26 ± 0.0	0.22 ± 0.0	0.41 ± 0.02	0.34 ± 0.01	-0.48 ± 0.00

Table 2. Results in POPGym's suite. The reported number is the max-mean episodic reward (MMER) used in Morad et al. (2023). To calculate this, we take the mean episodic reward for each epoch, and then take the maximum over all epochs. The mean and standard deviation across eight seeds are reported.

et al. (2023). Furthermore, the S5 architecture also runs over six times faster than the GRU architecture.

4.3. Multi-Environment Meta-Reinforcement Learning

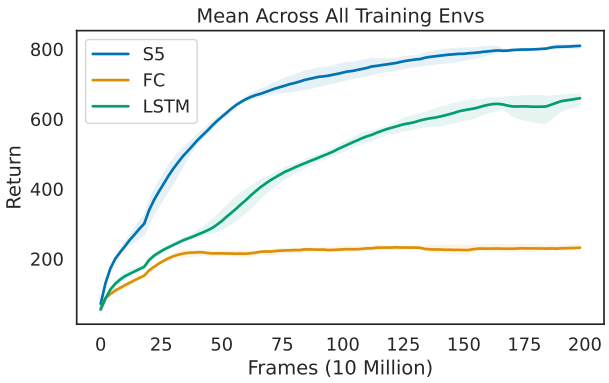


Figure 5. The mean of the return across all of the training environments. The shaded regions represent the range of returns reported across the three seeds. The environment observations and actions are randomly projected as described in Algorithm 1.

We run our S5 architecture, an LSTM baseline, and a memory-less fully-connected network in the environment described in Section 3.2. For these experiments, we use Muesli (Hessel et al., 2021) as our policy optimisation algorithm. We randomly project the environment observations to a fixed size of 12 and randomly project from an action space of size 2 to the corresponding environment's action space. We selected all of the DMControl environments and tasks that had observation and action spaces of size equal to or less than those values and split them into train and test set environments.

We use the S5 architecture described in Smith et al. (2022), which consists of multiple stacked layers of S5 blocks. For both S5 and LSTM architectures, we found that setting the trajectory length equal to the maximum episode length $1k$ achieved the best performance in this setting. We provide a more detailed description of our architecture and hyperparameters in Appendix B.

In-Distribution Training Results We meta-train the model across the six DMControl environments in Figure 4 and show the mean performance across them in Figure 5. Note

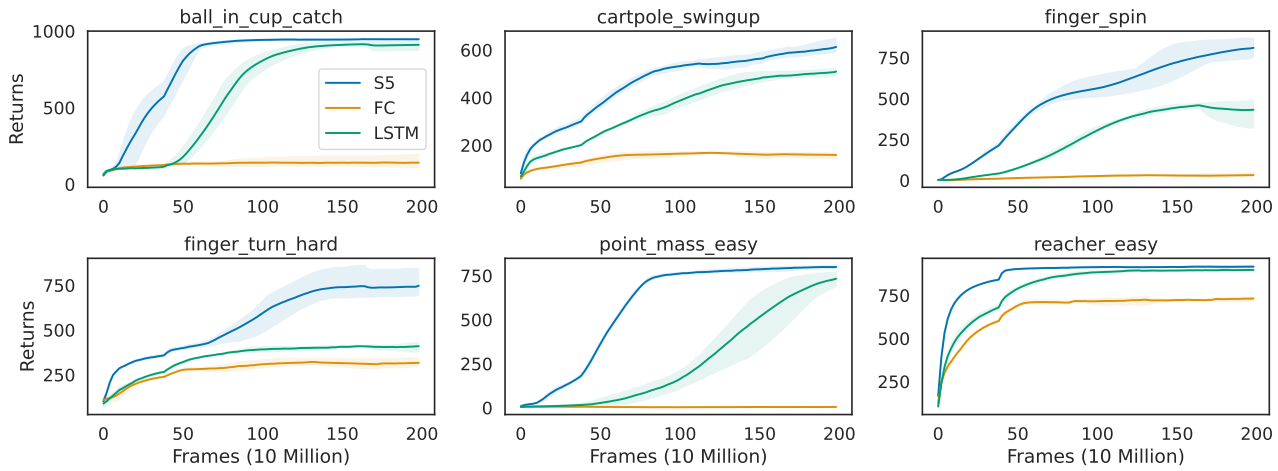


Figure 4. Results across the different environments of the training distribution. The shaded regions represent the range of returns reported across the three seeds. The environment observations and actions are randomly projected as described in Algorithm 1

that while these environments are in the training distribution, they are still being evaluated on *unseen random linear projections* of the state and action spaces and are *not given task labels*. Thus, the agent must infer from the reward and obfuscated dynamics which environment it is in. In this setting, S5 outperforms LSTMs in both sample efficiency and ultimate performance.

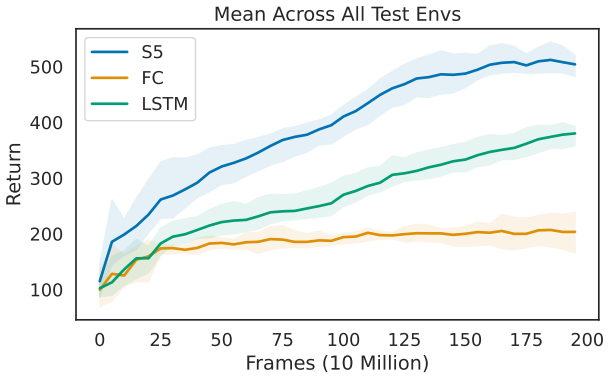


Figure 6. The mean of the return across all of the unseen held-out environments. The shaded regions represent the range of returns reported across the three seeds. The environment observations and actions are randomly projected as described in Algorithm 1.

Out-of-Distribution Evaluation Results: After training the model on the DMControl tasks in Figure 4, we next evaluate the trained model on random linear projections of five *held-out* DMControl tasks. We present the mean score across these tasks in Figure 6 and the results for each task in Figure 7. While the agent displays impressive transfer performance to some tasks with unseen rewards and dynamics, it fails to successfully transfer to a completely unseen task in the test set, pendulum swingup, which has an unseen observation space, action space, and reward dynamics.

5. Related Work

S4 models have previously been shown to work in a number of previous settings, including audio generation (Goel et al., 2022) and video modeling (Nguyen et al., 2022). Concurrent work investigated S4 models in other reinforcement learning settings. David et al. (2023) investigated S4 models in a largely offline RL setting, with some modifications for online finetuning of the model. Morad et al. (2023) investigated sequence models for POMDPs and found that naively using S4 models did not perform well. We include direct performance comparisons to their evaluated architectures, including S4D (Gu et al., 2022), in Appendix C.

Most works in memory-based meta-RL have not focused on generalisation to out-of-distribution tasks, but instead focused on maximising performance on the training task distribution (Duan et al., 2016; Wang et al., 2016; Beck et al., 2023). Many past works that increased generalisation in meta-RL did so by restricting the model architecture class or architecture to symmetric models (Kirsch et al., 2022a), loss functions (Houthooft et al., 2018), target values (Oh et al., 2020), or drift functions (Lu et al., 2022). In contrast, this work achieves generalisation by vastly increasing the task distribution *without limiting the expressivity of the underlying model*, which eliminates the need for hand-crafted restrictions.

Some works perform long-horizon meta-reinforcement learning through the use of evolution strategies (Houthooft et al., 2018; Lu et al., 2022). This is because RNNs and Transformers have historically struggled to model very long

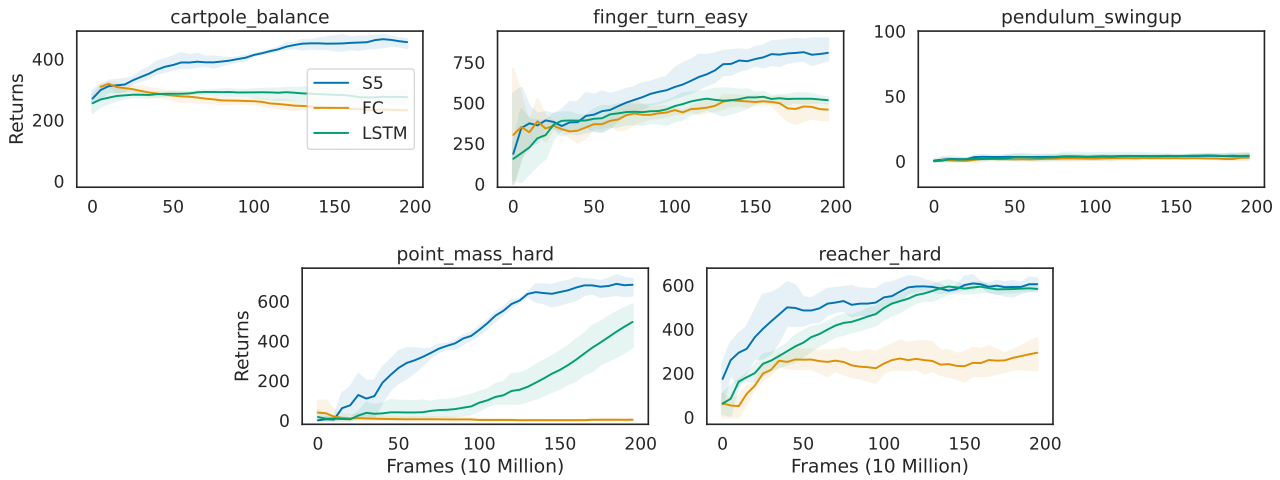


Figure 7. Results when evaluating on held-out DMControl tasks. The shaded regions represent the range of returns reported across the three seeds. The environment observations and actions are randomly projected as described in Algorithm 1

sequences due to computational constraints and vanishing or exploding gradients (Metz et al., 2021). However, evolution strategies are notoriously sample inefficient and computationally expensive.

Other works have investigated different sequence model architectures for memory-based reinforcement learning. Ni et al. (2022) showed that using well-tuned RNNs can be particularly effective compared to many more complicated methods in POMDPs. Parisotto et al. (2020) investigated the use of transformer-based models for memory-based reinforcement learning environments.

Sequence models have also been used for a number of other tasks in RL. For example, Transformers have been used for offline reinforcement learning (Chen et al., 2021), multi-task behavioural cloning (Reed et al., 2022), and algorithm distillation (Laskin et al., 2022). Concurrent work used transformers to also demonstrate out-of-distribution generalisation in meta-reinforcement learning by leveraging a large task space (Adaptive Agent Team et al., 2023).

6. Conclusion and Future Work

In this paper, we investigated the performance of the recently-proposed S5 model architecture in reinforcement learning. S5 models are highly promising for reinforcement learning because of their strong performance in sequence modelling tasks and their fast and efficient runtime properties, with clear advantages over RNNs and Transformers. After identifying challenges in integrating S5 models into existing recurrent reinforcement learning implementations, we made a simple modification to the method that allowed us to reset the hidden state within a training sequence.

We then showed the desirable properties S5 models in the bsuite memory length task. We demonstrated that S5 is *asymptotically* faster than Transformers in the sequence length. Furthermore, we also showed that S5 models run nearly twice as quickly as LSTMs with the same number of parameters while outperforming them. We further evaluated our S5 architecture on environments in the POPGym suite (Morad et al., 2023), where we match or outperform RNNs while also running over six times faster. We achieve strong results in the "Repeat Previous Hard" task, which previous models struggled to solve.

Finally, we proposed a new meta-learning setting in which we meta-learn across random linear projections of the observation and action spaces of randomly sampled DMControl tasks. We show that S5 outperforms LSTMs in this setting. We then evaluate the models on held-out DMControl tasks and demonstrate out-of-distribution performance to unseen tasks through in-context adaptation.

There are several possible ways to further investigate S5 models for reinforcement learning in future work. For one, it may be possible to learn or distill (Laskin et al., 2022) entire reinforcement learning algorithms within an S5 model, given its ability to scale to extremely long contexts. Another direction would be to investigate S5 models for continuous-time RL settings (Doya, 2000). While Δ, the discrete time between timesteps, is fixed for the original S4 model, S5 can in theory use a different Δ for each timestep. Finally, recent work on Linear Recurrent Units (LRU's) applies concepts from S4 and S5, such as linear recurrence, complex parameterisation, and parallel scans, to standard RNN's to achieve similar performance and wallclock time improvements while also simplifying the model (Orvieto et al., 2023). Evaluating LRU's on these tasks could result in simpler architectures with equivalent performance.

References

Adaptive Agent Team, Bauer, J., Baumli, K., Baveja, S., Behbahani, F., Bhoopchand, A., Bradley-Schmieg, N., Chang, M., Clay, N., Collister, A., Dasagi, V., Gonzalez, L., Gregor, K., Hughes, E., Kashem, S., Loks-Thompson, M., Openshaw, H., Parker-Holder, J., Pathak, S., Perez-Nieves, N., Rakicevic, N., Rocktäschel, T., Schroecker, Y., Sygnowski, J., Tuyls, K., York, S., Zacherl, A., and Zhang, L. Human-timescale adaptation in an open-ended task space. *arXiv e-prints*, 2023.

Beck, J., Vuorio, R., Liu, E. Z., Xiong, Z., Zintgraf, L., Finn, C., and Whiteson, S. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

David, S. B., Zimerman, I., Nachmani, E., and Wolf, L. Decision s4: Efficient sequence-based rl via state spaces layers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=kqHkCVS7wbj.

Doya, K. Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245, 2000.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. Rl2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. Implementation matters in deep policy gradients: A case study on ppo and trpo. In *International Conference on Learning Representations*, 2020.

Goel, K., Gu, A., Donahue, C., and Ré, C. It's raw! audio generation with state-space models. *arXiv preprint arXiv:2202.09729*, 2022.

Gu, A., Dao, T., Ermon, S., Rudra, A., and Ré, C. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33: 1474–1487, 2020.

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

Gu, A., Johnson, I., Goel, K., Saab, K., Dao, T., Rudra, A., and Ré, C. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems*, 34: 572–585, 2021b.

Gu, A., Goel, K., Gupta, A., and Ré, C. On the parameterization and initialization of diagonal state space models. *Advances in Neural Information Processing Systems*, 35: 35971–35983, 2022.

Hamilton, J. D. State-space models. *Handbook of econometrics*, 4:3039–3080, 1994.

Hessel, M., Danihelka, I., Viola, F., Guez, A., Schmitt, S., Sifre, L., Weber, T., Silver, D., and Van Hasselt, H. Muesli: Combining improvements in policy optimization. In *International Conference on Machine Learning*, pp. 4214–4226. PMLR, 2021.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Houthooft, R., Chen, Y., Isola, P., Stadie, B., Wolski, F., Jonathan Ho, O., and Abbeel, P. Evolved policy gradients. *Advances in Neural Information Processing Systems*, 31, 2018.

Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., and Araújo, J. G. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274): 1–18, 2022. URL http://jmlr.org/papers/v23/21-1342.html.

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

Kirsch, L., Flennerhag, S., van Hasselt, H., Friesen, A., Oh, J., and Chen, Y. Introducing symmetries to black box meta reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 7202–7210, 2022a.

Kirsch, L., Harrison, J., Sohl-Dickstein, J., and Metz, L. General-purpose in-context learning by meta-learning transformers. *arXiv preprint arXiv:2212.04458*, 2022b.

Lange, R. T. gymnax: A JAX-based reinforcement learning environment library, 2022. URL http://github.com/RobertTLange/gymnax.

Laskin, M., Wang, L., Oh, J., Parisotto, E., Spencer, S., Steigerwald, R., Strouse, D., Hansen, S., Filos, A., Brooks, E., et al. In-context reinforcement learning with algorithm distillation. *arXiv preprint arXiv:2210.14215*, 2022.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., and Stoica, I. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pp. 3053–3062. PMLR, 2018.

Lu, C., Kuba, J. G., Letcher, A., Metz, L., de Witt, C. S., and Foerster, J. Discovered policy optimisation. *arXiv preprint arXiv:2210.05639*, 2022.

Metz, L., Freeman, C. D., Schoenholz, S. S., and Kachman, T. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.

Morad, S., Kortvelesy, R., Bettini, M., Liwicki, S., and Prorok, A. POPGym: Benchmarking partially observable reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=chDrutUTs0K.

Nguyen, E., Goel, K., Gu, A., Downs, G., Shah, P., Dao, T., Baccus, S., and Ré, C. S4nd: Modeling images and videos as multidimensional signals with state spaces. In *Advances in Neural Information Processing Systems*, 2022.

Ni, T., Eysenbach, B., and Salakhutdinov, R. Recurrent model-free rl can be a strong baseline for many pomdps. In *International Conference on Machine Learning*, pp. 16691–16723. PMLR, 2022.

Oh, J., Hessel, M., Czarnecki, W. M., Xu, Z., van Hasselt, H. P., Singh, S., and Silver, D. Discovering reinforcement learning algorithms. *Advances in Neural Information Processing Systems*, 33:1060–1070, 2020.

O'Keefe, J. and Dostrovsky, J. The hippocampus as a spatial map: Preliminary evidence from unit activity in the freely-moving rat. *Brain research*, 1971.

Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gulcehre, C., Pascanu, R., and De, S. Resurrecting recurrent neural networks for long sequences. *arXiv preprint arXiv:2303.06349*, 2023.

Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepesvari, C., Singh, S., et al. Behaviour suite for reinforcement learning. *arXiv preprint arXiv:1908.03568*, 2019.

Parisotto, E. and Salakhutdinov, R. Efficient transformers in reinforcement learning using actor-learner distillation. *arXiv preprint arXiv:2104.01655*, 2021.

Parisotto, E., Song, F., Rae, J., Pascanu, R., Gulcehre, C., Jayakumar, S., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., et al. Stabilizing transformers for reinforcement learning. In *International conference on machine learning*, pp. 7487–7498. PMLR, 2020.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J. T., et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

Smith, J. T., Warrington, A., and Linderman, S. W. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. 2018.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Williams, R. J. and Zipser, D. Gradient-based learning algorithms for recurrent. *Backpropagation: Theory, architectures, and applications*, 433:17, 1995.

A. Proof of Associativity of Binary Operator

Recall that $\oplus$ is defined as:

$$a_i \oplus a_j := \begin{cases} (a_{j,a} \odot a_{i,a}, a_{j,a} \otimes a_{i,b} + a_{j,b}, a_{i,c}) & \text{if } a_{j,c} = 0 \\ (a_{j,a}, a_{j,b}, a_{j,c}) & \text{if } a_{j,c} = 1 \end{cases}$$

This is equivalent to the following:

$$a_i \oplus a_j := \begin{cases} ((a_i \bullet a_j)_a, (a_i \bullet a_j)_b, a_{i,c}) & \text{if } a_{j,c} = 0 \\ a_j & \text{if } a_{j,c} = 1 \end{cases}$$

where $\bullet$ is S5's binary operator defined in Equation 5. Note that $\bullet$'s associativity was proved in Smith et al. (2022). Using this, we can prove the associativity of $\oplus$.

Let x, y, and z refer to three elements. We will prove that for all possible values of x, y, and z, $\oplus$ retains associativity.

Case 1: $z_c = 1$

$$\begin{aligned} (x \oplus y) \oplus z &= z && (10) \\ &= y \oplus z && (11) \\ &= x \oplus (y \oplus z) && (12) \end{aligned}$$

Case 2: $z_c = 0$ and $y_c = 1$

$$\begin{aligned} (x \oplus y) \oplus z &= y \oplus z && (13) \\ \text{Note that } (y \oplus z)_c &= 1 \text{ thus,} && (14) \\ &= x \oplus (y \oplus z) && (15) \end{aligned}$$

Case 3: $z_c = 0$ and $y_c = 0$

$$\begin{aligned} (x \oplus y) \oplus z &= ((x \bullet y)_a, (x \bullet y)_b, x_c) \oplus z && (16) \\ &= (((x \bullet y) \bullet z)_a, ((x \bullet y) \bullet z)_b, x_c) && (17) \\ &= ((x \bullet (y \bullet z))_a, (x \bullet (y \bullet z))_b, x_c) && (18) \\ &= x \oplus ((y \bullet z)_a, (y \bullet z)_b, y_c) && (19) \\ &= x \oplus (y \oplus z) && (20) \end{aligned}$$

B. Hyperparameters

Table 3. Hyperparameters for training A2C on Bsuite

Parameter	Value
Adam Learning Rate	3e-4
Entropy Coefficient	0.0
Encoder Layer Sizes	[256, 256]
Number of Recurrent Layers	1
Size of Recurrent Layer	256
Discount γ	0.99
TD λ	0.9
Number of Environments	1
Unroll Length	32
Number of Episodes	10000
Activation Function	ReLU

Table 4. Hyperparameters for training PPO on POPGym

Parameter	Value
Adam Learning Rate	5e-5
Number of Environments	64
Unroll Length	1024
Number of Timesteps	15e6
Number of Epochs	30
Number of Minibatches	8
Discount γ	0.99
GAE λ	1.0
Clipping Coefficient ϵ	0.2
Entropy Coefficient	0.0
Value Function Weight	1.0
Maximum Gradient Norm	0.5
Learning Rate Annealing	None
Activation Function	LeakyReLU
Encoder Layer Sizes	[128, 256]
Recurrent Layer Hidden Size	256
Action Decoder Layer Sizes	[128, 128]
Value Decoder Layer Sizes	[128, 128]
S5 Layers	4
S5 Hidden Size	256
S5 Discretization	ZOH
S5 Δ min	0.001
S5 Δ max	0.1

Table 5. Hyperparameters for training Muesli on Multi-Environment Meta-RL. These experiments were run using 64 TPUv3's.

Parameter	Value
Adam Learning Rate	3e-4
Value Function Weight	1.0
Muesli Auxiliary Loss Weight	3.0
Muesli Model Unroll Length	1.0
Encoder Layer Sizes	[512, 512]
Number of Environments	1024
Discount γ	0.995
Rollout Length	1000
Offline Data Fraction	0.0
Total Frames	2e9
LSTM Hidden Size	512
Projected Observation Size	12
Projected Action Size	2
S5 Layers	10
S5 Hidden Size	256
S5 Discretization	ZOH
S5 Δ min	0.001
S5 Δ max	0.1

C. POPGym Discussion

	Stateless CartPole Hard	Noisy Stateless CartPole Hard	Stateless Pendulum Hard	Noisy Stateless Pendulum Hard	Repeat Previous Hard
S5 (ours)	$\mathbf{1.0 \pm 0.0}$	$\mathbf{0.28 \pm 0.0}$	$\mathbf{0.79 \pm 0.01}$	0.55 ± 0.01	$\mathbf{0.91 \pm 0.01}$
GRU (ours)	$\mathbf{1.0 \pm 0.0}$	0.27 ± 0.0	0.75 ± 0.0	$\mathbf{0.61 \pm 0.01}$	-0.46 ± 0.01
MLP (ours)	0.26 ± 0.0	0.22 ± 0.0	0.41 ± 0.02	0.34 ± 0.01	-0.48 ± 0.00
GRU	$\mathbf{1.000 \pm 0.000}$	0.390 ± 0.007	$\mathbf{0.828 \pm 0.001}$	$\mathbf{0.657 \pm 0.002}$	-0.428 ± 0.002
MLP	0.265 ± 0.002	0.229 ± 0.002	0.477 ± 0.030	0.351 ± 0.012	-0.486 ± 0.002
IndRNN	$\mathbf{1.000 \pm 0.000}$	$\mathbf{0.404 \pm 0.005}$	0.804 ± 0.023	0.521 ± 0.109	-0.384 ± 0.013
LMU	0.987 ± 0.007	0.352 ± 0.019	0.806 ± 0.006	0.563 ± 0.014	$\mathbf{0.191 \pm 0.041}$
S4D	0.127 ± 0.026	0.207 ± 0.007	0.303 ± 0.014	0.289 ± 0.011	-0.491 ± 0.001
FART	$\mathbf{0.996 \pm 0.000}$	0.366 ± 0.002	0.698 ± 0.077	0.553 ± 0.007	-0.485 ± 0.001

Table 6. Results in POPGym's suite. The reported number is the max-mean episodic reward (MMER) used in Morad et al. (2023). To calculate this, we take the mean episodic reward for each epoch, and then take the maximum over all epochs. For our results above, the mean and standard deviation across eight seeds are reported. The results below are selected architectures from Morad et al. (2023), which also includes the best-performing one from each environment. They report the mean and standard deviation across three trials.

Morad et al. (2023) used RLLib's (Liang et al., 2018) implementation of PPO, which differs significantly from standard implementations of PPO. It uses a dynamic KL-divergence coefficient on top of the clipped surrogate objective of PPO (Schulman et al., 2017). Furthermore, they use advanced orchestration to return full episode trajectories, rather than using the more commonly-studied "stored state" (Kapturowski et al., 2018) strategy.

Instead, we follow the design decisions outlined in the StableBaselines3 (Raffin et al., 2021) and CleanRL's (Huang et al., 2022) recurrent PPO implementations. While this results in different results shown in Table 6 for the same architecture, it recovers similar performance across the environments. Notably, our S5 architecture far outperforms the best performing architecture in Morad et al. (2023) in the "RepeatPreviousHard" environment.

We used the learning rate, number of environments, unroll length, timesteps, epochs, and minibatches, GAE λ, and model architectures from Morad et al. (2023). However, we used the standard clipping coefficient ϵ of 0.2 instead of 0.3 to account for the lack of a dynamic KL-divergence coefficient. Note that we also adjusted the S5 architecture to contain approximately the same number of *parameters* as the GRU implementation instead of matching the size of the *hidden state*, which was done in Morad et al. (2023).

We did not evaluate our architecture across the full POPGym suite. To enable more rapid experimentation, we implement our algorithms and environments end-to-end in JAX (Bradbury et al., 2018; Lu et al., 2022). While the original POPGym results took 2 hours per trial with a GRU with a Quadro RTX 8000 and 24 CPUs, we could run our experiments using only 3 *minutes* per trial on an NVIDIA A40. Because of this, we selected environments from Morad et al. (2023) to implement in JAX. We chose the CartPole, Pendulum, and Repeat environments because they are modified versions of existing environments in Lange (2022). We found that the "Easy" and "Medium" versions of these environments were not informative, as most models perform well on them and only report the "Hard" difficulty results.